CS6690: Pattern Recognition
Assignment 4: GMM, HMM, DTW & Non Parametric Methods

Due Date: 3/12/2013

Group 10
Abil N George (CS13S002)
Jom Kuriakose (CS13S028)

1. Problem Statement

Classification by modeling the class conditional density as:

1. GMM: Gaussian Mixture Model
   (a) Speaker identification dataset
   (b) Image dataset

2. HMM: Hidden Markov Model
   (a) Tidigit dataset (perform embedded re-estimation – using HTK toolkit)
   (b) Handwritten dataset

3. DTW: Dynamic Time Warping
   (a) Music dataset
   (b) Mandi dataset

4. Non-Parametric Methods:
   (1) Parzen Window Method

- Hyper sphere
- Gaussian Kernel

   (2) K-Nearest Neighbor Method
   (3) Fisher Discriminant Analysis (FDA)
   (4) Perceptron based classifier
   (5) Support Vector Machine (SVM) with linear kernel
      - for Image dataset

2. Methodology

2.1 K-Means Clustering

K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

Given an initial set of k means $m_1^{(1)},\ldots,m_k^{(1)}$ the algorithm proceeds by alternating between two steps:

Assignment step: Assign each observation to the cluster whose mean yields the least within-cluster sum of squares. Since the sum of squares is the squared Euclidean distance, this is intuitively the "nearest" mean.

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \le \|x_p - m_j^{(t)}\|^2 \ \forall\ 1 \le j \le k\},$$

where each x_p is assigned to exactly one $S^{(t)}$, even if it could be is assigned to two or more of them.

Update step: Calculate the new means to be the centroids of the observations in the new clusters.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

Since the arithmetic mean is a least-squares estimator, this also minimizes the within-cluster sum of squares objective.

The algorithm has converged when the assignments no longer change.

2.2 GMM: Gaussian Mixture Model

A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities. GMM parameters are estimated from training data using the iterative Expectation-Maximization (EM) algorithm.

EM for Gaussian Mixtures

Given a Gaussian mixture model, the goal is to maximize the likelihood function with respect to the parameters (comprising the means and covariances of the components and the mixing coefficients).

1. Initialize the means μ_k, covariances Σ_k and mixing coefficients π_k, and evaluate the initial value of the log likelihood.
2. *E step*: Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}.$$

3. *M step*: Re-estimate the parameters using the current responsibilities

$$\begin{aligned}
\boldsymbol{\mu}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \\
\boldsymbol{\Sigma}_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \left(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}\right) \left(\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}\right)^{\mathrm{T}} \\
\pi_k^{\text{new}} &= \frac{N_k}{N}
\end{aligned}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}).$$

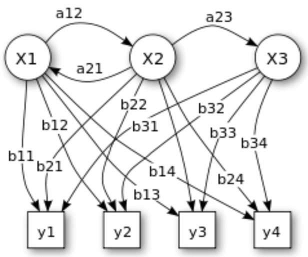
4. Evaluate the log likelihood

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied, return to step 2.

2.3 HMM: Hidden Markov Model

In Hidden Markov model (HMM), system being modeled is assumed to be a Markov process with unobserved (hidden) states. The Hidden Markov Model (HMM) is a variant of a *finite state machine* having a set of hidden *states*, Q, an output *alphabet* (observations), O, transition probabilities, A, output (emission) probabilities, B, and initial state probabilities, Π. The current state is not observable. Instead, each state produces an output with a certain probability (B). Usually the states, Q, and outputs, O, are understood, so an HMM is said to be a triple, (A, B, Π).



Canonical problems

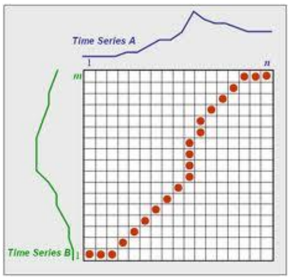
There are 3 canonical problems to solve with HMMs:

1. Given the model parameters, compute the probability of a particular output sequence. This problem is solved by the Forward and Backward algorithms.
2. Given the model parameters, find the most likely sequence of (hidden) states which could have generated a given output sequence. Solved by the Viterbi algorithm and posterior decoding.
3. Given an output sequence, find the most likely set of state transition and output probabilities. Solved by the Baum-Welch algorithm

Embedded Reestimation: In embedded reestimation the large HMMs are assembled from smaller individual models. This is done by converting the grammar into a graph representation, then replacing each node in the graph with the appropriate model. Embedded reestimation is done in cases where the dataset is large and which can have any sequence of output symbols. For example, in the case of character recognition, the model can take any character or word in each state and we can't predict a particular sequence of characters. So in this case embedded reestimation is done. Embedded reestimation is mainly done in speech based classification problems.

2.4 DTW: Dynamic Time Warping

Dynamic time warping (DTW) algorithm is for measuring similarity between two temporal sequences that may vary in time or speed. It aims at aligning two sequences of feature vectors by warping the time axis iteratively until an optimal match (according to a suitable metrics usually Euclidian distance) between the two sequences is found.



The two sequences can be arranged on the sides of a grid, with one on the top and the other up the left hand side. Both sequences start on the bottom left of the grid. Inside each cell a distance measure can be placed, comparing the corresponding elements of the two sequences. To find the best match or alignment between these two sequences one need to find a path through the grid that minimizes the total distance between them. The constraints allow restricting the moves that can be made from any point in the path and so limit the number of paths that need to be considered. Most commonly used constrain is Monotonic condition (i.e., the path will not turn back on itself, both the i and j indexes either stay the same or increase, they never decrease)

Computing the DTW requires O (N^2) in general.

2.5 Parzen Window Method

Parzen window density estimation is a Nonparametric Method. Given an instance of the random sample, **x**, Parzen-windowing estimates the PDF ***P(x)*** from which the sample was derived. It essentially superposes kernel functions placed at each observation. Suppose that we want to estimate the value of the PDF ***P(x)*** at point ***x***. Then, we can place a window function at ***x*** and determine how many observations $\boldsymbol{x_i}$ fall within our window or, rather, what is the contribution of each observation $\boldsymbol{x_i}$ to this window

Most frequently used window functions are:

- Hypersphere window

$$\varphi\left(\frac{\vec{x}-\vec{x_0}}{h}\right)=\begin{cases}1, & ||\frac{\vec{x}-\vec{x_0}}{h}||<1\\ 0, & else\end{cases}$$

- Gaussian kernel

$$P(x)=\frac{1}{n}\sum_{i=1}^{n}\frac{1}{(h\sqrt{2\pi})^d}\exp\left(-\frac{1}{2}\left(\frac{x-x_i}{h}\right)^2\right),$$

2.6 K-Nearest Neighbours

A k-Nearest neighbor is a supervised algorithm, which basically counts the k-nearest features to determine the class of a sample. The classifiers do not use any model to fit. Given a query, KNN counts the k nearest neighbor points and decide on the class that takes the majority of votes. It is a simple and efficient algorithm that only calculates the distance of a new sample to the nearest neighbors. Assign category based on majority vote of k nearest neighbor. The distance can be chosen as Euclidean distance.

K nearest neighbors is almost same as the Parzen window method, the only difference is that the window size or the kernel function changes to accommodate the K number of nearest neighbours. So the equation of K nearest neighbours is same as the Parzen window, the only condition is that the volume parameter will change to accordingly.

2.7 FDA: Fisher Discriminant Analysis

FLDA seeks to reduce dimensionality while preserving as much of the class discriminatory information as possible. We seek to obtain a scalar y by projecting the samples x onto a line $y = w^T x$

Of the all possible lines we have to select the one that separates the scalars maximum. In FLDA we aim to increase the distance between the means and decrease the within class scatter.

The Fisher linear discriminant is defined as a linear function $w^T x$ that maximizes the criterion function

$$J(w) = \frac{|\tilde{\mu}_1 - \tilde{\mu}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

To get an optimum w*, we must express J(w) as a function of w. Then we find S_W and S_B, the within class scatter matrix and between class scatter matrix respectively.

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

Solve the generalized eigen value problem yields

$$w^* = \arg\max\left[\frac{w^T S_B w}{w^T S_W w}\right] = S_W^{-1}(\mu_1 - \mu_2)$$

FLDA helps to reduce the dimension of the data and do the classification.

2.8 Perceptron based Classifier

Perceptron Classifier is Gradient Descent Procedure. In Gradient Descent Procedure a we define criterion function J(a) that is minimized if a is a solution vector. Thus problem reduces to one of minimizing a scalar function. We start with some arbitrarily chosen weight vector a(1) and compute the gradient vector ∇J(a(1)). Then next value a(2) is obtained by moving some distance from a(1) in the direction of steepest descent, i.e., along the negative of the gradient.

The Perceptron Criterion Function is defined by the number of samples misclassified by weight vector a. Thus aim of Perceptron based Classifier is move weight vector such that misclassification is reduced.

