CHAPTER 3

Performance Comparison of LDA and PCA for Classification of Gases/Odors

This chapter presents the performance comparison of linear discriminant analysis (LDA) and principal component analysis (PCA) as feature extraction tools for classification of gases/odors using responses obtained from the thick film sensor array. The PCA and the LDA have been used for feature extraction and dimensionality reduction in various fields of research. Present work compares their ability for classification of gases/odors. The results of the experiment performed for gases/odors classification show that PCA can be preferred for data preprocessing and dimensionality reduction due to its unsupervised nature and promising results obtained.

3.1 Introduction

The identification of gases/odors using thick film tin oxide sensor array has been a topic of keen interest for the ongoing researches in the field of e-nose [Watson (1984); Chaturvedi *et al.* (1999); Srivastava *et al.* (1999); Chaturvedi *et al.* (2000)]. Thick film gas sensors have proved their ability to identify various gases/odors. The doped tin oxide thick film gas sensors have been reported in the recent past for correct identification of different gases/odors. The sensitivity and selectivity of these sensors are improved by doping the intrinsic tin oxide layer with suitable dopants like Pd, Pt, ZnO etc. [Chaturvedi *et al.* (1999); Watson (1984)]. An array of such sensor elements is further used and preferred in varieties of gases/odors sensing systems since the sensor array carry more information [Srivastva *et al.* (1999)]. The information redundancy in sensor array response provides a unique signature pattern associated with a gas which greatly improves the classification results, in contrast to single sensor element system [Srivastva (2003)]. The sensor array along with signal processing techniques forms the inherent parts of electronic-nose (e-nose) which is a challenging research field in the gas sensing area**.**

The preprocessing and feature extraction from the original raw data is one of the important steps for pattern recognition methods which are utilized in e-nose

systems. The goal of preprocessing and feature extraction is to transform the data in such a form that it show more cluster separation among different classes and/or to transform the high dimensional data into low dimensional data still preserving the most discriminative information in the original raw dataset [Osuna (2002); Srivastava (2003); Kumar *et al.* (2009 (a)); Rajput *et al.* (2011)].

Principal component analysis (PCA) and linear discriminant analysis (LDA) are two of the most important techniques which are generally utilized for feature extraction and dimensionality reduction. The PCA has been used in many classification problems like face recognition, gases/odors discrimination and image text recognition problems etc. LDA has also been utilized for different pattern recognition problems [Kirby and Sirovich (1990); Belhumeour *et al.* (1997); Rajput *et al.* (2010); Capone *et al.* (2001)]. In the present study, these two techniques have been used for gases/odors identification tasks.

The LDA technique, which is also called Fisher Discriminant Analysis (FDA), is a supervised method of feature extraction since it takes class information into account. The PCA, on the other hand, is an unsupervised feature extraction scheme which does not take any class information into account [Na *et al.* (2010); Heo and Gader (2011)]. The LDA models the difference between the classes of data while PCA does not consider any difference in class. Thus, it is expected that LDA algorithms could be better as compared to PCA algorithms for classification problems. This could be true for large as well as medium training datasets [Martinez and Kak (2001)].

In the present study, the performance of LDA and PCA methods have been ensured using two types of datasets. First dataset is the published steady state responses of thick film sensor array taken by [Mishra and Agarwal (1998)]. Second dataset is the steady state responses generated from the thick film sensor array fabricated by the author. A simple multilayer feed-forward neural network is designed as classifier in the subsequent classification stage. This classifier network is trained and tested with PCA as well as with LDA transformed data. The performance of LDA and PCA transformed data have been tested for various gases/odors classification described in next sections.

The present chapter has been divided into 4 sections. The section 3.2 deals with the brief experimental background and data extraction method. Section 3.3

includes the results and discussion part. Section 3.4 concludes the findings of the experiment.

3.2 Materials and Methods

3.2.1 Experimental Background

As mentioned previously, two types of datasets have been used for the present study. The first dataset is the sensor array (denoted as Sensor Array-1) responses for various gases viz. LPG, CH_4, CO and H_2 taken by [Mishra and Agarwal (1998)]. The schematic of fabricated gas sensor array is shown in Fig. 3.1.

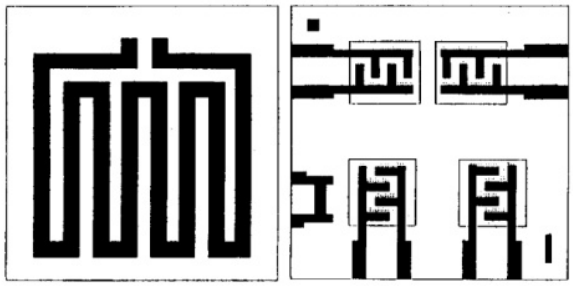

Fig. 3.1 Schematic of Sensor Array-1(heater and electrode sensors) fabricated by Mishra and Agarwal (1998)

It consisted of a gas sensitive layer (SnO_2), a pair of electrodes in array form underneath the gas sensing layers serving as a contact pad for sensors and a common heater element on the back side of the substrate. A temperature sensor adjacent to the gas sensor was also incorporated to measure the sensor temperature. Alumina substrate (96%) was used as a substrate for sensor fabrication. The thermistor pattern was screen printed first (paste NTC 2413 ESL), dried at a temperature of 100-125°C and fired at 950°C. In the second step, finger electrode patterns (in array form) and thermistor contact pad were screen printed using gold conductor paste (No. 5754 B Heraeus, Gmbh) and dried at a temperature of 100-125°C. Subsequently, a heater element was screen printed on the back side of the substrate using silver palladium conductor paste (No. C1214, Heraeus, Gmbh) dried at the same temperature. The dried screen printed films were fired at 850°C. In the third step, doped and undoped tin oxide pastes were screen printed over the electrode patterns and fired at 550°C after drying the print at 125°C for 15 min.

These doped pastes were prepared by adding 1% Pd, Pt and Au by weight in available SnO_2 paste (Type 3050, Electro Science Lab.) and subsequently ball milled for about 6 hours. The fabricated sensor array was tested for varying concentrations of LPG, H_2, CH_4 and CO, in a locally developed test chamber. The test chamber consisted of a glass chamber having volume 2047 cm^3 kept at metal base which rests on four brass legs. The upper opening of the chamber was also covered by a metal plate. The necessary provisions for electrical connections, gas inlet and outlet to the chamber were provided for sensor characterization in the presence of test gases. A small (5 V) d.c. fan was fitted in the chamber for stirring the gas to maintain the homogenous gas concentration in the chamber. The responses of the gas sensor array using these four gases have been shown in Fig. 3.2.

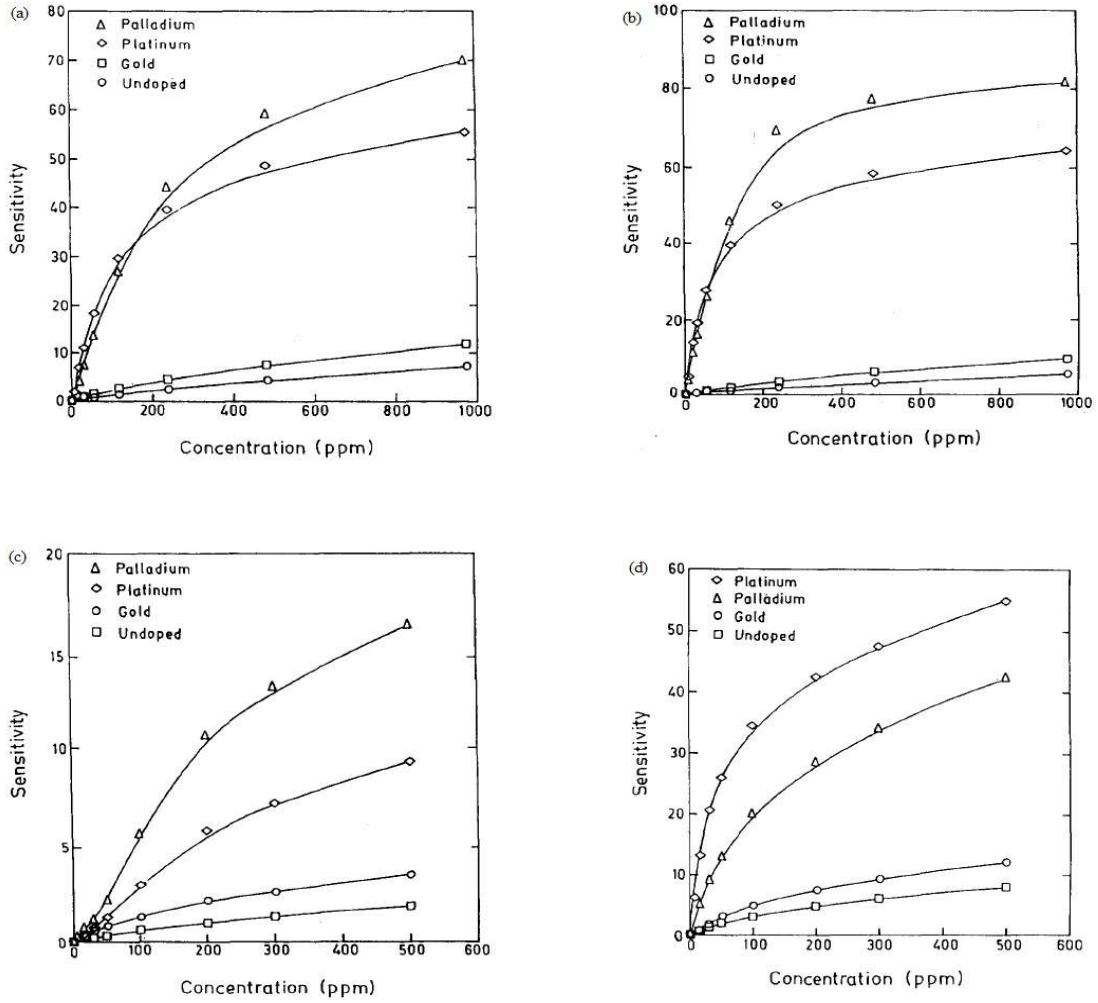

Fig. 3.2 Steady state responses of Sensor Array-1 upon exposure to (a) LPG, (b) H_2 (c) CH_4 and (d) CO [Mishra and Agarwal(1998)]

The percentage change in resistance of the sensor considered as the sensitivity of the sensor as defined previously in Section 2.5.1 of Chapter-2. Further, experimental details can be obtained in [Mishra and Agarwal (1998)]

The second dataset has been generated with the sensor array fabricated by the author (denoted as Sensor Array-2). The schematic of the fabricated sensor array is shown in Fig. 3.3.

