**EECS**

**University of Tennessee**

**Pattern Recognition – ECE 571**

**Project 1 – Two Category Classification Using Bayesian Decision Rule**

**Submitted by:**

Mahendra Duwal Shrestha

[mduwalsh@vols.utk.edu](mailto:mduwalsh@vols.utk.edu)

**Abstract**

Bayesian decision rule is one of the most used pattern classifier tools which uses statistical approach to classify sample testing data to one of its categories. Bayesian decision rule uses probabilistic method to calculate test data’s probability of falling into a category.

The objective of this project is to design a decision rule for the synthetic data set with two categories. This project carried out supervised methods using Bayesian decision rule to classify provided data set assuming probability density is Gaussian distribution.

Estimates mean and covariance were calculated using given training samples and used for classification purpose of testing samples. In this project, maximum posterior probability (MPP) with all three cases and likelihood ratio methods were implemented to classify samples. Accuracy was computed for each method and performance of decision rule of each method was evaluated based on it. Also two modal Gaussian was used to model the data set given and performance was compared with that of one modal Gaussian.

**Introduction**

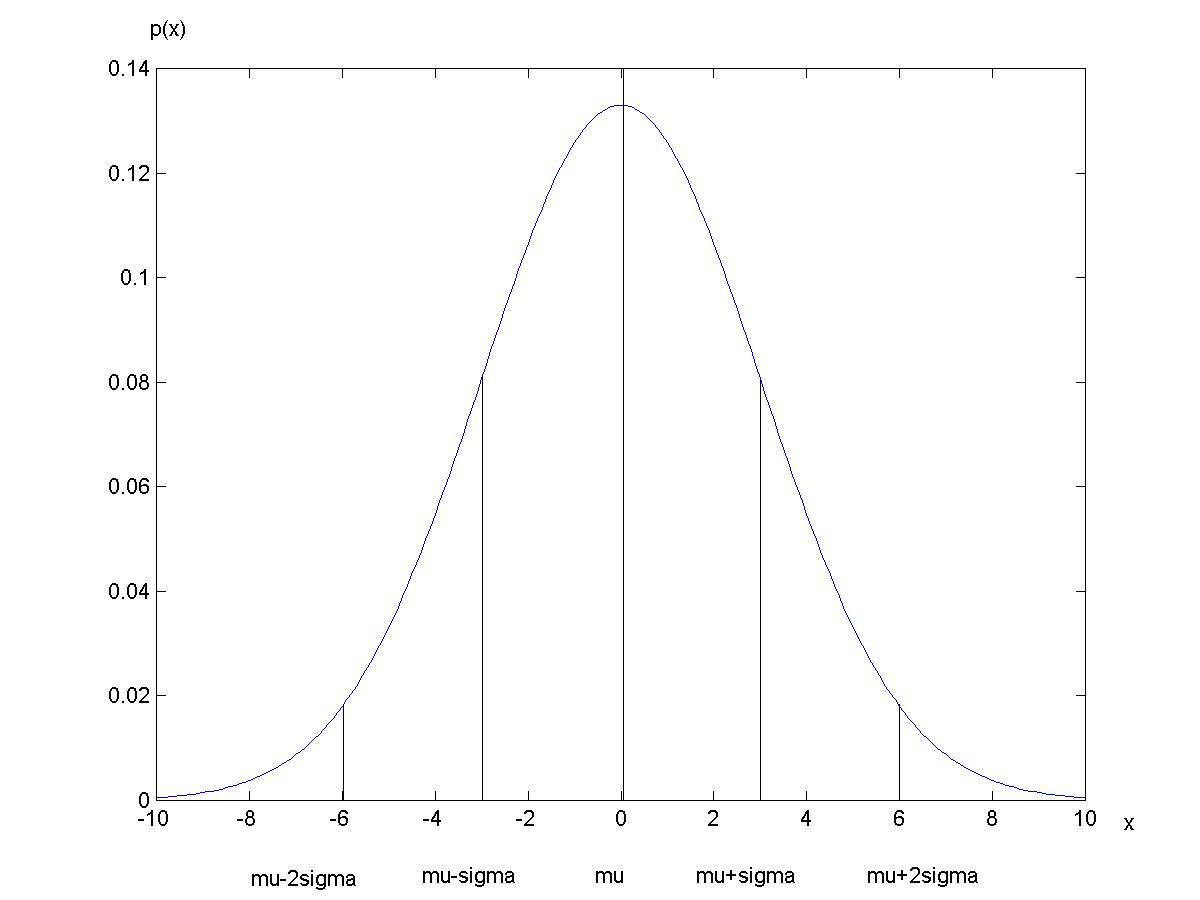
Pattern recognition is a science of making conclusion based on interpreting data, using tools like statistics, probability, signal processing, decision algorithms etc. It has great applications in science both physical and medical and engineering. Pattern classification is an integral part of pattern recognition which deals with work of identifying feature extracted input data as one of the categories associated to data set. Classification is done based on some established decision rule. Bayesian decision rule is one of the most commonly used decision rules. This decision rule uses Bayesian probability statistics to estimate expected value from input going through action with help of prior probability distribution.