In the multiclass classification with K classes, we will maintain a set of K weight vectors $w_1,\ldots,w_K$. The prediction (both at training and test time) done by

$$\hat{y}_n = \arg\max_k(\boldsymbol{w}_k^{!}\boldsymbol{x}_n + b)$$

The update condition is given by (assuming that y_n is the true label of x_n)

$$\text{if}(\hat{y}_n \neq y_n)$$
$$\begin{aligned} \boldsymbol{w}_{\hat{y}_n} &= \boldsymbol{w}_{\hat{y}_n} - \boldsymbol{x}_n \\ \boldsymbol{w}_{y_n} &= \boldsymbol{w}_{\hat{y}_n} + \boldsymbol{x}_n \end{aligned}$$

2.9 SVM: Support Vector Machine

The goal in training a Support Vector Machine is to find the separating hyperplane with the largest margin. Generally, a support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

Implementation

GMM: Gaussian Mixture Model

- Get dataset for each class from file (in preprocessing stage we separated the classes)
- Separation of Training data and Test data: randomly divide each class into training and test data, where 75% is training data and 25% is test data.
- Calculation of GMM parameters (EM Algorithm): Find the means μ_k, covariance Σ_k and mixing coefficients π_k, for each cluster of classes from training data. Initial parameters are calculated using K-means EM Algorithm (No of Iteration < 15), then 2/3 iteration of GMM EM algorithm is applied to get fine parameters: means μ_k, covariance Σ_k and mixing coefficients π_k,
- Classification of test data: Calculate the likelihood for each feature vectors in test data (for each class). Predict class of data such that likelihood (of that feature vectors) is maximum for that class.
- Confusion matrix and Accuracy: After classification of each set of feature vectors of test data, update the confusion matrix and accuracy.

HMM: Hidden Markov Model

(a) Discrete HMM

- The dataset is Telugu character coordinates.
- The 6 features are extracted from the coordinates, normalizing the x-y coordinates, first order derivative and second order derivative.
- Normalized coordinates are find using the equation $x'=(x-x_{min})/(x_{max}=x_{min})$
- First order derivative and second order derivatives are similarly calculated.
- Form a new dataset using the new features.
- Do K-means Clustering for finding out the symbols corresponding to each state.
- Get dataset for each class from file
- Separation of Training data and Test data: randomly divide each class into training and test data, where 75% is training data and 25% is test data.
- Calculation of K-Means (EM Algorithm): Find the means μ_k for each cluster from training data, taking all training data together. Initially distance calculation is done using

Euclidean distance, then mahalanobis distance is followed if need (if no of iteration exceeds 12)

- Creation of sequence vector using K-Means labels
- Train DHMM for each class using these sequence vectors of corresponding classes.
- Create test sequence from test data sample, using K-means labels. Give this test sequence to DHMM model created using above model.
- Confusion matrix and Accuracy: After classification of each set of feature vectors of test data, update the confusion matrix. Calculate the accuracy.

(b) Continuous HMM: Using HTK toolkit

- Make the filelist and the corresponding label.
- Write the grammar corresponding to the digit sequence.
- Convert the mfcc file to htk format.
- Generate the proto file for each digit.
- Using HMM Tool Kit generate HMM model and test the same.

DTW: Dynamic Time Warping

- Get dataset for each class from file (in preprocessing stage we separated the classes)
- Separation of Training data and Test data: randomly divide each class into training and test data. Since as no: of reference vectors sequence increases execution time increase exponentially, we limited no of reference vector sequence.
- Classification of test data: Calculate the distance for each feature vectors sequence in test data with respect to all reference vector of a class and then take average of distance. Predict class of data such that distance (of that feature vectors) is minimum for that class.
- Confusion matrix and Accuracy: After classification of each set of feature vectors of test data, update the confusion matrix. Calculate the accuracy.

Parzen Window Classifier

- Load all data to the Database.
- Input the size (h) for hypersphere and the variance (h) for Gaussian kernel.

(a) Hypersphere

- Calculate the Euclidean distance between the selected test vector and the vectors in the train dataset.
- Select those points that fall under the hypersphere.
- Calculate the probability of each class based on the points selected.
- Multiply the probability of points on vectors from the same data file.
- Find the class belonging to the maximum probability.

(b) Gaussian kernel

- Calculate the probability of the vector using the Gaussian kernel function corresponding to all the train data belonging to each class seperately.

- Multiply the probability of points on vectors from the same data file.
- Find the maximum and classify the test vector to that class.

K-Nearest Neighbours Method

- Load all the training data.
- Take each test vector and calculate the distance between all the train data.
- Take the K nearest points and calculate the probability of each class.
- Multiply the probability of points on vectors from the same data file.
- Calculate the maximum probability and classify the test to that particular class.

FDA: Fisher Discriminant Analysis

- Find the S_W, within class scatter matrix.
- Calculate the S_B, between class scatter matrix.
- Find the Eigen values and corresponding Eigen vectors using generalized eigen value decomposition equation.
- The eigen vectors corresponding to the top C-1 eigen values gives the C-1 lines onto which the data points can be projected, such a way that the means are separated and the variance is minimum.
- Project the data points to the C-1 dimensional space.
- Classify the test data using bayesian classifier.

Perceptron based Classifier

- For each class create a augmented weight vector
- Create augmented train vectors.
- Do gradient decent procedure until weight vectors converge (max no: of iteration is set to 2000)
- For each test File create set of augmented vectors, classify each vector and use maximum voting for determining class of File.

SVM: Support Vector Machine

- SVM classification is done using libsvm package, linear kernel.
- Tried svm using different types of svm types; C-svm, µ-svm.

3. Observations

Dataset1: Image Data

Dataset has 8 classes of data.
Classes are 'coast', 'forest', 'highway', 'insidecity', 'mountain', 'opencountry', 'street', 'tallbuildings'.

1. *Parzen Window Method*

(a) Hypersphere

Window Size	0.5	0.75	1	1.25
Accuracy	36.84	48.43	53.34	47.99

(b) Gaussian Kernel

Value of h	0.05	0.1	0.2	0.25	0.3	0.4	0.5
Accuracy	53.04	56.01	57.65	57.50	56.46	53.49	50.37

2. *K-Nearest Neighbours Classifier*

Value:K	1000	500	100	20	10	5	1
Accuracy	46.50	51.26	56.01	59.73	60.02	55.86	43.98

3. *Fisher Discriminant Analysis*

	Image as Normal Vector			Image as Super Vector		
Diff Trails	1	2	3	1	2	3
Accuracy	65.53	63.30	62.26	67.01	64.34	62.56

4. *Perceptron based Learning*

Different Trials	1	2
Accuracy	37.89	32.52

5. *Support Vector Machine*

Different Trials	1	2	3
Accuracy	62.11	57.80	53.05

6. *GMM*

No of Clusters	2	5	9	10	12
Accuracy	62.85	63.89	62.85	64.19	63.00

Dataset2: Tidigit Dataset

1. *HMM – Embedded Reestimation*

```
====================== HTK Results Analysis =======================
  Date: Tue Dec  3 21:34:47 2013
  Ref : AllTest.mlf
  Rec : recout.mlf
------------------------ Overall Results --------------------------
SENT: %Correct=9.09 [H=791, S=7909, N=8700]
WORD: %Corr=62.86, Acc=-160.05 [H=5469, D=0, S=3231, I=19393, N=8700]
===================================================================
```

```
===================== HTK Results Analysis ======================
  Date: Tue Dec  3 22:28:48 2013
  Ref : AllTest.mlf
  Rec : recout.mlf
------------------------ Overall Results --------------------------
SENT: %Correct=9.26 [H=806, S=7894, N=8700]
WORD: %Corr=62.49, Acc=-157.54 [H=5437, D=0, S=3263, I=19143, N=8700]
```

Dataset3: Handwritten Dataset

The coordinates of 7 Telugu characters are given.
We need to find the features out of the dataset. The feature selected are Normalized coordinates, First order derivative and Second order derivative.

1. *HMM*

No. Clusters	10	10	10	30	40
No. Symbols	10	10	10	30	40
States:Class1	10	10	10	10	10
States:Class2	10	10	10	10	10
States:Class3	15	14	13	13	13
States:Class4	12	12	13	13	13
States:Class5	10	10	10	10	10
States:Class6	12	12	12	12	12
States:Class7	10	10	13	13	13
Accuracy	80.46	81.03	82.13	89.08	86.21

Dataset4: Music Dataset

1. *DTW*

	Confusion Matrix		
	A	B	C
A	41	12	5
B	0	18	2
C	5	5	10

Accuracy = 70.40

4. Inferences

- In Parzen Window the Gaussian Kernel has more accuracy compared with Hypersphere, But Gaussian takes much time compared to Hypershpere.
- K nearest neighbours need just around 10, near neighbours to predict the class with almost good accuracy.
- Even 1 nearest neighbor gives good result in K nearest neighbor, but as the number of neighbor increases the accuracy first increases then decreases.
- FLDA assumes that the covariances of all classes are same. FLDA gives good accuracy without compromising the time performance.
- In GMM, accuracy largely depends upon; selection of K. Every GMM has an optimal K, which gives minimal error.
- Perceptron is very slow to converge and the accuracy is very low. If 'eta' is a function of iteration (eta decreases after each iteration), then we get faster convergence without affecting much on accuracy.
- Even though slope is a good feature in handwritten dataset, slope can be infinity, which may affect HMM. So normalized first and second derivatives become good choice for features.
- In DHMM, accuracy increases as the number of symbols increases.
- Mandi dataset was very noisy and hence DTW doesn't give good accuracy.