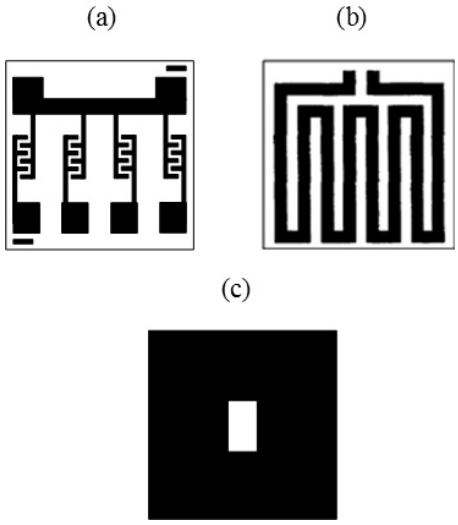

Fig. 3.3 Schematic of the sensor array (Sensor Array-2) fabricated by the author

An alumina substrate has been used as a substrate for sensor fabrication. The sensor array consisted of four different sensor elements including pure SnO_2, Pd-doped, Pt-doped and ZnO-doped sensor elements respectively. A common electrode as a reference node was provided for all the four sensors, instead of individual electrode to decrease the noise and other relative measurement complications. A heater element was fabricated on the back side of substrate to maintain a uniform temperature throughout the substrate. A complementary contact space scheme was adopted for sensor array and heater fabrication to avoid the wear and tear losses due to contact pins.

In the first step, the heater pattern screen printed and dried at a temperature of 100-125 °C for 15 minutes in the oven. The dried heater pattern was fired in thick film furnace (DEK Model-840) with total time profile of 40 minutes (15-10-15) with peak temperature zone of 850 °C for 10 minutes. In the second step, the finger electrode patterns were screen printed using gold conductor paste (8836; Electro Science Lab., USA), and dried at a temperature of 100-125 °C for 15 minutes and fired at 850 °C with total time profile of 40 minutes (15-10-15) with peak temperature

zone of 850 °C for 10 minutes. The undoped and doped tin oxide pastes were screen printed over the electrode pattern and dried at 125 °C for 15 minutes and then fired at 550 °C as mentioned in Section 2.4.8 previously. Three doped pastes were prepared by adding 1% by weight of Pd, Pt and ZnO in SnO_2 paste (Type 3050, Electro Science Lab). $PdCl_2$ and $PtCl_2$ and ZnO powder were used for Pd, Pt and ZnO doping respectively and subsequently ball milling was performed for 6 hours. The fabricated sensor array was tested for varying concentrations of LPG, N_2O, Acetone and 2-propanol in a locally developed test chamber. The responses for individual gases were taken by subsequently increasing the concentration as shown in Fig. 3.4.

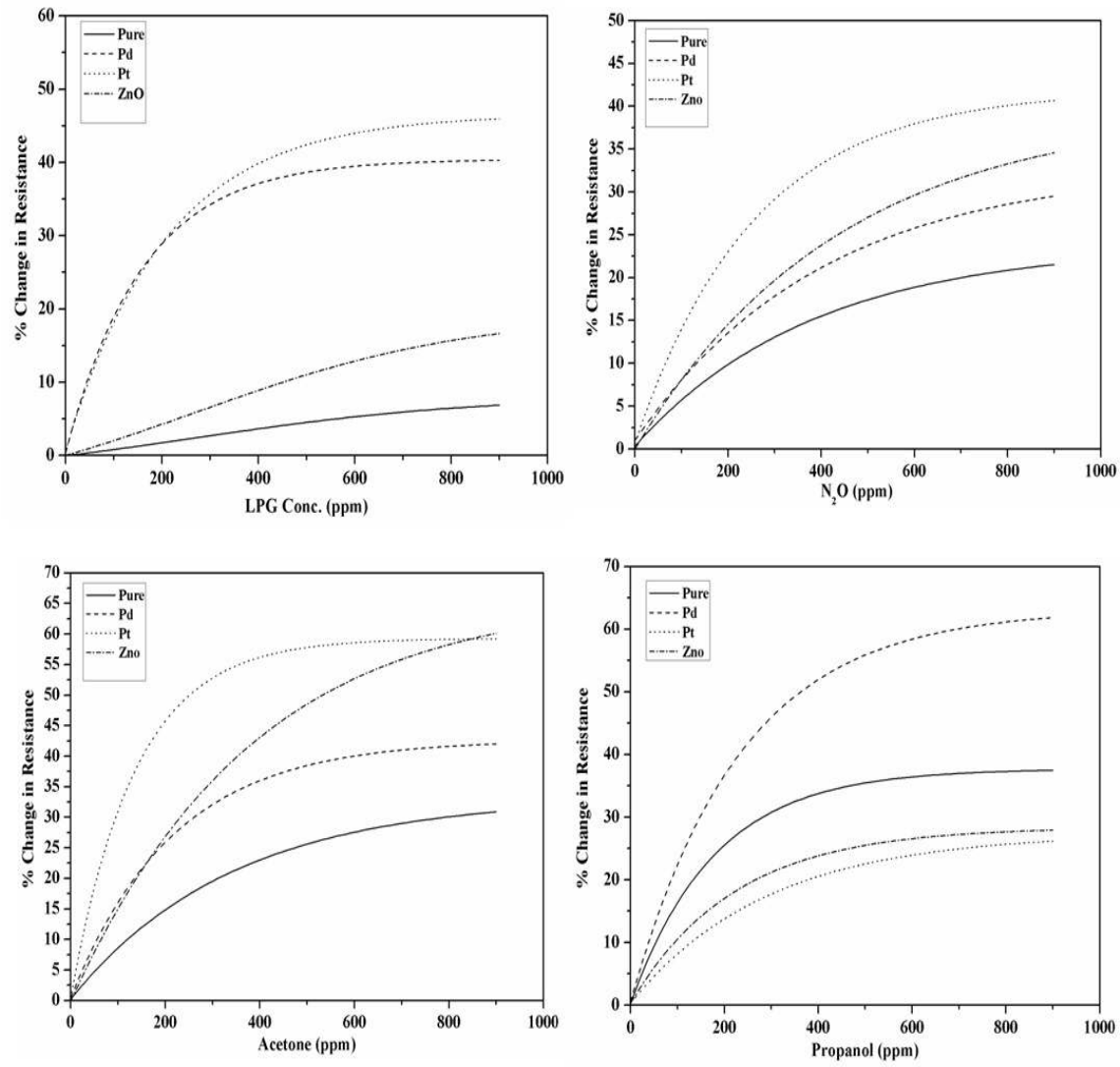

Fig. 3.4 Steady state responses of Sensor Array-2 upon exposure to LPG, N_2O Acetone and 2-propanol

3.2.2 Data Extraction

The results of the experiment have been verified using two types of data sets:

(i) The first dataset (Dataset-1) has been extracted from the steady state responses of sensor array for various gases viz. LPG, CH_4, CO and H_2 taken by [Mishra and Agarwal (1998)].

(ii)The second dataset (Dataset-2) has been prepared from generating the steady state responses of sensor array fabricated by the author (Sensor Array-2, as discussed in Chapter 2) for LPG, N_2O, Acetone and 2-propanol.

The first data set has been prepared by sampling response curves (Fig. 3.2) of the four sensors for each gas. 18 samples for LPG and H_2 each and 14 samples for CH_4 and CO each were sampled which formed an experimental data set with 64 samples. The experimental data were plotted on the 3-D scatter graphs by taking 3 sensor elements at a time. Four such graphs were obtained, one of which have been shown in Fig. 3.5.

The second data set has been prepared by sampling response curves (Fig. 3.4) of the sensor array for different gases. 24 samples for LPG and N_2O, Acetone and 2-propanol each were taken which formed an experimental data set with 96 samples.

The experimental data were plotted on the 3-D scatter graphs by taking 3 sensor elements at a time. Four such graphs were obtained, one of which have been shown in Fig. 3.6. These plots (Fig. 3.5 and Fig. 3.6) show that the raw datasets have some overlapping among different clusters in their raw form. It is expected that data preprocessing or transformation using some feature extraction technique can be used for improvement in cluster separation.

3.2.3 Principal Component Analysis [Smith (2002); Haykin (2008); MATLAB (2008)]

Principal component analysis (PCA) is a standard tool in modern data analysis. Being a simple, non-parametric method for extracting relevant information from complex data, it is used in diverse fields from neuroscience to computer graphics. Besides, other uses of PCA are simplification, data reduction, modeling, outlier detection, variable selection, classification and prediction. With minimal effort, PCA provides a roadmap for reducing a complex data to a lower dimensional data to reveal sometimes hidden, simplified structures that often underlie and without much loss of information.

Principal component analysis (PCA) is an unsupervised method widely used for the data preprocessing and dimensionality reduction purposes in various classification problems.