This project used a few methods of classification using Bayesian decision rule to design a decision rule for the synthetic data set with two categories and two dimensions (features) assuming probability distribution of data is Gaussian. Methods used in the project are maximum posterior probability (MPP) and likelihood ratio. Also two modal Gaussian was used to model the data set and compared performance with that of using single modal Gaussian.

The training data set provided was used to calculate estimates mean and covariance. The calculated mean and covariance were used to establish decision rule for each of three cases of MPP and for likelihood ratio methods. Testing samples were tested using each of the decision rules and each of their performances was evaluated. With comparison, third case decision rule with arbitrary covariance was found to have higher accuracy than other two with MPP. Likelihood ratio supposing zero-one loss had same level of performance as that of third case decision rule of MPP.

**Technical Approach**

The training and testing samples data set with two categories and two dimensions were provided as synth.tr and synth.te files. We needed to develop program that uses Bayesian decision rule to automatically classify and calculate accuracy for each of decision rules method. We were provided C++ code to calculate accuracy using MPP with all three cases. So we all need to develop was program to classify using likelihood ratio. As required the code for likelihood ratio method was developed using C++ language. The pattern of code was changed for likelihood ratio so that values of estimates mean and covariance can be easily accessed anytime when desired. A separate class was defined to store values of estimate where after calculation it will be stored as object and can be accessed anytime easily. It was especially helpful to view them when doing calculation other than programmatically where estimates values are individually required. Estimates mean and covariance were calculated from the training set using Maximum Likelihood. The calculated estimates were used to derive decision rules and the testing set was tested on those rules. For more detail and graphical illustration of decision rules, MATLAB simulation software was used. It was used to plot test sample data in two dimensions. Also decision boundary for each of the decision rules were plotted with the plot with sample data in the same figure.

**Gaussian Density Distribution:**

Multivariate Gaussian density is given by:



When d=1,

**Maximum Posterior Probability (MPP):**

Bayesian decision formula for MPP is given by:



Where P(ωj|x) is posterior probability; p(x| ωj) is conditional probability density function or likelihood; p(x) is normalization constant and is given by:



P(ωj) is prior probability.

Discriminant function for multivariate Gaussian density can be represented as:



**Case I:**

If features of sample data are statistically independent and have same variance, then we’ll have



where

The Euclidean distance is also called minimum distance because it is the distance of x from each of the c mean vectors. And when  is same for all classes the discriminant function is measuring minimum distance.



**Case II:**

If covariance matrices of all classes are same and not equal to scalar multiplication of identity matrix, then and discriminant function will be:



**Case III:**

If covariance matrices of all classes are arbitrary, then discrimanant function will be:



**Likelihood Ratio:**

Likelihood ratio for two classes is given by:



For zero-one loss, so the inequality becomes.

If it is true then the sample belongs to class 1 or else it belongs to class 2.