5. References

1. Pattern Classification 2ed: Richard O. Duda, Peter E. Hart, David G. Stork
2. Pattern Recognition and Machine Learning, Christopher M. Bishop
3. General references http://en.wikipedia.org , http://www.mathworks.com/
4. Perceptron for Imbalanced Classes and Multiclass Classification.
   http://www.cs.utah.edu/~piyush/teaching/imbalanced_multiclass_perceptron.pdf
5. HMM-based Online Handwriting Recognition System for Telugu Symbols Jagadeesh Babu V, Prasanth L, Raghunath Sharma R, Prabhakara Rao G.V., Bharath A HP Laboratories India. http://www.hpl.hp.com/techreports/2007/HPL-2007-107.pdf

Appendix A: Matlab Code

Misc

file:loadData.m

```
function [classNo,classList,testData,trainData,trainLabel] = loadData(datatype)

% LoadData - Load data for problems given the question number as input
% Input - datatype : Type of data to be loaded
%           1. Image
%           2. Linearly seperable dataset
%           ...
% Output - data : Data of different classes as cell array

switch datatype
    case 'image'
        % Load Filelist of the Image Dataset
        % dataPath : Path of Image Dataset
        if exist('../data/nonparametric/image','dir')
            dataPath = '../data/nonparametric/image';
        elseif exist('../data/gmm/image','dir')
            dataPath = '../data/gmm/image';
        else
            dataPath = uigetdir;
        end
        % classList{} : List of Classes
        classList = dir(dataPath);
        classList = {classList(~strncmpi({classList.name},'.',1)).name};
        % classNo : No of Classes
        [~,classNo] = size(classList);
        % classSize() : Size of each Class
        classSize = zeros(1,classNo);
        % fileList{} : List of Files in each Class
        fileList{classNo} = {};
        % K : Percent of Test Data
        K = input('Percent of Test Data: ')/100;
        % trainInd{} : Index of Files in Training for each Class
        trainInd{classNo} = {};
        % testInd{} : Index of Files in Testing for each Class
        testInd{classNo} = {};
        % testData{} : Test Data of each Class
        testData{classNo} = {};
        % trainData : Training Data
        trainData = [];
        % trainLabel : Label of Training Data
        trainLabel = [];
        for i = 1:classNo
            fileList{i} = dir([dataPath '/' classList{i}]);
            fileList{i} = {fileList{i}(~strncmpi({fileList{i}.name},'.',1)).name};
            classSize(i) = size(fileList{i},2);
            % Divide Image Dataset to Train and Test
            Index = randperm(classSize(i));
            % numTest : Number of Test Files per Class
            numTest = round(classSize(i)*K);
            numTrain = classSize(i)-numTest;
            testInd{i} = sort(Index(1:numTest));
            trainInd{i} = sort(Index(numTest+1:classSize(i)));
            for j = 1:numTest
                testData{i}{j} =
load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(testInd{i}(j)))));
            end
            % Temporary Variables
            tempLabel = [];
            [p,q] = size(testData{i}{j});
            tempTrain = zeros(numTrain*p,q);
```

```
            for j = 1:numTrain
                tempTrain((j-1)*p+1:j*p,:) =
load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(trainInd{i}(j)))));
            end
            trainData = [trainData;tempTrain];
            p = size(tempTrain,1);
            tempLabel(1:p) = i;
            trainLabel = [trainLabel tempLabel];
            clear p j tempData tempLabel numTest numTrain Index;
        end
        clear i K dataPath classSize testInd trainInd fileList;

    case 'image_flda'
        % Load Filelist of the Speaker Dataset
        % dataPath : Path of Speaker Dataset
        if exist('../data/nonparametric/image','dir')
            dataPath = '../data/nonparametric/image';
        elseif exist('../data/gmm/image','dir')
            dataPath = '../data/gmm/image';
        else
            dataPath = uigetdir;
        end
        % classList{} : List of Classes
        classList = dir(dataPath);
        classList = {classList(~strncmpi({classList.name},'.',1)).name};
        % classNo : No of Classes
        classNo = size(classList,2);
        % classSize() : Size of each Class
        classSize = zeros(1,classNo);
        % fileList{} : List of Files in each Class
        fileList{classNo} = {};
        % K : Percent of Test Data
        K = input('Percent of Test Data: ')/100;
        % trainInd{} : Index of Files in Training for each Class
        trainInd{classNo} = {};
        % testInd{} : Index of Files in Testing for each Class
        testInd{classNo} = {};
        % testData{} : Test Data of each Class
        testData{classNo} = [];
        % trainData : Training Data
        trainData = [];
        % trainLabel : Label of Training Data
        trainLabel = [];

        for i = 1:classNo
            fileList{i} = dir([dataPath '/' classList{i}]);
            fileList{i} = {fileList{i}(~strncmpi({fileList{i}.name},'.',1)).name};
            classSize(i) = size(fileList{i},2);
            % Divide Image Dataset to Train and Test
            Index = randperm(classSize(i));
            % numTest : Number of Test Files per Class
            numTest = round(classSize(i)*K);
            numTrain = classSize(i)-numTest;
            testInd{i} = sort(Index(1:numTest));
            trainInd{i} = sort(Index(numTest+1:classSize(i)));
            for j = 1:numTest
                testData{i}(j,:) =
makeLine(load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(testInd{i}(j)))))
);
            end
            % Temporary Variables
            tempTrain = [];
            tempLabel = [];
            for j = 1:numTrain
                tempTrain =
```

```
[tempTrain;makeLine(load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(trainI
nd{i}(j))))))];
            end
            trainData = [trainData;tempTrain];
            p = size(tempTrain,1);
            tempLabel(1:p) = i;
            trainLabel = [trainLabel tempLabel];
            clear p j tempData tempLabel numTest numTrain Index;
        end
        clear i K dataPath classSize testInd trainInd fileList;


    case 'speaker'
        % Load Filelist of the Speaker Dataset
        % dataPath : Path of Speaker Dataset
        if exist('../data/gmm/speaker','dir')
            dataPath = '../data/gmm/speaker';
        else
            dataPath = uigetdir;
        end
        % classList{} : List of Classes
        classList = dir(dataPath);
        classList = {classList(~strncmpi({classList.name},'.',1)).name};
        % classNo : No of Classes
        classNo = size(classList,2);
        % classSize() : Size of each Class
        classSize = zeros(1,classNo);
        % fileList{} : List of Files in each Class
        fileList{classNo} = {};
        % K : Percent of Test Data
        K = input('Percent of Test Data: ')/100;
        % trainInd{} : Index of Files in Training for each Class
        trainInd{classNo} = {};
        % testInd{} : Index of Files in Testing for each Class
        testInd{classNo} = {};
        % testData{} : Test Data of each Class
        testData{classNo} = {};
        % trainData : Training Data
        trainData = [];
        % trainLabel : Label of Training Data
        trainLabel = [];

        for i = 1:classNo
            fileList{i} = dir([dataPath '/' classList{i}]);
            fileList{i} = {fileList{i}(~strncmpi({fileList{i}.name},'.',1)).name};
            classSize(i) = size(fileList{i},2);
            % Divide Image Dataset to Train and Test
            Index = randperm(classSize(i));
            % numTest : Number of Test Files per Class
            numTest = round(classSize(i)*K);
            numTrain = classSize(i)-numTest;
            testInd{i} = sort(Index(1:numTest));
            trainInd{i} = sort(Index(numTest+1:classSize(i)));
            for j = 1:numTest
                testData{i}{j} =
load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(testInd{i}(j)))));
            end
            % Temporary Variables
            tempTrain = [];
            tempLabel = [];
            for j = 1:numTrain
                tempTrain =
[tempTrain;load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(trainInd{i}(j))
)))];
            end
```

```
            trainData = [trainData;tempTrain];
            p = size(tempTrain,1);
            tempLabel(1:p) = i;
            trainLabel = [trainLabel tempLabel];
            clear p j tempData tempLabel numTest numTrain Index;
        end
        clear i K dataPath classSize testInd trainInd fileList;

    otherwise
        error('Unrecognized Dataset');
end
end
```

file:fMea.m

```
function [avgPre,avgRec,avgSpe,avgFmea,totAcc] = fMea(confMat)
% Function to calculate Accuracy, Precision, Recall, F Measure
% Input : confMat - Confusion Matrix
disp('Confusion Matrix');
disp(confMat);
n = size(confMat,1);
pre = zeros(1,n);
rec = zeros(1,n);
fmea = zeros(1,n);
spe = zeros(1,n);
totAcc= sum(diag(confMat))/sum(sum(confMat));
for i = 1:n
    pre(i) = confMat(i,i)/sum(confMat(:,i));
    rec(i) = confMat(i,i)/sum(confMat(i,:));
    fmea(i) = 2*pre(i)*rec(i)/(pre(i)+rec(i));
    temp = confMat;
    temp(i,:) = [];
    den = sum(sum(temp));
    temp(:,i) = [];
    num = sum(sum(temp));
    spe(i) = num/den;
    %fprintf('Precision (Class %d): %d\n',i,pre(i)*100);
    %fprintf('Recall or Sensitivity (Class %d): %d\n',i,rec(i)*100);
    %fprintf('Specificity (Class %d): %d\n',i,spe(i)*100);
    %fprintf('F-Measure (Class %d): %d\n\n',i,fmea(i)*100);
end
avgPre = sum(pre)/n;
avgRec = sum(rec)/n;
avgFmea = sum(fmea)/n;
avgSpe = sum(spe)/n;
%fprintf('Average Precision = %f\n',avgPre*100);
%fprintf('Average Recall = %f\n',avgRec*100);
%fprintf('Average Specificity = %f\n',avgSpe*100);
%fprintf('Average F-Measure = %f\n',avgFmea*100);
%fprintf('Total Accuracy = %f\n',totAcc*100);
fprintf('Precision = %f\n',avgPre*100);
fprintf('Recall = %f\n',avgRec*100);
fprintf('Specificity = %f\n',avgSpe*100);
fprintf('F-Measure = %f\n',avgFmea*100);
fprintf('Accuracy = %f\n',totAcc*100);
end
```