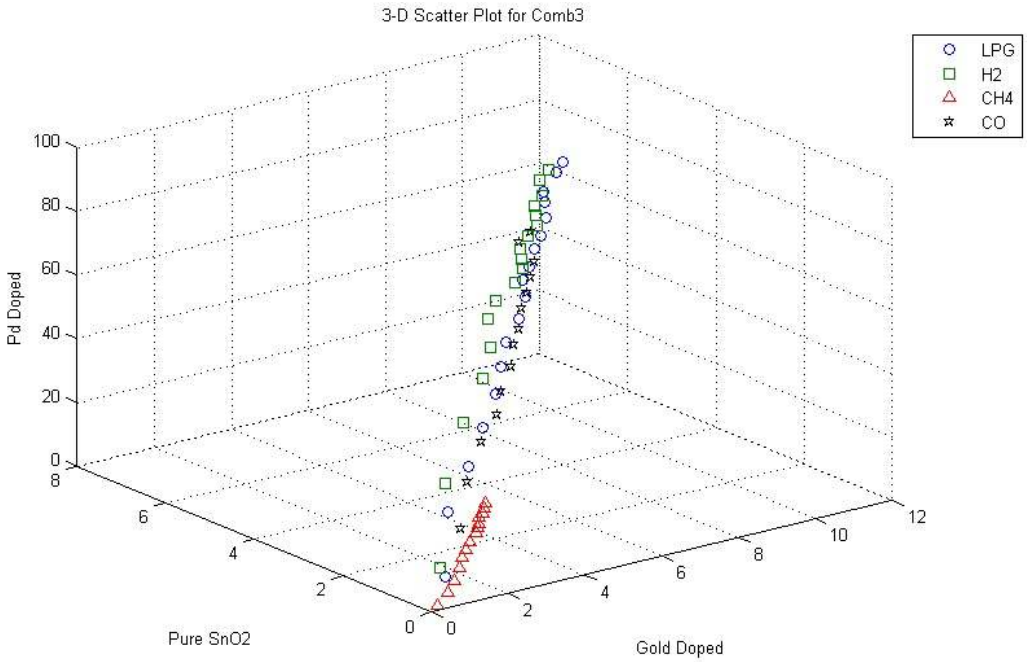

Fig. 3.5 3-D Scatter plot of the raw data obtained from response of the Sensor Array-1

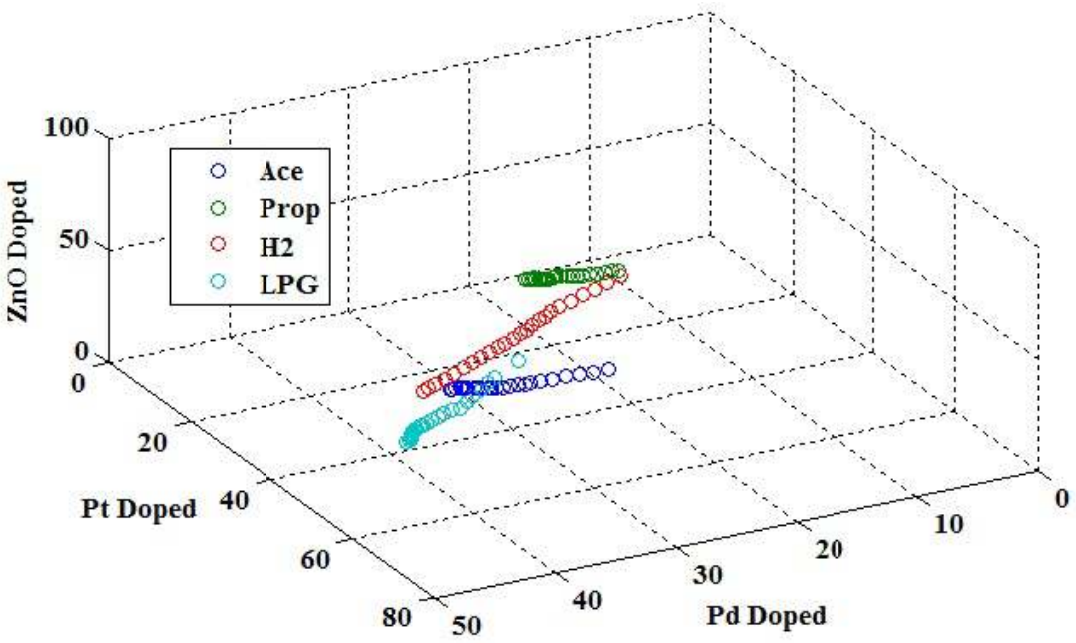

Fig. 3.6 3-D Scatter plot of the raw data obtained from response of the Sensor Array-2

PCA uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. A vector of n random variables x for which covariance matrix is Σ, the principal components (PCs) can be defined by

$$z = Ax \tag{3.1}$$

where, z is the vector of n PCs and A is the $n \times n$ orthogonal matrix with rows that are the eigenvectors of Σ [Gardner (1987)].

Following are the main steps used to perform PCA transformation of a dataset:

a) Taking the whole dataset ignoring the class labels
b) Computing the mean vector followed by subtracting from the original dataset
c) Calculation of the covariance matrix of the mean subtracted data
d) Calculation of the eigenvectors and eigenvalues of the covariance matrix
e) Choosing components and form a feature vector
f) Deriving the new data set

a) Suppose we have a data set X in the form of m x n matrix where m is the number of training example and n represents the dimensions. The notation p corresponds to the number of sensor elements in the sensor array used for the measurement.

b) The mean subtracted is the average across each dimension. So, each of the p dimension data (p_1, p_2... p_n) values have corresponding mean value which is subtracted from all the values in the respective dimension. Thus, all the p values have $\overline{p}$ subtracted from them. This produces a data set whose mean is zero.

c) Covariance is always measured between two dimensions. Suppose we have two dimension p_1 and p_2 the covariance can be calculated as:

$$\operatorname{cov}(p_1, p_2) = \frac{\sum_{i=1}^{m} (p_{1i} - \overline{p_1})(p_{2i} - \overline{p_2})}{m-1} \tag{3.2}$$

For the data set with more than 2 dimensions, there is more than one covariance measurement that can be calculated. For example, from a three dimensional dataset (dimensions (p_1, p_2, p_3)), the covariance could be calculated as *cov(x, y)*, *cov(x, z)* and *cov(y, z)*. In fact, for an n-dimensional dataset, one can calculate $\frac{n!}{(n-2)!\times 2}$ different covariance values. Thus for n dimensions we have covariance formula as follows:

$$A = Cov\,(P_1, P_2, \ldots\ldots P_n) = \begin{bmatrix} Cov(p_1,p_1) & Cov(p_1,p_2) & \ldots\ldots & Cov(p_1,p_n) \\ Cov(p_2,p_1) & Cov(p_2,p_1) & \ldots\ldots & Cov(p_2,p_n) \\ \vdots & \vdots & \vdots & \vdots \\ Cov(p_n,p_1) & Cov(p_n,p_1) & \ldots\ldots & Cov(p_n,n) \end{bmatrix} \quad (3.3)$$

d) Since the covariance matrix is square, one can calculate the eigenvectors and eigenvalues for this matrix. These are rather important, as they tell us useful information about our data. The eigenvector of a square matrix (obtained covariance matrix in equation (3.3)) is a non-zero vector v that, when the matrix multiplies v, yields a constant multiple of v. The multiplier is generally denoted by λ

$$Av = \lambda v \quad (3.4)$$

The number λ is called the eigenvalue of A corresponding to v.

One can write the equation (3.4) as:

$$(A - \lambda I)v = 0 \quad (3.5)$$

where, I is the n x n identity matrix.

For a non-zero vector v to satisfy this equation, $A - \lambda I$ must not be invertible *i. e.* the determinant of $A - \lambda I$ must equal 0. The $det(A-\lambda I)$ is called the characteristic polynomial of n^{th} of A. The eigenvalues of A are simply the roots of the characteristic polynomial of A and λ is called an eigenvalue of A and v an eigenvector of A. in order to keep eigenvectors standard, the eigenvector is usually scaled to make it to have a length of 1, so that all eigenvectors have the same length.

The goal of principal component analysis is to identify the most meaningful basis to re-express a dataset. It is expected that the new basis will filter out the noise and reveal hidden structure. It turns out that the eigenvector with the highest eigenvalue is the principle component of the data set. This gives the components in order of significance. However, one can ignore the components of lesser significance to reduce the data complexity

The number of principal components is less than or equal to the number of original variables. This transformation is defined in such a way that the first principal component has the largest possible variance. First variable thus transformed accounts for as much of the variability in the data as possible and each succeeding component in turn have the highest variance possible under the constraint that it be orthogonal to the preceding components.

Feature vector is thus constructed by taking the eigenvectors that one want to keep from the list of eigenvectors, and forming a matrix with these eigenvectors in the columns.

$$feature\ \ vector = (eign_1\ eign_2\ .....eign_n) \tag{3.6}$$

One can choose to leave out the smaller, less significant component

e) Once the components (eigenvectors) have been chosen that one wish to keep in the data and formed a feature vector, simply the transpose of the vector is taken and multiplied on the left of the original data set, transposed.

$$Final\ Data = Row\ feature\ Vector \times Row\ feature\ Adjust \tag{3.7}$$

The PCA technique has been applied to the raw datasets of gases/odors' responses described in section 3.2. Dimensionality reduction is also done for the transformed data from four to three, as first three principal components were covering nearly 98% of the variance for both types of datasets.

3.2.4 Linear Discriminant Analysis [Fisher (1938); Wilks (1963); Swet and Weng (1996); Balakrishnama and Ganapathiraju (1998)]

Linear discriminant analysis (LDA) is generally a supervised feature extraction tool which finds a linear transformation that maximizes the ratio between-class separation to within-class separation. LDA easily handles the case where the within-class frequencies are unequal. In the present work, LDA preprocessing has been applied on sensor responses in order to investigate its effect on gases/odors' classification in comparison to PCA method described earlier.