Taking natural log to both sides and rearranging, it can be expressed in the form of

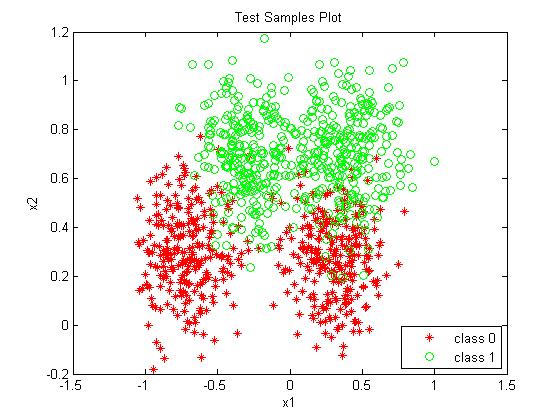


 where both terms can be expressed as in the case III of MPP.

**Experiments and Results:**

**Test samples plot**

The provided test samples were plotted graphically using MATLAB.



**Estimates calculation:**

Estimates were calculated using training data set and C++ code which were found to be:

; ;

;

;

;

;

**Decision Boundary**

**MPP Case I**

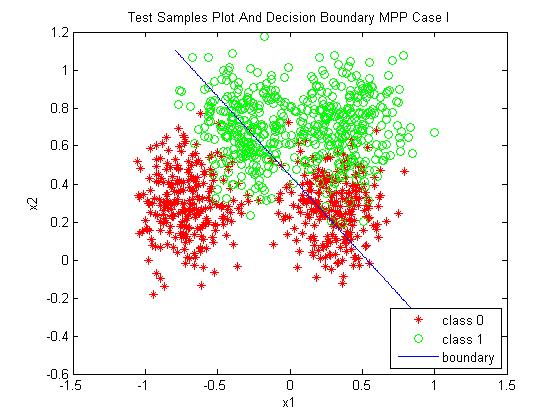
Discriminant function of case I is given by



For decision boundary is given by which yielded,

which is a linear equation.

And the boundary was plotted using MATLAB.



**MPP Case II**

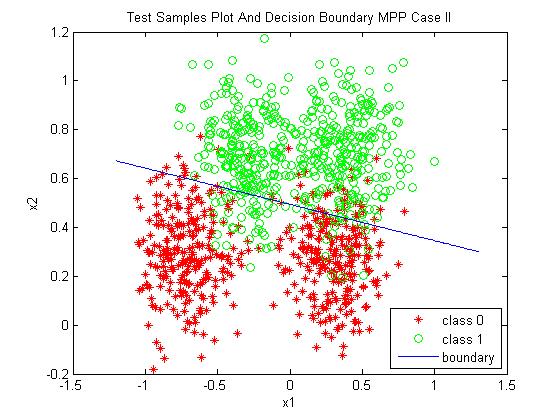
Discriminant function of case II is given by



For decision boundary is given by which yielded,

which is also a linear equation.

And the boundary was plotted using MATLAB.



Though the boundary is straight line as in case I, its slope and intercepts are different and visually boundary in case II seems to be better performing decision.

**MPP Case III**

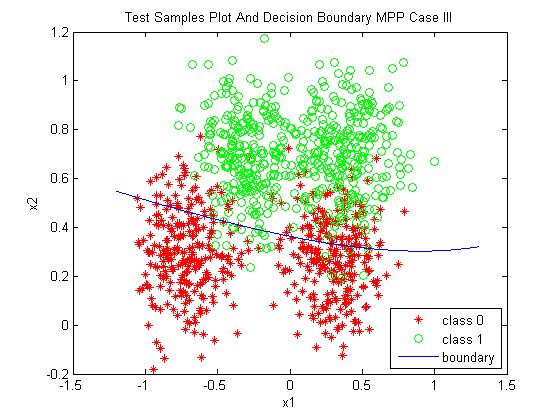
Discriminant function of case III is given by



For decision boundary is given by which yielded,

which is quadratic equation.

And the boundary was plotted using MATLAB.



In case III, decision boundary is not straight line but quadratic line unlike previous cases. Visually the decision boundary is better in performance than others.

Decision boundary obtained from likelihood ration with zero-one loss was same as in MPP case III. So the plot is not included.