Parzen Window

file:parzenMain.m

```
%Parzen Window Main

[classNo,classList,testData,trainData,trainLabel] = loadData('image');
h = input('Input the Window Size: ');
windowType = input('Input the Window Type(hypersphere(1)/gaussian(2)): ');
```

```
confMat = zeros(classNo);
for i = 1:classNo
    % Class
    switch(windowType)
        case 1
            for j = 1:size(testData{i},2)
                % File
                class = parzenWindow(testData{i}{j}, ...
                    trainData,trainLabel',h,classNo,'hypersphere');

                confMat(i,class) = confMat(i,class) + 1;
            end
        case 2
            for j = 1:size(testData{i},2)
                % File
                class = parzenWindow(testData{i}{j}, ...
                    trainData,trainLabel',h,classNo,'gaussian');
                confMat(i,class) = confMat(i,class) + 1;
            end
        otherwise
            error('Unknown Type');
    end
    fprintf('Classification of class "%s" finished\n',classList{i});
end
accuracy = sum(diag(confMat))/sum(sum(confMat));
disp(confMat);
fprintf('Accuracy = %f\n',accuracy);
clear i j class classNo classList testData trainData trainLabel;
```

file:parzenWindow.m

```
%% Parzen Window
%
%%
function [ class ] = parzenWindow(testVectors,refVectors,refLabel,h,C,type)
%parzenWindow Parzen Window Classifer
%
%Input
% testVectors
% refVectors
% refLabel
% h
% C
% type
%
%Output:
% loglikhood

likelihood = zeros(1,C);

switch(type)
    case 'hypersphere'
        windows = rangesearch(refVectors,testVectors,h);
        for j = 1:size(testVectors,1)
            prob = ones(1,C);
            window = windows{j};
            K = size(window,2)+1;
            windowLabels = refLabel(window);
            prob = prob + sum(bsxfun(@eq,windowLabels,1:C),1);
            prob = prob./K;
            prob = log(prob);
            likelihood = likelihood + prob;
        end

    case 'gaussian'
```

```
        [N,d] = size(testVectors);
        denominator = (2*pi*h*h)^(d/2);
        for i = 1:N
            prob = zeros(1,C);
            for j = 1:C
                classPoints = refVectors(refLabel == j,:);
                exponent = bsxfun(@minus,classPoints,testVectors(i,:));
                exponent = exponent./h;
                exponent = sum((exponent.*exponent),2)./2;
                prob(j) = sum(exp(-exponent))/denominator;
            end
            K = size(refVectors,1);
            prob = log(prob./K);
            likelihood = likelihood + prob;
        end
    otherwise
        error('Unknown option');
end;
[~,class] = max(likelihood);
end

%%
```

K Nearest Neighbours

file:kNearNMain.m

```
%K Nearest Neighbour Main

[classNo,classList,testData,trainData,trainLabel] = loadData('image');
K = input('Input No of Nearest Neighbours (K): ');
confMat = zeros(classNo);
for i = 1:classNo
    % Class
    for j = 1:size(testData{i},2)
        % File
        prob = zeros(1,classNo);
        for k = 1:size(testData{i}{j},1)
            % Vector
            prob = prob +
kNearN(testData{i}{j}(k,:),trainData,trainLabel',K,classNo);
        end
        [~,class] = max(prob);
        confMat(i,class) = confMat(i,class) + 1;
    end
    fprintf('Classification of class "%s" finished\n',classList{i});
end
accuracy = sum(diag(confMat))/sum(sum(confMat));
disp(confMat);
fprintf('Accuracy = %f\n',accuracy);
clear i j k class classNo classList testData trainData trainLabel prob;
```

file:kNearN.m

```
function prob = kNearN(testVector,refVectors,refLabel,k,C)

N = size(refVectors,2);
prob = ones(1,C);
[~,window] = pdist2(refVectors,testVector,'euclidean','Smallest',k);
K = size(window,2)+1;
windowLabels = refLabel(window);
prob = prob + sum(bsxfun(@eq,windowLabels,1:C),1);
prob = prob./K;
prob = log(prob);
if N == 2
    %plot data
```

```
    cmap = hsv(C);
    for j = 1:C
        class = refVectors(refLabel == j,:);
        scatter(class(:,1),class(:,2),...
            '*','CData',cmap(j,:));hold on;
    end
    [~,class] = max(prob);
    plot(testVector(1),testVector(2), ...
        'kx','MarkerSize',12, ...
        'LineWidth',3,'MarkerEdgeColor','black');
    viscircles(testVector,h,'EdgeColor',[1 1 1]-cmap(class,:));
end

end
```

FLDA

File:flda.m

```
%function fda()
[classNo,classList,testData,trainData,trainLabel] = loadData('image');
classMean = zeros(classNo,size(trainData,2));
scatterW = zeros(size(trainData,2));scatterB = zeros(size(trainData,2));
mean = sum(trainData)/size(trainData,1);
for i = 1:classNo
    classMean(i,:) = sum(trainData(trainLabel==i,:))/sum(trainLabel==i);
    temp = bsxfun(@minus,trainData(trainLabel==i,:),classMean(i,:));
    scatterW = scatterW + (temp'*temp);
    temp = classMean(i,:) - mean;
    scatterB = scatterB + (temp'*temp) * sum(trainLabel==i);
end
[W,D] = eig(scatterB,scatterW);
[~,index] = sort(diag(D),'descend');
W = W(:,index(1:classNo-1));
projTrain = trainData*W;
for i = 1:classNo
    for j = 1:size(testData{i},2)
        projTest{i}{j} = testData{i}{j}*W;
    end
end

projMean = zeros(classNo,classNo-1);
projCov{classNo} = {};
for i = 1:classNo
    fprintf('Class "%s" : Starting Training...\n',classList{i});
    projMean(i,:) = sum(projTrain(trainLabel==i,:))/sum(trainLabel==i);
    projCov{i} = cov(projTrain(trainLabel==i,:));
    fprintf('Class "%s" : Finished Training \n',classList{i});
end
%Confusion Matrix
ConfMat = zeros(classNo);
%Classification Accuracy
Accu(1:classNo) = 0;

disp('Starting Testing...');

for s = 1:classNo
    fprintf('Testing Class "%s"...\n',classList{s});
    for i = 1:size(projTest{s},2)
        likelihood=zeros(1,classNo);
        for j = 1:size(projTest{s}{i},1)
            for k = 1:classNo
```

```
                %likelihood(k) = (projTest{s}(i,:)-
projMean(k,:))/projCov{k}*(projTest{s}(i,:)-
projMean(k,:))'+log(abs(det(projCov{k})));
                %likelihood(k) = -likelihood(k)/2 +
log(sum(trainLabel==i)/size(trainLabel,2));
                likelihood(k) = likelihood(k) +
log(mvnpdf(projTest{s}{i}(j,:),projMean(k,:),projCov{k}));
            end
        end
        [~,class] = max(likelihood);
        ConfMat(s,class) = ConfMat(s,class) + 1;
    end
end

for s = 1:classNo
    Accu(s) = ConfMat(s,s) / sum(ConfMat(s,:));
end
AvgAccu = sum(diag(ConfMat))/sum(sum(ConfMat));

clc;

disp('Confusion Matrix');
disp(ConfMat);
disp('Classification Accuracy');
disp(Accu);
disp('Average Accuracy');
disp(AvgAccu);

clear D W i index scatterB scatterW temp;
%end
```

Perceptron

File:perceptronData.m

```
function [classNo,classList,testData,testLabel,trainData,trainLabel] =
perceptronData()
% Load Filelist of the Speaker Dataset
% dataPath : Path of Speaker Dataset
if exist('../data/nonparametric/image','dir')
    dataPath = '../data/nonparametric/image';
elseif exist('../data/gmm/image','dir')
    dataPath = '../data/gmm/image';
else
    dataPath = uigetdir;
end
% classList{} : List of Classes
classList = dir(dataPath);
classList = {classList(~strncmpi({classList.name},'.',1)).name};
% classNo : No of Classes
classNo = size(classList,2);
% classSize() : Size of each Class
classSize = zeros(1,classNo);
% fileList{} : List of Files in each Class
fileList{classNo} = {};
% K : Percent of Test Data
K = input('Percent of Test Data: ')/100;
% trainInd{} : Index of Files in Training for each Class
trainInd{classNo} = {};
% testInd{} : Index of Files in Testing for each Class
testInd{classNo} = {};
% testData{} : Test Data of each Class
testData = [];
% testLabel : Label for Test Data
testLabel = [];
```

```
% trainData : Training Data
trainData = [];
% trainLabel : Label of Training Data
trainLabel = [];

for i = 1:classNo
    fileList{i} = dir([dataPath '/' classList{i}]);
    fileList{i} = {fileList{i}(~strncmpi({fileList{i}.name},'.',1)).name};
    classSize(i) = size(fileList{i},2);
    % Divide Image Dataset to Train and Test
    Index = randperm(classSize(i));
    % numTest : Number of Test Files per Class
    numTest = round(classSize(i)*K);
    numTrain = classSize(i)-numTest;
    testInd{i} = sort(Index(1:numTest));
    trainInd{i} = sort(Index(numTest+1:classSize(i)));

    tempTest = [];
    tempLabel = [];
    for j = 1:numTest
        tempTest =
[tempTest;makeLine(load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(testInd
{i}(j))))))];
    end
    testData = [testData;tempTest];
    p = size(tempTest,1);
    tempLabel(1:p) = i;
    testLabel = [testLabel tempLabel];
     % Temporary Variables
    tempTrain = [];
    tempLabel = [];
    for j = 1:numTrain
        tempTrain =
[tempTrain;makeLine(load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(trainI
nd{i}(j))))))];
    end
    trainData = [trainData;tempTrain];
    p = size(tempTrain,1);
    tempLabel(1:p) = i;
    trainLabel = [trainLabel tempLabel];
    clear p j tempTrain tempTest tempLabel numTest numTrain Index;
end
testLabel = testLabel';
trainLabel = trainLabel';
clear i K dataPath classSize testInd trainInd fileList;

end

function line=makeLine(matrix)
n = size(matrix,1);
line = [];
for i = 1:n
    line = [line matrix(i,:)];
end
end
```