The main difference between PCA and LDA is that LDA does data classification whereas the PCA does more of feature classification. In PCA, the shape and location of the original datasets change when transformed to a different space whereas LDA doesn't change the location but only tries to provide more class separability and draw a decision region between the given classes. The LDA method also helps to understand the distribution of the feature data in a better way [Balakrishnama and Ganapathiraju (1998)]. Mathematical steps used to perform class dependent LDA which is used in the present work can be explained with the help of two class problem which can be generalized for multiclass problem.

a) Let's suppose there are two sets of data with class C_1 and C_2 respectively as shown below:

$$C_1 = \begin{bmatrix} a_{11} & a_{12} & \ldots\ldots & a_{1n} \\ a_{21} & a_{22} & \ldots\ldots & a_{2n} \\ : & : & : & : \\ a_{m1} & a_{m2} & \ldots\ldots & a_{mn} \end{bmatrix}, \qquad C_2 = \begin{bmatrix} b_{11} & b_{12} & \ldots\ldots & b_{1n} \\ b_{21} & b_{22} & \ldots\ldots & b_{2n} \\ : & : & : & : \\ b_{m1} & b_{m2} & \ldots\ldots & b_{mn} \end{bmatrix}$$

b) The mean of each dataset and mean of entire dataset is computed. Let μ_1 and μ_2 be the mean of class 1 and class 2 respectively and be μ mean of entire data, which is obtained by merging set 1 and set 2, is given by (3.8).

$$\mu = p_1\mu_1 + p_2\mu_2 \tag{3.8}$$

where p_1 and p_2 are the apriori probabilities of the classes. In the case of this simple two class problem, the probability factor is assumed to be 0.5.

c) LDA, within-class and between-class scatter are used to formulate criteria for class separability. Within-class scatter is the expected covariance of each of the classes. The scatter measures are computed using following equations.

$$S_w = \sum_j p_j \times (\text{cov}_j) \tag{3.9}$$

So, for two class problem one can write

$$S_w = 0.5 \times \text{cov}_1 + 0.5 \times \text{cov}_2 \tag{3.10}$$

d) Covariance matrix is computed using the following equation

$$\text{cov}_j = (x_j - \mu_j)(x_j - \mu_j)^T \tag{3.11}$$

The between-class scatter is computed using the following equation.

$$S_b = \sum_j (\mu_j - \mu)(\mu_j - \mu)^T \tag{3.12}$$

e) The optimizing criterion in LDA is the ratio of between-class scatter to the within-class scatter. The solution obtained by maximizing this criterion defines the axes of the transformed space. For *C*-class separate optimizing criterion is required for each class. The optimizing factors in case of class dependent type are computed as:

$$criterian_j = inv(\text{cov}_j) \times S_b \tag{3.13}$$

For LDA, the transformations are found as the eigenvector matrix of the different criteria defined in equations. After obtaining the transformation matrices, the datasets are transformed using the class specific transforms. For the class dependent LDA,

$$transformed\_set\_j = transform\_j^T \text{ x } set\_j \tag{3.14}$$

In discriminant analysis, the target is to determine the projection matrix W that maximizes the ratio $\frac{\det\{S_b\}}{\det\{S_w\}}$ *i.e.* to maximize the between-class scatter while minimizing the within-class scatter [Swet and Weng (1996)]. It has been proven that this ratio is maximized when the column vectors of projection matrix W are the eigenvectors of $S_w^{-1}S_b$ associated with the largest eigenvalues [Fisher (1938); Wilks (1963)]. Then, the scalar components in Z are feature values of the given samples and the column vectors of W are the most discriminating feature vectors.

The PCA and LDA methods described above have been used on the raw datasets (Dataset-1 and Dataset-2) of responses of sensor array for various gases/odors as described previously. The transformed data of Dataset-1 using the LDA and PCA model displayed with the Gaussians fitted by the Maximum-Likelihood have been shown in Fig. 3.7 and Fig. 3.8, for only first and then for first two dimensions, respectively. Fig. 3.9 shows the transformed data for Dataset-2 using the LDA and PCA model with the Gaussians fitted by the Maximum-Likelihood for first two dimensions. The Statistical Pattern Recognition Toolbox for MATLAB was used to generate these plots [Franc and Hlavac (2004)].

It is clear from these figures that data separation among different clusters of gases is improved for LDA and PCA transformed data. Thus, both these techniques are expected to perform better for neural classifier. Simple feedforward neural network with different backpropagation algorithms has been used for comparing the network performance with the LDA and PCA transformed gases/odor' data. The performance has been compared in terms of how accurately the network has classified transformed data of different gases/odors using both these feature extraction techniques.

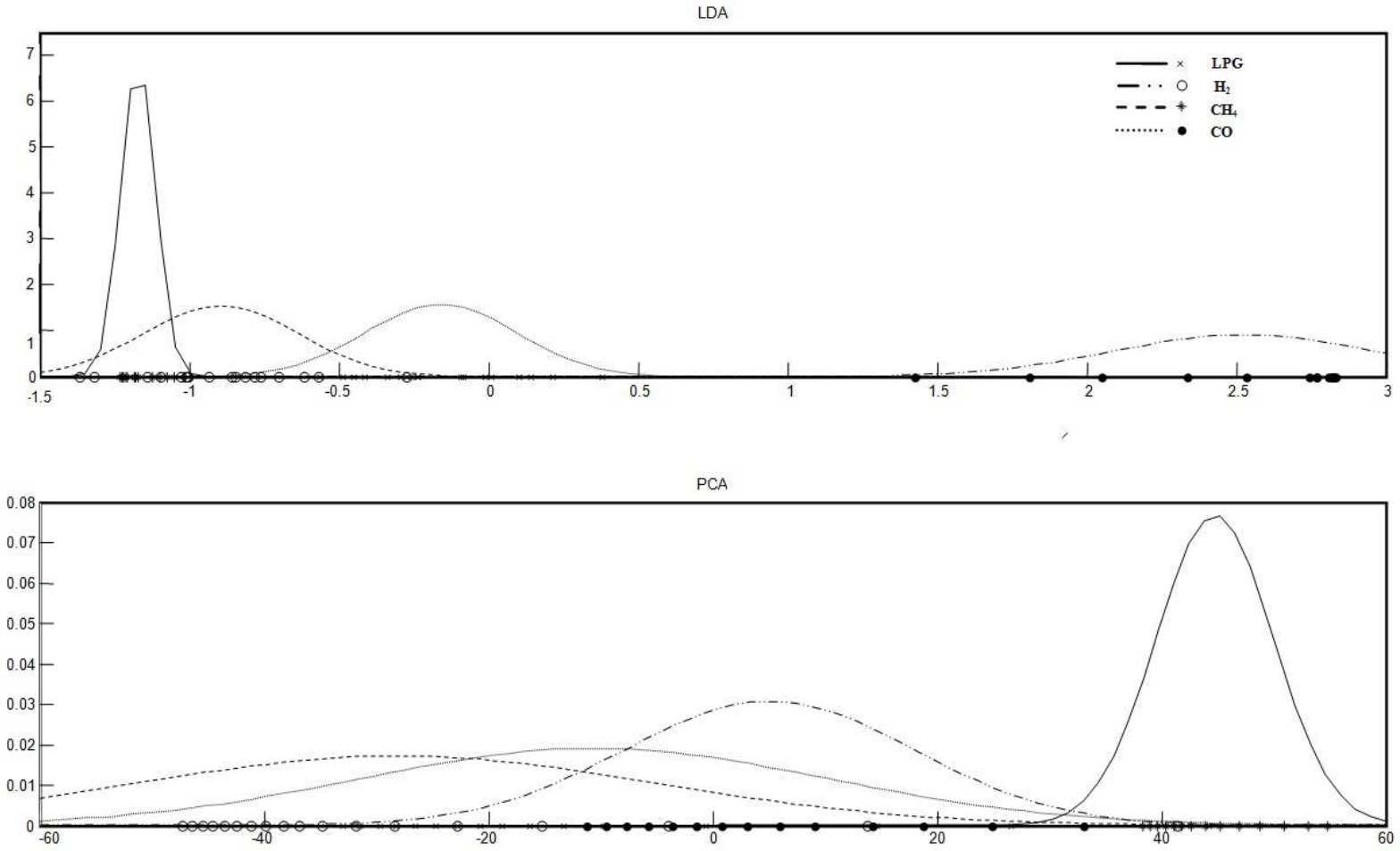

Fig. 3.7 The extracted data using the LDA and PCA model with the Gaussians fitted by the Maximum-Likelihood for the first dimension for Dataset-1

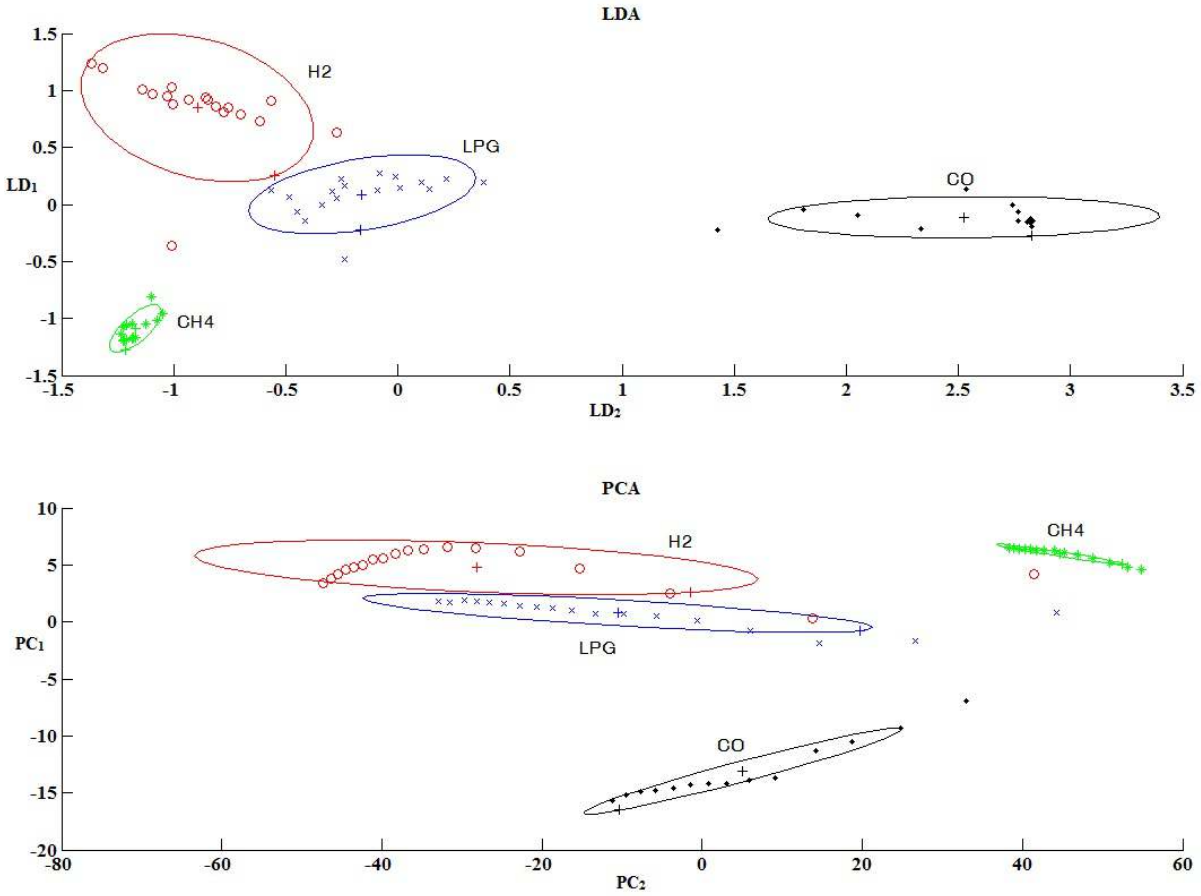

Fig. 3.8 The extracted data using the LDA and PCA model with the Gaussians fitted by t the Maximum-Likelihood for the first two dimensions of Dataset-1

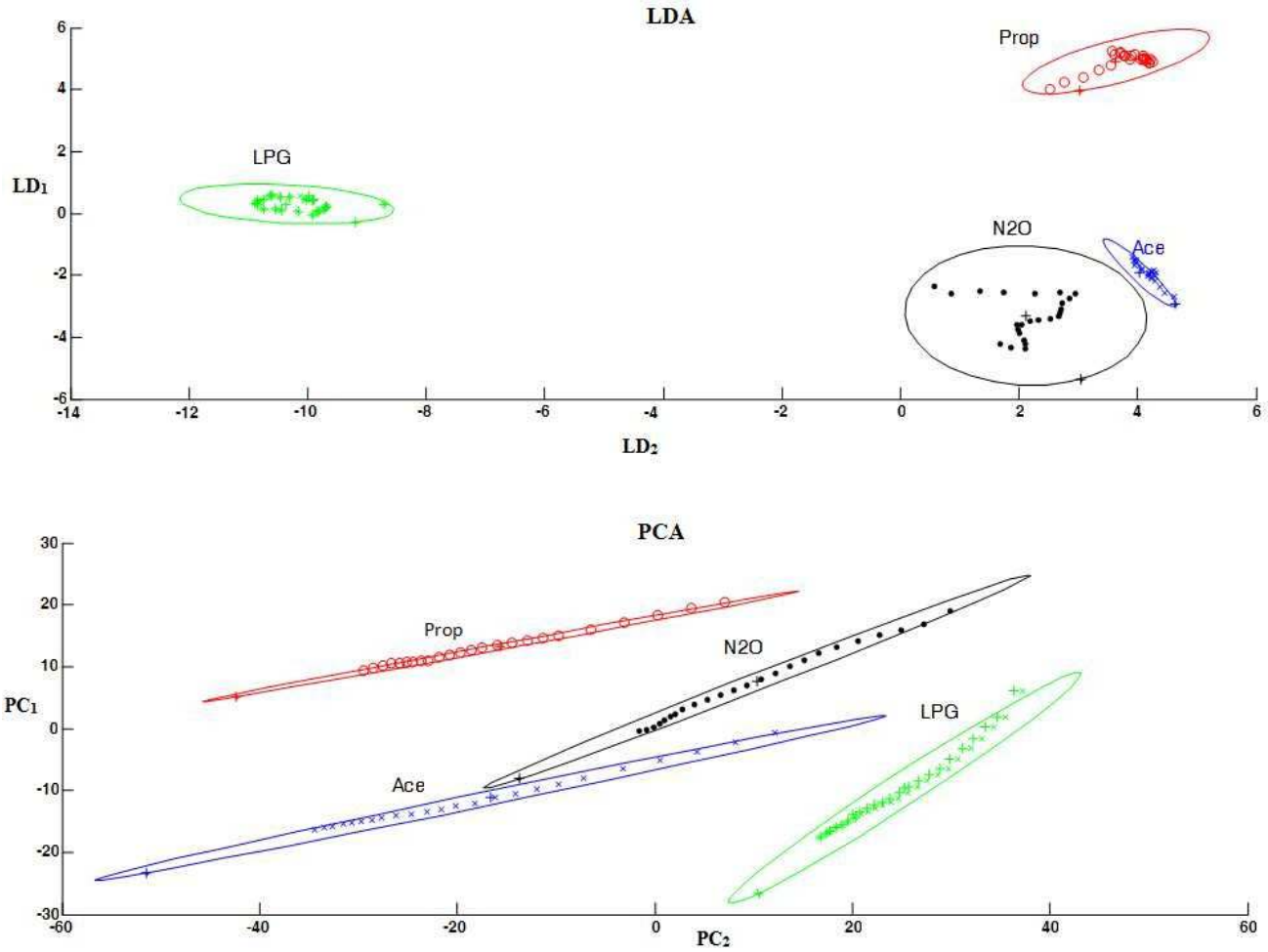

Fig. 3.9 The extracted data using the LDA and PCA model with the Gaussians fitted by the Maximum-Likelihood for the first two dimensions of Dataset-2

3.2.5 Back Propagation Neural Network (BPNN) [Haykin (2008); Kumar *et al.* (2011(a)); MATLAB (2008)]

Multilayer perceptron (MLP) with back propagation algorithms are the basic neural network techniques used for pattern recognition in machine learning process. MLP constitute an important class of neural network which typically consist of a set of sensory units called source node which form the input layer. It consists of one or more hidden layers of computation nodes having different threshold function. Typical multilayer feed-forward neural network architecture is shown in Fig. 3.10

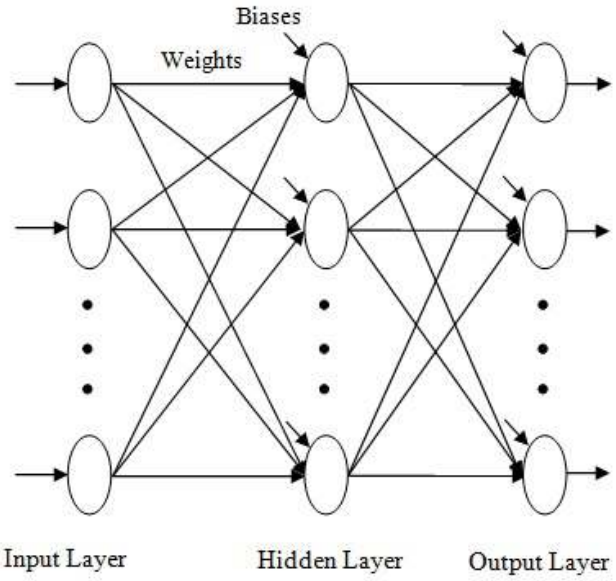

Fig. 3.10 Typical Multilayer Feed Forward Neural Network

The input signal propagates through the network in a forward direction, on a layer by layer basis. The MLP with error back propagation has been successfully applied to many problems in pattern recognition by training in a supervised manner. The process of training a neural network involves tuning the values of the weights and biases of the network to optimize network performance, as defined by the network performance function. The default performance function for feed-forward networks is mean square error *mse*, the average squared error between the network outputs and the target outputs which is defined as:

$$mse = \frac{1}{N}\sum_{i}^{N}(\varepsilon_i) = \frac{1}{N}\sum_{i}^{N}(t_i - o_i)^2 \tag{3.15}$$

where, o_i denotes actual outputs and t_i denotes the desired outputs .

Backpropagation is the generalization of the least mean square (LMS) algorithm [Widrow & Hoff, (1960)] to multiple-layer networks and nonlinear differentiable

transfer functions. Input vectors and the corresponding target vectors are used to train a network until it can approximate a function, associate input vectors with specific output vectors, or classify input vectors in an appropriate way as defined by the user.

Networks with biases, a hidden layer, and a linear output layer are capable of approximating any function with arbitrary accuracy. Standard backpropagation is a gradient descent algorithm, in which the network weights are moved along the negative of the gradient of the performance function. The term backpropagation refers to the manner in which the gradient is computed for nonlinear multilayer networks. Basically, error backpropagation consists of two passes through the different layers of the network viz. *forward pass* and *backward Pass*.

In the *forward pass*, the input vectors are applied to the sensory nodes of the network, and its effect propagates through the network layer-by-layer. The actual response of the network is delivered by the output nodes in the form of an output vector. The outputs are compared with a target vector and the difference is generated as error and propagated in *backward pass*. Let the error signal at the output of neuron *j* at iteration *n* be defined by [Haykin (2008); Kumar (2011(a))]:

$$e_j(n) = d_j(n) - y_j(n) \tag{3.16}$$

where, $d_j(n)$ represents the desired output at the output node *j* at iteration *n*, and $y_j(n)$ be the actual output at the output node *j* at iteration *n*.