File:perceptron.m

```
% Perceptron based Learning
[classNo,classList,testData,testLabel,trainData,trainLabel] = perceptronData();

%Initialize Weight
W = 0.1*randn(classNo,size(trainData,2));
learningrate = 0.2;
var = zeros(classNo,size(trainData,1));
```

```
for i=1:classNo
    for j=1:size(trainData,1)
        if trainLabel(j)==i
            var(i,j)=1;
        else
            var(i,j)=-1;
        end
    end
end

trail = input('No of trails: ');

for times=1:trail
    for K=1:classNo
        prob=zeros(1,size(trainData,1));
        for i=1:size(trainData,1)
            if (sign(W(K,:)*trainData(i,:)')~=sign(var(K,i)))
                W(K,:)=W(K,:)+(trainData(i,:)'*var(K,i)'*1.5)';
            end
        end
    end
end

count=0;
predict=zeros(1,size(testData,1));

for i=1:size(testData,1)
    prob=zeros(1,classNo);
    for j=1:classNo
        prob(j)=W(j,:)*testData(i,:)';
    end
    [~,ind]=max(prob);
    count=count+1;
    predict(count)=ind;
end

for i = 1:classNo
    for j = 1:classNo
        confMat(i,j) = sum(predict(testLabel==i)==j);
    end
end
```

SVM

File:svmData.m

```
function [classNo,classList,testData,testLabel,trainData,trainLabel] = svmData()
% Load Filelist of the Speaker Dataset
% dataPath : Path of Speaker Dataset
if exist('../data/nonparametric/image','dir')
    dataPath = '../data/nonparametric/image';
elseif exist('../data/gmm/image','dir')
    dataPath = '../data/gmm/image';
else
    dataPath = uigetdir;
end
% classList{} : List of Classes
classList = dir(dataPath);
classList = {classList(~strncmpi({classList.name},'.',1)).name};
% classNo : No of Classes
classNo = size(classList,2);
% classSize() : Size of each Class
classSize = zeros(1,classNo);
% fileList{} : List of Files in each Class
fileList{classNo} = {};
```

```
% K : Percent of Test Data
K = input('Percent of Test Data: ')/100;
% trainInd{} : Index of Files in Training for each Class
trainInd{classNo} = {};
% testInd{} : Index of Files in Testing for each Class
testInd{classNo} = {};
% testData{} : Test Data of each Class
testData = [];
% testLabel : Label for Test Data
testLabel = [];
% trainData : Training Data
trainData = [];
% trainLabel : Label of Training Data
trainLabel = [];

for i = 1:classNo
    fileList{i} = dir([dataPath '/' classList{i}]);
    fileList{i} = {fileList{i}(~strncmpi({fileList{i}.name},'.',1)).name};
    classSize(i) = size(fileList{i},2);
    % Divide Image Dataset to Train and Test
    Index = randperm(classSize(i));
    % numTest : Number of Test Files per Class
    numTest = round(classSize(i)*K);
    numTrain = classSize(i)-numTest;
    testInd{i} = sort(Index(1:numTest));
    trainInd{i} = sort(Index(numTest+1:classSize(i)));

    tempTest = [];
    tempLabel = [];
    for j = 1:numTest
        tempTest =
[tempTest;makeLine(load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(testInd
{i}(j))))))];
    end
    testData = [testData;tempTest];
    p = size(tempTest,1);
    templabel(1:p) = i;
    testLabel = [testLabel templabel];
     % Temporary Variables
    tempTrain = [];
    tempLabel = [];
    for j = 1:numTrain
        tempTrain =
[tempTrain;makeLine(load(char(strcat(dataPath,'/',classList{i},'/',fileList{i}(trainI
nd{i}(j))))))];
    end
    trainData = [trainData;tempTrain];
    p = size(tempTrain,1);
    tempLabel(1:p) = i;
    trainLabel = [trainLabel tempLabel];
    clear p j tempTrain tempTest tempLabel numTest numTrain Index;
end
testLabel = testLabel';
trainLabel = trainLabel';
clear i K dataPath classSize testInd trainInd fileList;

end

function line=makeLine(matrix)
n = size(matrix,1);
line = [];
for i = 1:n
    line = [line matrix(i,:)];
end
end
```

File:SVM.m

```
% SVM code

[classNo,classList,testData,testLabel,trainData,trainLabel] = svmData();
addpath('libsvm-3.17/matlab/');

% Train Model for each class
classModel = cell(classNo,1);
for K = 1:classNo
    fprintf('Training Class %s\n',classList{K});
    classModel{K} = svmtrain(trainLabel, trainData, '-s 4 -t 0 -q');
end
% Test for Accuracy
for K = 1:classNo
    fprintf('Testing Class %s\n',classList{K});
    [predict,~,~] = svmpredict(testLabel, testData, classModel{K},'-q');
end
for i = 1:classNo
    for j = 1:classNo
        confMat(i,j) = sum(predict(testLabel==i)==j);
    end
end
confMat
acc = sum(diag(confMat))/sum(sum(confMat))
```

GMM

File:gmmMainImage .m

```
% Gaussian Mixture Model

[classNo,classList,testData,trainData,trainLabel] = loadData('image');

%No of Clusters
K = zeros(1,classNo);
for s =1:classNo
    K(s) = input(['Enter No. of Clusters(Class "' classList{s} '"): ']);
end

gmm(1:classNo) = GMM(0);
for s = 1:classNo
    gmm(s) = GMM(K(s));
end

for i = 1:classNo
    fprintf('Class "%s" : Starting Training...\n',classList{i});
    gmm(i).train(trainData(trainLabel==i,:));
    fprintf('Class "%s" : Finished Training \n',classList{i});
end


%Confusion Matrix
ConfMatFull = zeros(classNo);
ConfMatNaive = zeros(classNo);
%Classification Accuracy
AccuFull(1:classNo) = 0;
AccuNaive(1:classNo) = 0;

disp('Starting Testing...');

for s = 1:classNo
    fprintf('Testing Class "%s"...\n',classList{s});
```

```
    for i = 1:size(testData{s},2)
        likelihood=zeros(1,classNo);
        for k = 1:classNo
            for j = 1:size(testData{s}{i},1)
                likelihood(k) = likelihood(k) +
gmm(k).getLikelihood(testData{s}{i}(j,:));
            end
        end
        [~,class] = max(likelihood);
        ConfMatFull(s,class) = ConfMatFull(s,class) + 1;

        likelihood=zeros(1,classNo);
        for k = 1:classNo
            for j = 1:size(testData{s}{i},1)
                likelihood(k) = likelihood(k) +
gmm(k).getLikelihood(testData{s}{i}(j,:),true);
            end
        end
        [~,class] = max(likelihood);
        ConfMatNaive(s,class) = ConfMatNaive(s,class) + 1;
    end
end

for s = 1:classNo
    AccuFull(s) = ConfMatFull(s,s) / sum(ConfMatFull(s,:));
    AccuNaive(s) = ConfMatNaive(s,s) / sum(ConfMatNaive(s,:));
end
AvgAccuFull = sum(diag(ConfMatFull))/sum(sum(ConfMatFull));
AvgAccuNaive = sum(diag(ConfMatNaive))/sum(sum(ConfMatNaive));

clc;

for s =1:classNo
    disp(['No. of Clusters(Class "' classList{s} '"): ' num2str(K(s))]);
end

disp('With Full Covariance Matrices')
disp('Confusion Matrix');
disp(ConfMatFull);
disp('Classification Accuracy');
disp(AccuFull);
disp('Average Accuracy');
disp(AvgAccuFull);

disp('With Diagonal Covariance Matrices')
disp('Confusion Matrix');
disp(ConfMatNaive);
disp('Classification Accuracy');
disp(AccuNaive);
disp('Average Accuracy');
disp(AvgAccuNaive);

% clear all;
```

File:gmmMainSpeaker.m

```
% Gaussian Mixture Model

[classNo,classList,testData,trainData,trainLabel] = loadData('speaker');

%No of Clusters
%K = zeros(1,classNo);
%for s =1:classNo
%    K(s) = input(['Enter No. of Clusters(Class "' classList{s} '"): ']);
```

```
%end
K(1:classNo) = input('Enter No. of Clusters: ');

gmm(1:classNo) = GMM(0);
for s = 1:classNo
    gmm(s) = GMM(K(s));
end

disp('Starting Training...');

for i = 1:classNo
    fprintf('Class %d "%s" : Starting Training...\n',i,classList{i});
    while(1)
        try
            gmm(i).train(trainData(trainLabel==i,:));
            break;
        catch Error
            if (strcmp(Error.identifier,'stats:mvnpdf:BadMatrixSigma'))
                K(i) = K(i) - 1;
                disp(K(i));
                gmm(i).resetK(K(i));
            else
                rethrow(Error);
            end
        end
    end
    fprintf('Class %d "%s": Finished Training \n',i,classList{i});
end


%Confusion Matrix
ConfMatFull = zeros(classNo);
ConfMatNaive = zeros(classNo);
%Classification Accuracy
AccuFull(1:classNo) = 0;
AccuNaive(1:classNo) = 0;

disp('Starting Testing...');

for s = 1:classNo
    %fprintf('Testing Class "%s"...\n',classList{s});
    for i = 1:size(testData{s},2)
        likelihood=zeros(1,classNo);
        for k = 1:classNo
            for j = 1:size(testData{s}{i},1)
                likelihood(k) = likelihood(k) +
gmm(k).getLikelihood(testData{s}{i}(j,:));
            end
        end
        [~,class] = max(likelihood);
        ConfMatFull(s,class) = ConfMatFull(s,class) + 1;

        likelihood=zeros(1,classNo);
        for k = 1:classNo
            for j = 1:size(testData{s}{i},1)
                likelihood(k) = likelihood(k) +
gmm(k).getLikelihood(testData{s}{i}(j,:),true);
            end
        end
        [~,class] = max(likelihood);
        ConfMatNaive(s,class) = ConfMatNaive(s,class) + 1;
    end
end
```

```
for s = 1:classNo
    AccuFull(s) = ConfMatFull(s,s) / sum(ConfMatFull(s,:));
    AccuNaive(s) = ConfMatNaive(s,s) / sum(ConfMatNaive(s,:));
end
AvgAccuFull = sum(diag(ConfMatFull))/sum(sum(ConfMatFull));
AvgAccuNaive = sum(diag(ConfMatNaive))/sum(sum(ConfMatNaive));

clc;

%for s =1:classNo
%    disp(['No. of Clusters(Class "' classList{s} '"): ' num2str(K(s))]);
%end

disp('With Full Covariance Matrices')
%disp('Confusion Matrix');
%disp(ConfMatFull);
disp('Classification Accuracy');
disp(AccuFull);
disp('Average Accuracy');
disp(AvgAccuFull);

disp('With Diagonal Covariance Matrices')
%disp('Confusion Matrix');
%disp(ConfMatNaive);
disp('Classification Accuracy');
disp(AccuNaive);
disp('Average Accuracy');
disp(AvgAccuNaive);

% clear all;
```

File:GMM.m

```
classdef GMM < handle
    %GMM Summary of this class goes here
    %   Detailed explanation goes here
    % Usage example:
    %   G = GMM(3);          %3 = no of mixtures
    %   G.train(c)           %c = input Vectors
    %   G.mixureMeans        %to get Means
    %   G.mixureVariance     %to get Variance
    %
    %

    properties (SetAccess = public)
        K; % No: of Mixures
        mixureMeans; % Means of Mixures
        mixureVariance; % Covariances of Mixures
        pi; %pi(k)  = N(k)/sum(N(k))
    end
    properties (Hidden)
        KM;
    end

    methods
        function obj = GMM(k)
            %set K (No: of Mixure)
            %obj = GMM(k)
            %parameters:
            %   k - no of mixture
            obj.K = k;
            obj.pi = zeros(1,k);
            obj.KM = KmeansCluster(k);
```

```
        end %constructor

        function resetK(obj,k)
            %parameters:
            %   k - no of mixture
            obj.K = k;
            obj.pi = zeros(1,k);
            obj.KM = KmeansCluster(k);
        end

        function train(obj,inputVectors)
            %Train GMM
            %Usage: obj.train(inputVectors)
            %parameters:
            % inputVectors - input Vectors (MxN)
            %
            %[M,~] = size(inputVectors);
            [M,N] = size(inputVectors);
            obj.KM.train(inputVectors,12);
            obj.mixureMeans=obj.KM.Kmeans;
            obj.mixureVariance= obj.KM.Kvariance;
            gama = zeros(M,obj.K);

            for j = 1:M
                gama(j,obj.KM.getLabel(inputVectors(j,:))) = ...
                    gama(j,obj.KM.getLabel(inputVectors(j,:))) +1;
            end
            obj.pi = sum(gama)./M;


            for i =(1:3)
                %E step
                for j = 1:M
                    gama(j,:) = obj.getResp(inputVectors(j,:));
                end
                %M step
                Mk = sum(gama);
                for k = 1:obj.K
                    var = zeros(N,N);
                    %for j = 1:M
                    %    mean = mean + inputVectors(j,:).*gama(j,k);
                    %end
                    mean = sum(bsxfun(@times,inputVectors,gama(:,k)));
                    mean = mean./Mk(k);
                    diffVector = bsxfun(@minus,inputVectors,mean);
                    %diff_Gama = bsxfun(@times,diff,gama(:,k));
                    %var = diff(j,:)'*diff_Gama(j,:);
                    for j = 1:M
                        var = var + diffVector(j,:)'*diffVector(j,:).*gama(j,k);
                    end
                    var = var./Mk(k);
                    obj.mixureMeans(k,:) = mean;
                    obj.mixureVariance(:,:,k) = var;
                end
                obj.pi = Mk./M;
            end
        end

        function [ resp ] = getResp(obj,inputVector)
            %Find responsibility of input Vector
            %Usage: obj.getResp(inputVector)
            %parameters:
            % inputVector - input Vector whose responsibility to find
            %
```

```
            k = obj.K;
            resp = zeros(1,k);
            for i = 1:k
                resp(i) = mvnpdf(inputVector, ...
                    obj.mixureMeans(i,:),obj.mixureVariance(:,:,i));
            end
            resp = resp.*obj.pi;
            respTotal = sum(resp);
            resp = resp./respTotal;
        end

        function [ probability ] = getLikelihood(obj,inputVector,naiveBayes)
            %Find probability of input Vector
            %Usage: obj.getProbability(inputVector)
            %parameters:
            % inputVector - input Vector whose Probability to find
            % naiveBayes(optional) - boolean [default value ='false']
            %
            if (nargin == 2)
                naiveBayes = false;
            end

            k = obj.K;
            probability = zeros(1,k);

            if (naiveBayes)
                for i = 1:k
                    probability(i) = mvnpdf(inputVector,obj.mixureMeans(i,:), ...
                        diag(diag(obj.mixureVariance(:,:,i))));
                end
            else
                for i = 1:k
                    probability(i) = mvnpdf(inputVector,obj.mixureMeans(i,:), ...
                        obj.mixureVariance(:,:,i));
                end
            end
            probability = probability.* obj.pi;
            probability = log(sum(probability));
        end
    end
end
```

K-Means Clustering

File: KmeansCluster.m

```
classdef KmeansCluster < handle
    %KMeans Clusting
    % partitions the points in the nXm data matrix into K clusters
    % Usage example:
    %   KM = KmeansCluster(3); %3 = no of mixtures
    %   KM.train(c);           %c = input Vectors
    %   KM.Kmeans        %to get Means
    %   KM.Kvariance     %to get Variance
    %   KM.getLabel(ip) % get label of `ip` vector
    %

    properties (SetAccess = public)
        K; % No: of Clusters
        Kmeans; % Means of Clusters
        Kvariance; % Covariances of Clusters
    end

    methods
        function obj = KmeansCluster(k)
```

```
        %constructor:set K (No: of Clusters)
        %Usage: obj = KmeansCluster(k);
        %Parameter:
        %   k - # of Clusters
        %
        obj.K = k;
    end %constructor

    function train(obj,inputVectors,stepCount)
        %Train k-means clusting
        %Usage:
        %   obj.train(inputVectors);
        %   obj.train(inputVectors,stepCount);
        %Parameters:
        % inputVectors - input Vectors (MxN)
        % stepCount (optional) - Maxium no of iteration
        %       default value = 15
        if(nargin==2)
            stepCount = 15;
        end
        [M,N] = size(inputVectors);
        classLabel = randi(obj.K,1,M);
        obj.Kmeans=zeros(obj.K,N);
        obj.Kvariance=zeros(N,N,obj.K);
        %init Random mean
        for trail = 1:5
            for j = 1:obj.K
                obj.Kmeans(j,:) = mean(inputVectors(classLabel==j,:));
            end
            i=0;
            threshold = 0.001;
            meanError = threshold +1;
            while (i < 2 || (meanError > threshold && i < stepCount ))
                i=i+1;
                %E step

                if(i<12)
                    %euclidean distance
                    for j = 1:M
                        classLabel(j) = obj.getLabel(inputVectors(j,:),true);
                    end
                else
                    s = warning('off','all');
                    %Mahalanobis distance
                    for j = 1:M
                        classLabel(j) = obj.getLabel(inputVectors(j,:),false);
                    end
                    warning(s);
                end
                %M step
                KmeanOld = obj.Kmeans;
                for j = 1:obj.K
                    classVectors = inputVectors(classLabel==j,:);
                    obj.Kmeans(j,:) = mean(classVectors);
                    obj.Kvariance(:,:,j) = cov(classVectors);
                end
                meanError = norm(obj.Kmeans-KmeanOld);
            end
            if sum(isnan(obj.Kmeans))== 0
                break;
            end
        end
        fprintf('After %d iteration\n',i);
    end
```

```
        function [ label ] = getLabel(obj,inputVector,euclidean)
            %Find label of input Vector
            %Usage:
            %   obj.getLabel(inputVector);
            %   obj.getLabel(inputVector,euclidean)
            %parameters:
            % inputVector - input Vector to be labeled
            % euclidean (optional) -
            %       if `true` distance measure is euclidean
            %       else distance measure is Mahalanobis distance
            %       default value :true

            if(nargin <3)
                euclidean = true;
            end
            k = obj.K;
            diffVector = bsxfun(@minus,obj.Kmeans,inputVector);
            if euclidean
                distances = sqrt(sum(diffVector.*diffVector,2));
            else
                distances = zeros(1,k);
                for i = 1:k
                    distances(i) =
sqrt(diffVector(i,:)/obj.Kvariance(:,:,i)*diffVector(i,:)');
                end
            end
            [~,label] = min(distances);
        end
    end
end
```