Let the instantaneous value of the error energy for neuron *j* be defined as: $\frac{1}{2}e_j^2(n)$ Then, for all neurons in the output layer instantaneous value ξ(n) of the total energy is given by:

$$\xi(\mathrm{n}) = \frac{1}{2}\sum_{\mathrm{j}\in\mathrm{C}} e_j^2(n) \tag{3.17}$$

where, *C* is the set of all neurons in the output layer of the network. Let *N* denotes the total number of patterns contained in the training set. The average squared energy over the entire training sample is now given by:

$$\xi_{avg}(N) = \frac{1}{N}\sum_{n=1}^{N}\xi(\mathrm{n}) \tag{3.18}$$

For a given training set, ξ_{avg} is called the cost function. The cost function is a measure of learning performance. Minimization of the cost function is done iteratively. The weights associated with the network are updated on a pattern-by-pattern basis until one complete presentation of the entire training set (epochs) has been done. The adjustments to the weights are made in accordance with the respective

errors computed for each pattern presented to the network. The arithmetic average of these individual weight changes over the entire training sets presents an estimate of the true change that would result from modifying the weights based on minimizing the cost function over the entire training set. Fig 3.11 shows a neuron *j* being fed by a set of input signals produced by a layer of its neurons to its left. The induced local field $v_j(n)$ produced at the input of the activation function associated with neuron *j* is given by,

$$v_j(n) = \sum_{j=0}^{m} \omega_{ji}(n)\, y_j(n) \tag{3.19}$$

where, *m* is the total number of inputs applied to neuron *j*, and ω_{ji} is the synaptic weight from neuron *i* to neuron *j*. The signal appearing at the output of neuron *j* at iteration *n* is a function of the induced local field.

$$y_j(n) = \varphi\{v_j(n)\} \tag{3.20}$$

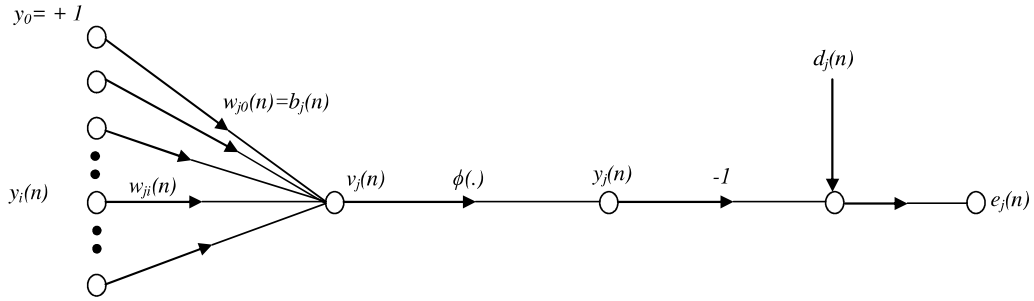

Fig. 3.11 Neuron '*j*' being fed by a set of signals from a previous layer of neurons

The backpropagation algorithm applies a correction $\Delta\omega_{ji}(n)$ to the synaptic weight $\omega_{ji}(n)$, which is proportional to the partial derivative $\frac{\delta\xi(n)}{\delta\omega_{ji}(n)}$. Applying the chain rule of calculus, this gradient can be expressed as:

$$\frac{\delta\xi(n)}{\delta\omega_{ji}(n)} = \frac{\delta\xi(n)}{\delta e_j(n)} \frac{\delta e_j(n)}{\delta y_j(n)} \frac{\delta y_j(n)}{\delta v_j(n)} \frac{\delta v_j(n)}{\delta\omega_{ji}(n)} \tag{3.21}$$

The partial derivative $\frac{\delta\xi(n)}{\delta\omega_{ji}(n)}$ represents a sensitivity factor. It determines the direction of search in weight space for the synaptic weight ω_{ji}. Differentiating both sides with respect to $e_j(n)$ we get,

$$\frac{\delta\xi(n)}{\delta e_j(n)} = e_j(n) \tag{3.22}$$

Differentiating both sides of (3.22) with respect to $y_j(n)$, we get,

$$\frac{\delta e_j(n)}{\delta y_j(n)} = -1 \tag{3.23}$$

Differentiating both sides of (3.20) with respect to $v_j(n)$, we get,

$$\frac{\delta y_j(n)}{\delta v_j(n)} = \phi_j^i\{v_j(n)\} \tag{3.24}$$

Also, differentiating (3.19) with respect to ω_{ji}, we get,

$$\frac{\delta v_j(n)}{\delta \omega_{ji}(n)} = y_j(n) \tag{3.25}$$

The use of (3.22) to (3.25) in (3.21) yields:

$$\frac{\delta \xi(n)}{\delta \omega_{ji}(n)} = -e_j(n)\phi_j^i\{v_j(n)\}y_j(n) \tag{3.26}$$

The correction $\Delta\omega_{ji}(n)$ applied to $\omega_{ji}(n)$ is defined by the delta rule:

$$\Delta\omega_{ji}(n) = -\eta\frac{\delta \xi(n)}{\delta \omega_{ji}(n)} \tag{3.27}$$

where, η is the learning-rate parameter of the backpropagation algorithm. The gradient descent in weight space takes place in a direction for weight change that reduces the value of ξ(n). The use of (3.26) in (3.27) yields:

$$\Delta\omega_{ji}(n) = -\eta\delta_j(n)y_j(n) \tag{3.28}$$

where, the local gradient $\delta_j(n)$ is defined by

$$\begin{aligned}\delta_j(n) &= -\frac{\delta \xi(n)}{\delta v_j(n)} \\ &= -\frac{\delta \xi(n)}{\delta e_j(n)}\frac{\delta e_j(n)}{\delta y_j(n)}\frac{\delta y_j(n)}{\delta v_j(n)} \\ &= e_j(n)\phi_j'\{v_j(n)\}\end{aligned} \tag{3.29}$$

The local gradient points to required change in synaptic weights. Hence, the above relation between the learning rate, local gradient and weight correction can be summarized as follows:

(Weight Correction)= (Learning Rate)(Local Gradient)*(Input Signal of Neuron 'j')*

Learning Rate and Momentum Constant parameters: The learning rate parameter η is a measure of the changes to the synaptic weights in the network over subsequent iterations. Thus, a smaller learning rate parameter makes smaller changes in the synaptic weights and hence, the trajectory in the weight space is smoother. A smaller learning rate results in a slow learning. If the learning rate parameter η is made too

large with an aim to speed up the learning process, the resulting large changes in the synaptic weights may cause the network to become unstable. To avoid the danger of instability while keeping the learning rate fast enough, another term is added to the delta rule, which is known as the momentum constant and is denoted by α. Hence, the equation (3.28) becomes:

$$\Delta\omega_{ji}(n) = \alpha\Delta\omega_{ji}(n-1) + \eta\delta_j(n)y_j(n) \tag{3.30}$$

The inclusion of momentum term in the backpropagation algorithm has a stabilizing effect in directions that oscillate in sign. The momentum term also prevents the learning process from terminating it in a shallow local minima on the error surface. The training through backpropagation algorithm proceeds iteratively. A prescribed set of training examples are fed repeatedly to the multilayer perceptron. The learning process continues on an epoch-by-epoch basis till the stabilization of the synaptic weights and bias levels of the network till the average squared error over the entire training set converges to some minimum value. Backpropagation algorithm cannot be shown to converge. However, it is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small. The rate of change in the average squared error is typically considered to be small enough if it lies in the range of 0.1 to 1 percent per epoch [Kumar (2011); Haykin (2008)].

3.2.6 Training Functions [Seiffert (2006); Haykin (2008); MATLAB (2007); Kumar *et al.* (2011(a))]

Different training functions exist for training multilayer feed-forward networks. Any of them can be used to optimize the performance function. Each of these functions has its own specific purpose and way of optimizing the network. In the present study, four training functions viz. *trainlm*, *trainoss*, *trainscg* and *trainrp* have been used.

(i) Levenberg-Marquardt algorithm (Trainlm):

Trainlam methodology implemented in MATLAB is based on Levenberg Marquardt algorithm. It is a network training function that updates weight and bias values according to Levenberg-Marquardt optimization. This is often the fastest backpropagation algorithm in the Neural Network toolbox of MATLAB, and is highly recommended as a first-choice supervised algorithm, although it does require more computational memory than other algorithms. This algorithm can train any network as long as its weights, net input, and transfer functions have derivative functions.

Marquardt-Levenberg algorithm [Marquardt (1963)] is an approximation to Newton's method. Consider a function $V(\underline{x})$ which we want to minimize with respect to the parameter vector $\underline{x}$ (lower bar denotes the 'matrix form'), then Newton's method would be [Hagan (1994)]

$$\Delta\underline{x} = -\left[\nabla^2 V(\underline{x})\right]^{-1} \nabla V(\underline{x}) \tag{3.31}$$

where, $\nabla^2 V(\underline{x})$ is the Hessian matrix and $\nabla V(\underline{x})$ is the gradient and it is assumed that $V(\underline{x})$ is a sum of squares function

$$V(\underline{x}) = \sum_{i=1}^{N} e_i^2(\underline{x}) \tag{3.32}$$

then it can be shown that

$$\nabla V(\underline{x}) = J^T(\underline{x})\underline{e}(\underline{x}) \tag{3.33}$$

$$\nabla^2 V(\underline{x}) = J^T(\underline{x})J(\underline{x}) + S(\underline{x}) \tag{3.34}$$

where *J(x)* is the Jacobian matrix

$$S(\underline{x}) = \sum_{i=1}^{N} e_i(\underline{x})\nabla^2 e_i(\underline{x}) \tag{3.35}$$

For the Gauss-Newton method it is assumed that $S(\underline{x}) \approx 0$, and the update (3.33) becomes

$$\nabla(\underline{x}) = \left[J^T(\underline{x})J(\underline{x})\right]^{-1} J^T(\underline{x})\underline{e}(\underline{x}) \tag{3.36}$$

The Marquardt-Levenberg modification to the Gauss-Newton method is

$$\nabla(\underline{x}) = \left[J^T(\underline{x})J(\underline{x}) + \mu I\right]^{-1} J^T(\underline{x})\underline{e}(\underline{x}) \tag{3.37}$$

The Marquardt-Levenberg algorithm can be considered a trust region modification to Gauss-Newton [Hagan and Menhaj (1994)]. The key step in this algorithm is the computation of the Jacobian matrix.

(ii) One-step secant backpropagation (Trainoss)

Trainoss is a network training function implemented in MATLAB which updates weight and bias values according to the one-step secant method. Backpropagation is used to calculate derivatives of performance with respect to the weight and bias variables *X*. Each variable is adjusted according to the following:

$$X = X + a*dX \tag{3.38}$$

where *dX* is the search direction. The parameter is selected to minimize the performance along the search direction. The line search function is used to locate the

minimum point. The first search direction is the negative of the gradient of performance. In succeeding iterations the search direction is computed from the new gradient and the previous steps and gradients, according to the following formula:

$$dX = -gX + Ac*X\_step + Bc*dgX \quad (3.39)$$

where gX is the gradient, X_step is the change in the weights on the previous iteration, and dgX is the change in the gradient from the last iteration

(iii) Scaled conjugate gradient backpropagation (Trainscg)

Trainscg function implemented in MATLAB is a network training function that updates weight and bias values according to the scaled conjugate gradient method. The algorithm can train any network as long as its weight, net input, and transfer functions have derivative functions. The scaled conjugate gradient algorithm is based on conjugate directions but does not perform a line search at each of iterations. Thus, the scaled conjugate gradient algorithm (SCG) was designed to avoid the time-consuming line search. The *trainscg* routine can require more iterations to converge than the other conjugate gradient algorithms, but the number of computations in each iteration is significantly reduced because no line search is performed. This algorithm combines features of Levenberg Marquardt algorithm with the conjugate gradient approach.