HMM

File: hmmData.m

```
% HMM code for Handwritten dataset
function [classNo, classList, data] = hmmData()

if exist('../data/hmm/handwritten','dir')
    dataPath = '../data/hmm/handwritten';
else
    dataPath = uigetdir;
end

classList = dir(dataPath);
classList = {classList(~strncmpi({classList.name},'.',1)).name};
classNo = size(classList,2);
data{classNo} = {};

for i = 1:classNo
    fId = fopen(char(strcat(dataPath,'/',classList(i))));
    line1 = fgets(fId);
    j = 1;
    while ischar(line1)
        line2 = fgets(fId);
        line3 = fgets(fId);
        [~,line3] = strtok(line3);
        line3 = reshape(str2num(line3),2,[])';

        % Feature : Slope (but inf values, so not taking)
        % ftr = diff(reshape(str2num(line3),2,[])');
        % ftr = (ftr(:,2)./ftr(:,1))';

        % Feature1 : Normalized X-Y Coordinates b/w [0,1]
```

```
        line3(:,1) = bsxfun(@minus,line3(:,1),min(line3(:,1)))./(max(line3(:,1))-
min(line3(:,1)));
        line3(:,2) = bsxfun(@minus,line3(:,2),min(line3(:,2)))./(max(line3(:,2))-
min(line3(:,2)));
        ftr(:,1:2) = line3;

        % Feature2 : Normalized First Derivatives
        tmpArray = padarray(line3,[2 0]);
        tmp = 2*circshift(tmpArray,[-2,0]);
        tmp = tmp + circshift(tmpArray,[-1,0]);
        tmp = tmp - circshift(tmpArray,[1,0]);
        tmp = tmp - 2*circshift(tmpArray,[2,0]);
        tmp = tmp/10;
        ftr(:,3:4) = tmp(3:end-2,:);

        % Feature3 : Normalized Second Derivatives
        tmpArray = padarray(ftr(:,3:4),[2 0]);
        tmp = 2*circshift(tmpArray,[-2,0]);
        tmp = tmp + circshift(tmpArray,[-1,0]);
        tmp = tmp - circshift(tmpArray,[1,0]);
        tmp = tmp - 2*circshift(tmpArray,[2,0]);
        tmp = tmp/10;
        ftr(:,5:6) = tmp(3:end-2,:);

        % Feature4 : Curvature
        % x'.y'' - x''.y'/((x'^2+y'^2)^(3/2))
        %numerator = ftr(:,3).*ftr(:,6)-ftr(:,5).*ftr(:,4);
        %denominator = (sum(ftr(:,3:4).^2,2)).^(3/2);
        %ftr(:,7) = numerator./denominator;

        line1 = fgets(fId);
        data{i}{j} = ftr;
        j = j + 1;
        clear ftr;
    end

    fclose(fId);
end
clear ans dataPath denominator fId i j line1 line2 line3 num numerator tmp tmpArray
end
```

File: HMM.m

```
% Load Filelist of the Handwritten Dataset
[classNo, classList, data] = hmmData();
% classSize() : Size of each Class
classSize = zeros(1,classNo);
% K : Percent of Test Data
K = input('Percent of Test Data: ')/100;
% trainInd{} : Index of Files in Training for each Class
trainInd{classNo} = {};
% testInd{} : Index of Files in Testing for each Class
testInd{classNo} = {};
% testData{} : Test Data of each Class
testData{classNo} = {};
% trainData : Training Data
trainData{classNo} = {};
kmData = [];

for i = 1:classNo
    classSize(i) = size(data{i},2);
    % Divide Image Dataset to Train and Test
    Index = randperm(classSize(i));
    % numTest : Number of Test Files per Class
```

```
    numTest = round(classSize(i)*K);
    numTrain = classSize(i)-numTest;
    testInd{i} = sort(Index(1:numTest));
    trainInd{i} = sort(Index(numTest+1:classSize(i)));
    for j = 1:numTest
        testData{i}(j) = data{i}(testInd{i}(j));
    end
    tempData = [];
    for j = 1:numTrain
        trainData{i}(j) = data{i}(trainInd{i}(j));
        tempData = [tempData;cell2mat(data{i}(trainInd{i}(j)))];
    end
    kmData = [kmData;tempData];
    clear j numTest numTrain Index tempData;
end

K = input('Enter No. of Clusters: ');
km = KmeansCluster(K);
km.train(kmData);

for s = 1:classNo
    trainlabel = {};
    for i = 1:size(trainData{s},2)
        labeldata = trainData{s}{i};
        [n,~] =size(labeldata);
        label = zeros(1,n);
        for j = 1:n
            label(j) = km.getLabel(labeldata(j,:))-1;
        end
        trainlabel{i} = label;
    end
    trainLabel{s} = trainlabel;
end

for s = 1:classNo
    file = fopen(['hmm/train' num2str(s) '.txt'],'w');
    for i = 1:size(trainData{s},2)
        fprintf(file,'%d ',trainLabel{s}{i});
        fprintf(file,'\n');
    end
    fclose(file);
end

for s = 1:classNo
    testlabel = {};
    for i = 1:size(testData{s},2)
        labeldata = testData{s}{i};
        [n,~] =size(labeldata);
        label = zeros(1,n);
        for j = 1:n
            label(j) = km.getLabel(labeldata(j,:))-1;
        end
        testlabel{i} = label;
    end
    testLabel{s} = testlabel;
end

for s = 1:classNo
    file = fopen(['hmm/test' num2str(s) '.txt'],'w');
    for i = 1:size(testData{s},2)
        fprintf(file,'%d ',testLabel{s}{i});
        fprintf(file,'\n');
    end
    fclose(file);
end
```

```
x = [];
for s =1:classNo
    x(s) = input(['Enter No. of States(Class ' num2str(s) '): ']);
end
y = 1/K;

for s = 1:classNo
    file = fopen(['hmm/test' num2str(s) '.hmm.seq'],'w');
    fprintf(file,'states: %d\nsymbols: %d\n\n',x(s),K);
    for j = 1:(x(s)-1)
        fprintf(file,'0.5');
        for i = 1:K
            fprintf(file,'\t%d',y);
        end
        fprintf(file,'\n');
        fprintf(file,'0.5');
        for i = 1:K
            fprintf(file,'\t%d',y);
        end
        fprintf(file,'\n');
        fprintf(file,'\n');
    end
    fprintf(file,'1.0');
    for i = 1:K
        fprintf(file,'\t%d',y);
    end
    fprintf(file,'\n');
    fprintf(file,'0.0');
    for i = 1:K
        fprintf(file,'\t%d',y);
    end
    fclose(file);
end

for s = 1:classNo
    system(['./train_hmm hmm/train' num2str(s) '.txt hmm/test' num2str(s) '.hmm.seq
.01']);
end

ConfMat = zeros(classNo);
Accu = zeros(1,classNo);
for i = 1:classNo
    picout = [];
    for j = 1:classNo
        system(['./test_hmm hmm/test' num2str(i) '.txt hmm/train' num2str(j)
'.txt.hmm']);
        out = load('alphaout');
        picout = [picout ; out];
    end
    [~,n] = size(picout);
    for j = 1:n
        [~,class] = max(picout(:,j));
        ConfMat(i,class) = ConfMat(i,class) + 1;
    end
end
system('rm alphaout');

for s = 1:classNo
    Accu(s) = ConfMat(s,s) / sum(ConfMat(s,:));
end
AvgAccu = sum(diag(ConfMat))/sum(sum(ConfMat));

clc;
```

```
disp(['No. of Clusters: ' num2str(K)]);
disp(['No. of Symbols: ' num2str(K)]);

for s =1:classNo
    disp(['No. of States(Class ' num2str(s) '): ' num2str(x(s))]);
end

disp('Confusion Matrix');
disp(ConfMat);
disp('Classification Accuracy');
disp(Accu);
disp('Average Accuracy');
disp(AvgAccu);

clear all;
```