(iv) Resilient Gradient Descent backpropagation (Trainrp)

Trainrp is a network training function that updates weight and bias values according to the resilient backpropagation algorithm (Rprop). Due to the commonly used semi-linear transfer functions (sigmoid, hyperbolic tangent) and their actually desired property to compress an infinite input range into a finite output range by the fact that their slope approaches zero as the input moves towards ±1, the gradient can have a very small magnitude, even though the weights are far away from their optimal values.

Therefore, *trainrp* uses only the sign (plus an externally defined value) instead of the signed magnitude of the current slope to update the weights. This way the algorithm requires relatively few resources and is rather fast to obtain the solution.

The *trainlm* has fastest training and memory reduction features while *trainoss*, *trainscg* and *trainrp* have fast convergence property. All these four training algorithms have been used for training of multilayered feedforward neural network (MLFNN).

3.2.7 Cross Validation

Validation is a necessary step for the correctness of any classification scheme. A classifier trained by the training data is expected not only to produce the correct label on the training data but also to predict correctly the label for any unseen data. The basic process of validation involves partitioning the training examples into training set and validation set. Training set is used for training or learning purpose and the validation set is used to test or validate the classification accuracy. It is natural requirement that the training and validation sets must cross-over in successive rounds such that each training example has a chance of being validated against [Rodriguez *et al.* (2010)].

k-fold cross validation scheme has been used in this work. In this scheme the total training examples are randomly divided into *k* parts of approximately equal size $D= [D_1, D_2, ...D_k]$. Here *k* has been chosen such that it divides the *N*, where *N* is the total number of training examples. At each of the *k* splits, *N/k* examples are used for validation and the remaining *N(k-1)/k* are used for training purpose. The training and validation sets have been chosen in random manner. The process has been repeated for all possible combination of training and validation sets. Finally, the estimation of the error is the average value of the errors committed in each fold. Moreover, choosing appropriate value of *k* for a given dataset and neural classifier is also another necessary step. The *k* value has been optimized to have maximum classification accuracy for a given dataset and neural classifier.

3.3 Results and Discussion

The performance of LDA and PCA transformed data with different training algorithms viz. *trainlm*, *trainoss*, *trainscg* and *trainrp* have been compared for the classification of responses of thick film sensor array for different gases. The discrete raw dataset was prepared from the responses of respective gases for the thick film sensor array. The discrete dataset was divided into training and test datasets. The neural network was trained with the training dataset. Then, the unseen data (test data) was fed to the duly trained network. Cross validation scheme was adopted for the experiments performed with neural classifier.

The experiment was performed first using Dataset-1 which are prepared from published responses of thick film gas sensor array for different gases/odors viz. LPG, CH_4, CO and H_2 [Mishra and Agarwal (1998)]. The same experiment was then

repeated using responses of thick film sensor array fabricated by the author for LPG, N_2O, Acetone and 2-propanol. The extracted data using the LDA and PCA model with the Gaussians fitted by the maximum-likelihood using first two dimensions can be compared with visual inspection as shown in Fig. 3.8 and Fig. 3.9 respectively. It is clear from these figures that both LDA and PCA have good discriminative capability for both datasets.

The raw data, PCA and LDA transformed datasets were further used for training and validation using back propagation neural network (BPNN) classifier. The data dimensions were reduced from 4 to 3 as the first three dimensions were covering most of the variance. For the neural classifier, the minimum training error was chosen 0.0001. The classification errors were recorded by changing the number of hidden layers in the neural network classifier. The classification errors were found minimum for single hidden layer for all types of data viz. raw data, PCA data and LDA data. The single hidden layer was then tuned for number of neuron in the layer.

The maximum classification accuracy obtained with raw data was 84.1% for *trainlm* algorithm as shown in Fig. 3.12. The network was then trained with PCA processed dataset. The same process was repeated for all the four mentioned training algorithms. The PCA data with *trainlm*, and *trainoss*, gave accurate classification for the unseen data providing 100% classification accuracy. The accuracy obtained for *trainscg* and *trainrp* was 96% and 98% respectively. The results obtained with the PCA processed Dataset-1 have been shown in Fig. 3.13(a-b) and summarized in Table 3.1. The same neural network was then trained and tested with LDA transformed dataset. The classification accuracy was 100% for *trainlm* and *trainoss* with single hidden layer with 5 neurons. The *trainscg* and *trainrp* training functions showed accurate classification (100%) with single hidden layer and with 4 neurons. The results obtained with raw dataset and the LDA processed dataset has been shown in Fig. 3.14 (a-b). The results for Dataset-1 have been summarized in Table 3.2.

The whole experiment was repeated for the Dataset-2. The distribution of LDA and PCA preprocessed Dataset-2 has been shown in Fig. 3.5. The classification results obtained before and after preprocessing of raw Dataset-2 with BPNN classifier have been summarized in Table 3.3 and Table 3.4 for PCA and LDA transformed datasets respectively. It is obvious from the results obtained with Dataset-2 that performance of both LDA and PCA are comparable in terms of classification accuracy.

The classification accuracy with PCA preprocessed dataset is again found be promising and comparable with LDA preprocessed data. Fig. 3.12 to Fig.3.14 show the results for Dataset-1 only. The results for Dataset-2 have not been shown in pictorial form for the sake of simplicity. The results for Dataset-2 have been summarized in Table 3.3 and Table 3.4.

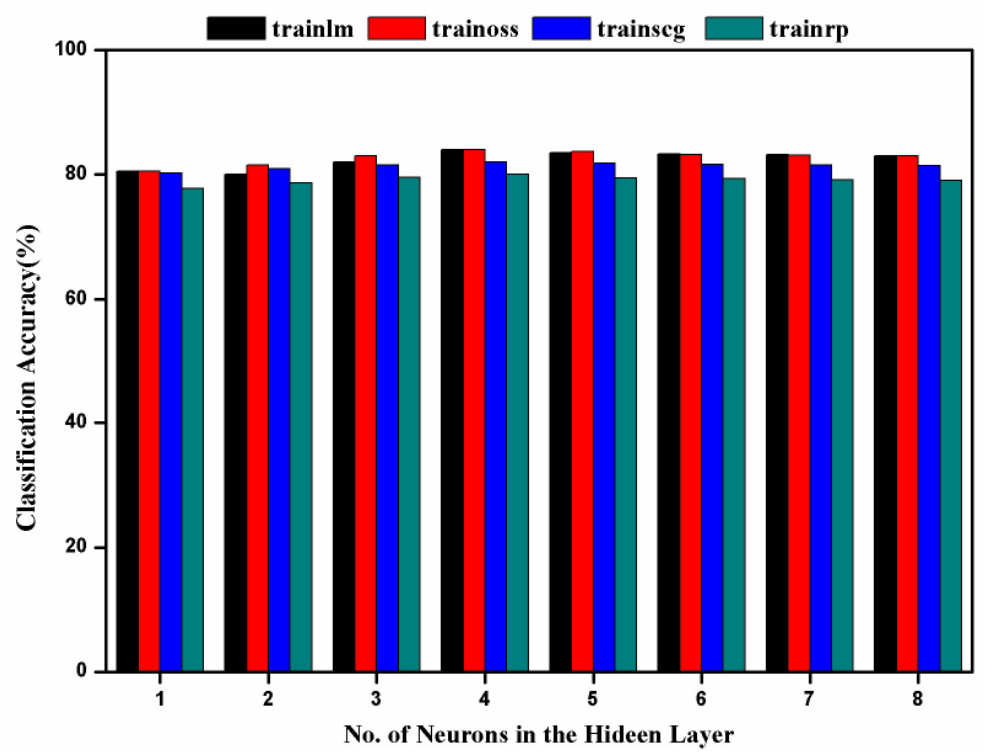

Fig. 3.12 Classification accuracy vs. number of neurons in the single hidden layers for BPNN trained with raw data (Dataset-1)

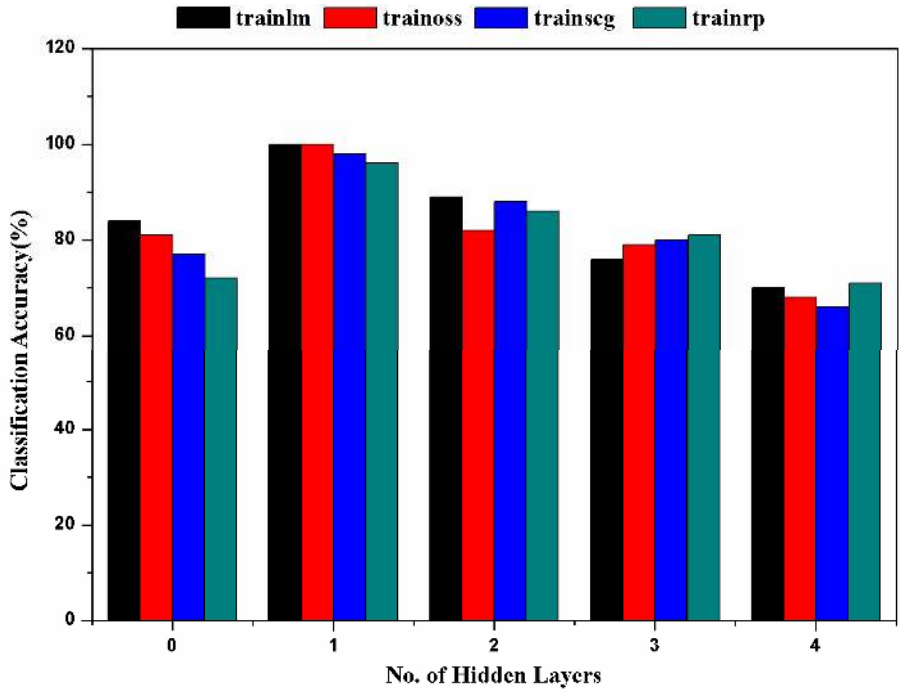

Fig.3.13 (a) Classification accuracy vs. number of hidden layers for BPNN trained with PCA transformed data (Dataset-1)

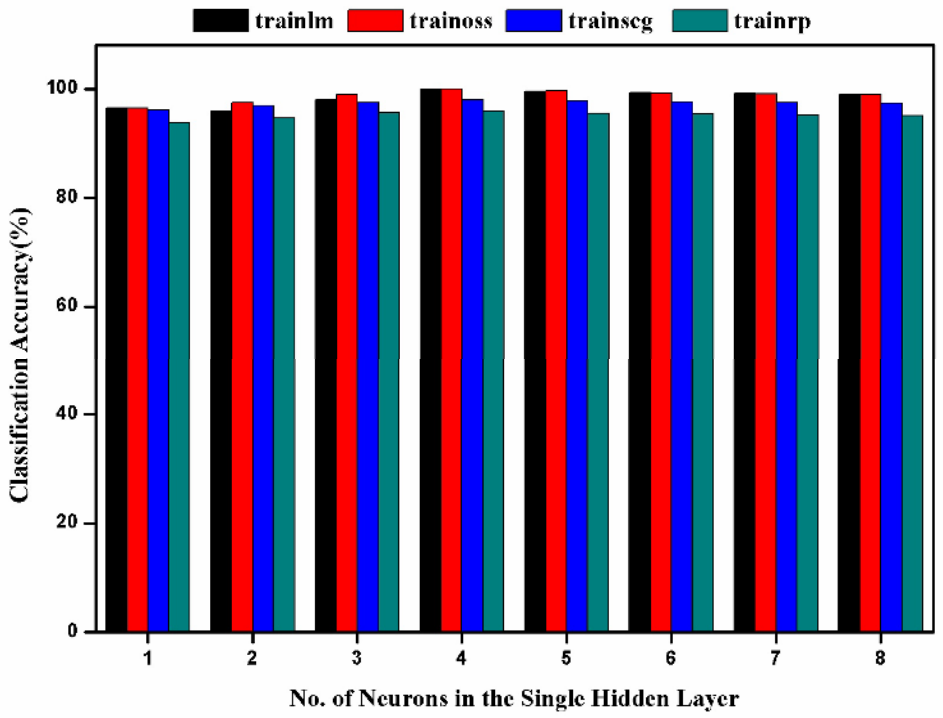

Fig. 3.13 (b) Classification accuracy vs. number of neurons in the single hidden layer for BPNN trained with PCA preprocessed data (Dataset-1)

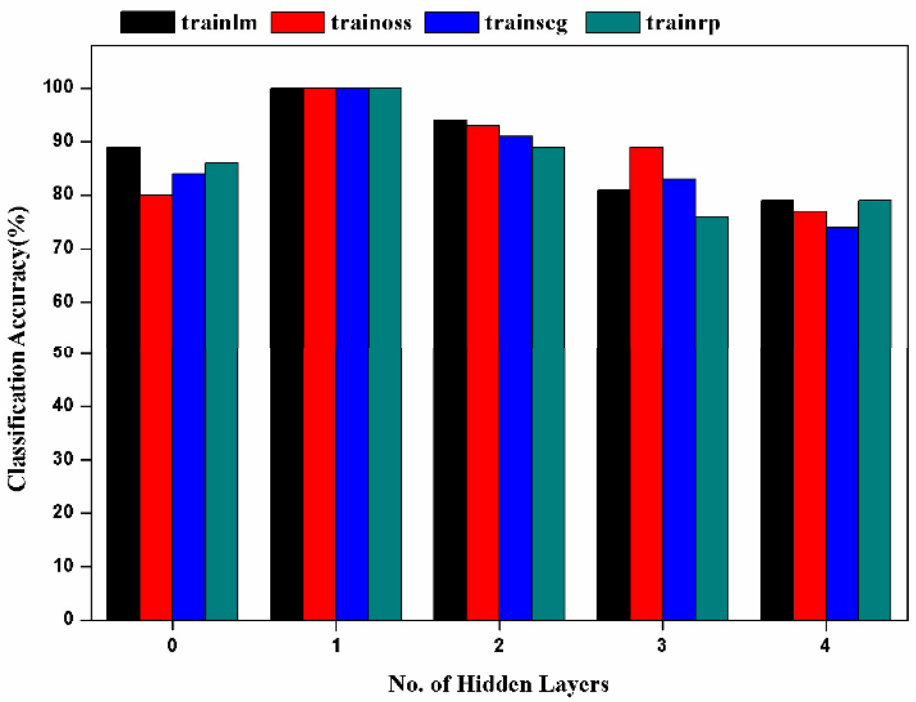

Fig. 3.14 (a) Classification accuracy vs. number of hidden layers for BPNN trained with LDA transformed data (Dataset-1)

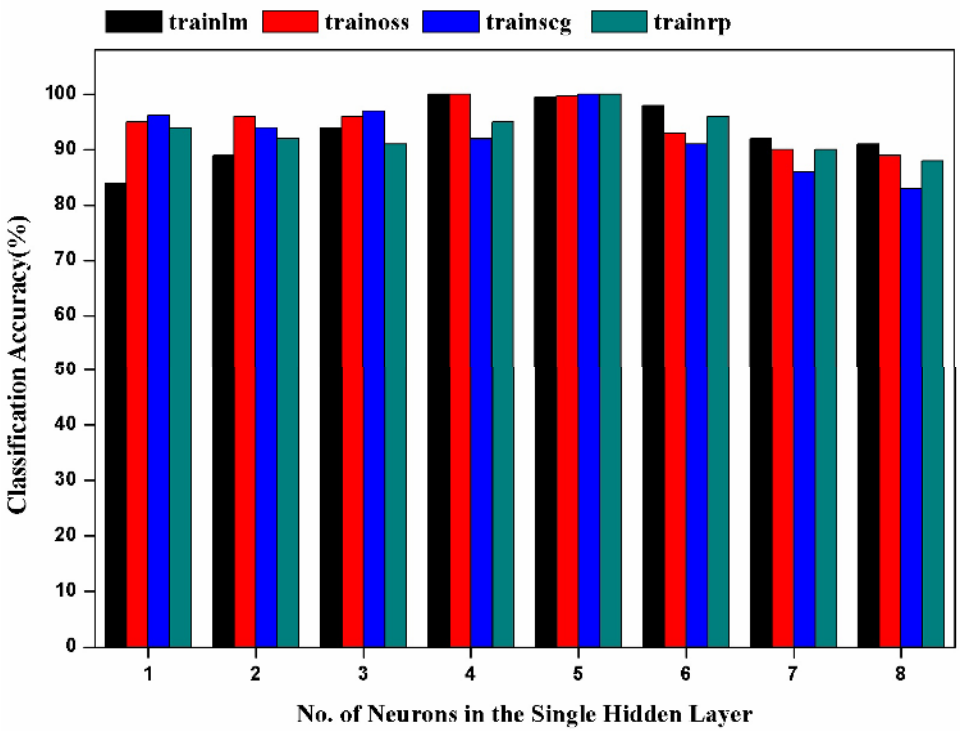

Fig. 3.14 (b) Classification accuracy vs. number of neurons in the single hidden layer for BPNN trained with LDA transformed data (Dataset-1)

The Average Classification Accuracy with Raw Dataset-1: 84.1%

Table 3.1 Classification results for Dataset-1 using PCA and BPNN

Training Function	Number of hidden layers	Number of neurons in hidden layer	Classification Accuracy (A_C) %	Epochs
Trainlm	1	4	100	445
Trainoss	1	4	100	522
Trainscg	1	5	96	612
Trainrp	1	5	98	408

Table 3.2 Classification results for Dataset-1 using LDA and BPNN

Training Function	Number of hidden layers	Number of neurons in hidden layer	Classification Accuracy (A_C) %	Epochs
Trainlm	1	5	100	472
Trainoss	1	5	100	488
Trainscg	1	4	100	475
Trainrp	1	4	100	449

The Average Classification Accuracy with Raw Dataset-2: 86.4%

Table 3.3 Classification results for Dataset-2 using PCA and BPNN

Training Function	Number of hidden layers	Number of neurons in hidden layer	Classification Accuracy (A_C) %	Epochs
Trainlm	1	4	100	225
Trainoss	1	5	100	327
Trainscg	1	4	97.5	380
Trainrp	1	4	100	211

Table 3.4 Classification results for Dataset-2 using LDA and BPNN

Training Function	Number of hidden layers	Number of neurons in hidden layer	Classification Accuracy (A_C) %	Epochs
Trainlm	1	4	100	262
Trainoss	1	5	100	368
Trainscg	1	5	97.9	276
Trainrp	1	4	98.9	198

3.4 Conclusion

Gases/odors identification task is highly problem dependent. In the present work, the classification of gases/odors using two different datasets extracted from responses of different thick film gas sensor array for different gases/odors has been performed with LDA and PCA data transformation techniques. Both, the PCA and the LDA transformations have shown good separation among different clusters as compared to raw data which is clear from visual inspection. A simple multilayer feed-forward network with back propagation was chosen for classification purpose and a performance comparison was done with different training algorithms. Further, the network was tuned for the number of hidden layers and the number of neurons in the hidden layers. Thus, it has been shown that the gases/odors classification can be done with high accuracy using simple neural classifier and with suitable feature extraction techniques such as LDA or PCA.

Also, from the experiments performed, it has been found that PCA can be good alternative preprocessing tool as compared to LDA. The PCA provides classification accuracy comparable to LDA in spite of the fact that it is unsupervised in nature. This removes the extra burden of handling class information in PCA as compared LDA which requires class information. Moreover, LDA is mathematically more complex as compared to PCA. Thus, for the present task of gas/odor identification/classification, PCA seems to be a good compromise in terms of classification accuracy, mathematical complexity and real time operation of gas sensing system. In the view of these results, the PCA has been chosen for preprocessing and dimensionality reduction in the classification and quantification tasks of individual and mixture of gases/odors which has been presented in following chapters.