DTW

File: DTW.cpp

```
/*
 * DTW.cpp
 *
 *  Created on: Dec 2, 2013
 *      Author: abil
 */

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <cfloat>


#define REF_LEN 3
#define MAX 20
#define DEBUG
using namespace std;

struct TestInfo
{
    int start[MAX];
    int end[MAX];
    int trueClass[MAX];
    int pClass[MAX];
    int n;
};

double DTW(vector<double> x,vector<double> y)
{
  int i,j;
  double **Dist;
  double **DTWtable;
  double distance;

  int xsize = x.size();
  int ysize = y.size();

  //allocate memory to dist
```

```
  Dist = new double *[xsize];
  for (i=0; i < xsize; i++)
    Dist[i] = new double[ysize];

  //#pragma omp parallel
  for(i=0; i<xsize;i++) {
    for(j=0;j<ysize;j++) {
      Dist[i][j] = abs(x[i]-y[j]);
    }
  }

  //allocate memory to DTWtable
  DTWtable = new double *[xsize];
  for (i=0; i < xsize; i++)
    DTWtable[i] = new double[ysize];

  DTWtable[0][0] = Dist[0][0];
  for(i=1;i<xsize;i++)
    DTWtable[i][0] = Dist[i][0] + DTWtable[i-1][0];

  for(j=1;j<ysize;j++)
    DTWtable[0][j] = Dist[0][j] + DTWtable[0][j-1];

  for(i=1;i<xsize;i++) {
   // for(j=max(1,i-w); j<min(ysize,i+w);j++){
    for(j=1; j<ysize;j++){
        DTWtable[i][j] = Dist[i][j] +
          min(DTWtable[i][j-1],min(DTWtable[i-1][j-1],DTWtable[i-1][j]));
      //    | (i,j-1)   | (i,j)  |
      //    | (i-1,j-1)|  ---   |

      /*

      double minDist = DBL_MAX;
      for(int k=1; k<ysize;k++){
        minDist = min(Dist[i][k] + DTWtable[i-1][k],minDist);
      }
      DTWtable[i][j] = minDist;

      */
    }
  }

  distance = Dist[xsize-1][ysize-1];

  //Deallocating Memory
  for (i=0; i<xsize; ++i)
    delete [] Dist[i];
  delete [] Dist;
  for (i=0; i<xsize; ++i)
    delete [] DTWtable[i];
  delete [] DTWtable;

  return distance;
}


bool loadData(const string &filename,vector<double> &doubleVec)
{
  fstream fs;
  string line;
  //double fNum;
  fs.open(filename.c_str(),ios::in);
```

```
  if (!fs.is_open())
  {
    cerr<<"Error:Opening "<<filename<<" failed"<<endl;
    return false;
  }

  while(fs>>line){
    if(line.size() == 0)
      continue;
    if(line=="-Inf")
      doubleVec.push_back(-1*DBL_MAX);
    else
      doubleVec.push_back(atof(line.c_str()));
  }
  fs.close();
  return true;
}


bool readGroundTruth(TestInfo gt[],const string filename[],int nTest,
    string classNames[],int nClass)
{

  fstream fs;
  string line;
  double start;
  double end;
  string className;

  int nLine;
  for (int i = 0; i < nTest; ++i) {
    nLine = 0;
    fs.open(filename[i].c_str(),ios::in);
    if (!fs.is_open())
    {
      cerr<<"Error:Opening "<<filename[i]<<" failed"<<endl;
      return false;
    }

    while(getline(fs,line))
    {
      if(line.length()==0)
        continue;
      stringstream  linestream(line);
      linestream>>start;
      linestream>>end;
      linestream>>className;

      gt[i].start[nLine]=floor(start*100);
      gt[i].end[nLine]=floor(end*100);
      gt[i].trueClass[nLine] = -1;
      for(int j =0; j<nClass;j++) {
        if(className == classNames[j]){
          gt[i].trueClass[nLine]=j;
          break;
        }
      }
      if(gt[i].trueClass[nLine] == -1)
        cerr<<"ERROR:["<<filename[i]
            <<"] Unknown Class:"<<className<<endl;
      ++nLine;
    }
    fs.close();
    gt[i].n = nLine;
```

```
  }
  return true;
}


int getClass(vector<double> testVector,vector<double> refVectors[],int nClass,int
start,int end)
{

  vector<double>::const_iterator first = testVector.begin() + start;
  vector<double>::const_iterator last = testVector.begin() + end;
  vector<double> testVectorSlice(first, last);

  double classDist[nClass];

#pragma omp parallel for shared(classDist)
  for (int j = 0; j < nClass; ++j) {
    classDist[j]=DTW(refVectors[j],testVectorSlice);
    //cout<<"("<<j<<")"<<classDist[j]<<"\t";
  }

  double minDist=DBL_MAX;
  int minIndex = -1;
  for(int i=0;i<nClass;i++){
    if(classDist[i]<minDist){
      minDist=classDist[i];
      minIndex=i;
    }
  }
  //cout<<minIndex;
  //cout<<endl;
  return minIndex;
}




int main()
{

  int nClass = 3;

  int confMat [nClass][nClass];
  vector<double> ref[nClass];

  int nTest = 16;
  vector<double> testVectors[nTest];


  string refFilename[] = {
      "Music Data/Bhairavi_Ref/Bhairavi_44100.vp1.wav.cent.spline1",
      "Music Data/Bhairavi_Ref/Bhairavi_44100.vp2.wav.cent.spline1",
      "Music Data/Bhairavi_Ref/Bhairavi_44100.vp3.wav.cent.spline1"
  };

  string classNames[] = {
      "VI_VP1",
      "VI_VP2",
      "VI_VP3"
  };

  string labelFilename[] = {
```

```
        "Music Data/Bhairavi_Test_Ground_Truth/VS_Bhairavi1_289VSKozhikode.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/TVS_Bhairavi1_Sigamani3dvd8062.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/TMK_Bhairavi3_US2002Toronto.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/TMK_Bhairavi2_NJ05.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/Sanjay_Bhairavi1_FASfeb2001.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/RK_Bhairavi1_1964jamshedpur.wav.lab",
        "Music
Data/Bhairavi_Test_Ground_Truth/Nedanuri_Bhairavi1_concert40tmk1tb.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/Musiri_Bhairavi1_musirinew.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/MSS_Bhairavi2_MSSConcertIX.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/MSS_Bhairavi1_academy1970.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/MDR_Bhairavi1_065MDRhari101.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/KVN_Bhairavi2_NF117.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/KVN_Bhairavi1_NF598KVN.wav.lab",
        "Music
Data/Bhairavi_Test_Ground_Truth/GNB_Bhairavi1_0641965kallidaikurichi.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/ARI_Bhairavi1_aritnkpmi1ssrk.wav.lab",
        "Music Data/Bhairavi_Test_Ground_Truth/ALB_Bhairavi1_70albmgpnganesan.wav.lab"
  };

  string testFilename[]={
        "Music Data/Bhairavi_Test/VS_Bhairavi1_289VSKozhikode.wav.cent.spline1",
        "Music Data/Bhairavi_Test/TVS_Bhairavi1_Sigamani3dvd8062.wav.cent.spline1",
        "Music Data/Bhairavi_Test/TMK_Bhairavi3_US2002Toronto.wav.cent.spline1",
        "Music Data/Bhairavi_Test/TMK_Bhairavi2_NJ05.wav.cent.spline1",
        "Music Data/Bhairavi_Test/Sanjay_Bhairavi1_FASfeb2001.wav.cent.spline1",
        "Music Data/Bhairavi_Test/RK_Bhairavi1_1964jamshedpur.wav.cent.spline1",
        "Music Data/Bhairavi_Test/Nedanuri_Bhairavi1_concert40tmk1tb.wav.cent.spline1",
        "Music Data/Bhairavi_Test/Musiri_Bhairavi1_musirinew.wav.cent.spline1",
        "Music Data/Bhairavi_Test/MSS_Bhairavi2_MSSConcertIX.wav.cent.spline1",
        "Music Data/Bhairavi_Test/MSS_Bhairavi1_academy1970.wav.cent.spline1",
        "Music Data/Bhairavi_Test/MDR_Bhairavi1_065MDRhari101.wav.cent.spline1",
        "Music Data/Bhairavi_Test/KVN_Bhairavi2_NF117.wav.cent.spline1",
        "Music Data/Bhairavi_Test/KVN_Bhairavi1_NF598KVN.wav.cent.spline1",
        "Music
Data/Bhairavi_Test/GNB_Bhairavi1_0641965kallidaikurichi.wav.cent.spline1",
        "Music Data/Bhairavi_Test/ARI_Bhairavi1_aritnkpmi1ssrk.wav.cent.spline1",
        "Music Data/Bhairavi_Test/ALB_Bhairavi1_70albmgpnganesan.wav.cent.spline1"
  };

  //loading ref file
  cout<<"Loading Query Vectors"<<endl;
  for(int i=0;i<nClass;i++){
    loadData(refFilename[i],ref[i]);
#ifdef DEBUG
    cout<<"loaded "<<refFilename[i]
        <<"("<<ref[i].size()
        <<")"<<endl;
#endif
  }
  cout<<"Query Vectors:Loaded "<<endl<<endl;


  cout<<"Loading Test Vectors"<<endl;
  for(int i=0;i<nTest;i++){
    loadData(testFilename[i],testVectors[i]);
#ifdef DEBUG
    cout<<"loaded "<<testFilename[i]
        <<"("<<testVectors[i].size()
        <<")"<<endl;
#endif
  }
  cout<<"Test Vectors:Loaded "<<endl<<endl;

  TestInfo gtruth[nTest];
```

```
  readGroundTruth(gtruth,labelFilename,
      nTest,classNames,nClass);


  for(int i=0;i<nClass;i++){
    for (int j = 0; j < nClass; ++j) {
      confMat[i][j] = 0;
    }
  }

  for(int i=0;i<nClass;i++){
      for (int j = 0; j < nClass; ++j) {
        cout<<confMat[i][j]<<"\t";
      }
      cout<<endl;
    }


  for(int i=0;i<nTest;i++){
    for (int j = 0; j < gtruth[i].n; ++j) {
      gtruth[i].pClass[j] = getClass(testVectors[i],
          ref,nClass,gtruth[i].start[j],gtruth[i].end[j]);

#ifdef DEBUG
      cout<<i<<" =>"
          <<gtruth[i].start[j]<<"  "
          <<gtruth[i].end[j]<<"  "
          <<gtruth[i].end[j]-gtruth[i].start[j]<<" "
          <<gtruth[i].trueClass[j];
      cout<<"==>"<<gtruth[i].pClass[j]<<endl;
#endif
       confMat[gtruth[i].trueClass[j]][gtruth[i].pClass[j]]++;
    }
  }

  int correct=0,total=0;
  cout<<"Confusion Matrix:"<<endl;
  for(int i=0;i<nClass;i++){
    for (int j = 0; j < nClass; ++j) {
      cout<<confMat[i][j]<<"\t";
      if(i==j)
        correct+=confMat[i][j];
      total+=confMat[i][j];
    }
    cout<<endl;
  }

  cout<<endl;
  cout<<"Accuracy:"<<(double(correct)/total)<<endl;

}
```