**Performance evaluation**

Assuming equal prior probability for both classes, the error rates were measured for each decision rules on the same testing data set. The error rate for MPP case I was measured to be 0.287. For MPP case II, it was 0.108. And for MPP case III, it was 0.102. Error rate from likelihood ratio was also found to be 0.102. From the results, it can be figured that error rate was least in case of MPP case III and likelihood ratio and highest in MPP case I.

Different prior probabilities were assigned to each class and experiment was carried out. Class 0 was assigned 0.3 and class 1 was assigned 0.7. Then the new error rates were 0.266, 0.128 and 0.126 for MPP case I, case II and case III respectively. Error rate for likelihood ratio was again found to be same as MPP case III.

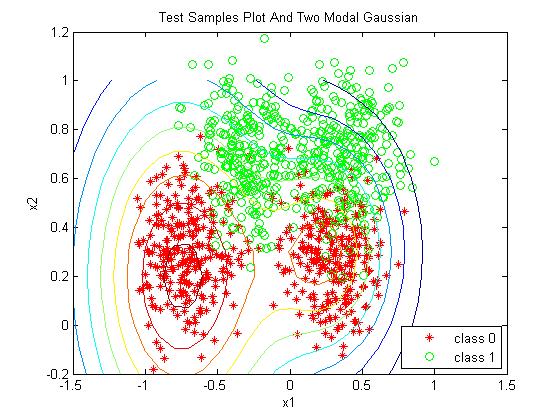
In both cases, MPP case III and likelihood ratio seemed to perform better than any other.

**Two Modal Gaussian**

Instead of one modal Gaussian, two modal Gaussian was used to model the data set with parameters

;;;; ;;

The contour plot with this model was plotted with MATLAB.



The error rate of this model was calculated using likelihood ratio (or MPP case III) which was measured to be 0.484 which seems to be worse than with of one modal Gaussian.

**Discussion**

In this project, estimates for trained samples were calculated using maximum likelihood. Decision rules using Bayesian decision theory were designed using three cases of discriminant functions and likelihood ratio. Decision boundaries of each designed decision rules were observed and studied. Performance for each decision rule was measured and evaluated comparing with each other. Performance of decision rule using MPP case III discriminant function and likelihood ratio was found to be same and better than performance of other decision rules. With this project, we also learnt modeling of decision rule in C++ code and use of MATLAB in simulation of data and functions. There are still other decision rules which may yield higher performance and better decision, these have to be modeled in our program to evaluate performance and find decision rule with better performance.

**References**

* Lectures notes from class ECE 471/571 Pattern Recognition (Prof. Hairong Qi)
* [www.wikipedia.com](http://www.wikipedia.com)

**Appendix**

C++ codes (included only developed or changed files and exclude given code)

1. **/include/Estimate.h**

// Estimate class contains attributes means and covs which are estimates of sample data matrices mean and covariance

// and nf and ncl as no of features and no of classes respectively

using namespace std;

class Estimate

{

private:

Matrix \*means; // pointer to mean matrix to hold mean matrix for all classes

Matrix \*covs; // pointer to covariance matrix to hold covariance matrix for all classes

int nf;

int ncl;

public:

Estimate(); // default constructor

Estimate(int,int); // parameterized constructor

~Estimate(); // destructor

Matrix \* getMean(); // member function to access means attribute

void setMean(Matrix \*); // member function to set means attribute

Matrix \* getCovariance(); // member function to access covs attribute

void setCovariance(Matrix \*); // member function to set covs attibute

// this was developed as alternate method to calculate estimates

void calculateEstimates(const Matrix &train, int c); // function to calculate means and covariances for given sets of trained data

};

1. **/include/Pr.h**

/\*

\* pr.h - header file of the pattern recognition library

\*

\* Author: Hairong Qi, ECE, University of Tennessee

\*

\* Date: 01/25/04

\*

\* Please send all your comments to hqi@utk.edu

\*

\* Modified:

\* - 09/24/13: add "const" to the filename parameters to remove warning

\* msg in new compilers (Steven Clukey)

\* - 04/26/05: reorganized for the Spring 2005 classs

\*/

#ifndef \_PR\_H\_

#define \_PR\_H\_

#include "Matrix.h"

#include "Estimate.h"

/////////////////////////

// file I/O

Matrix readData(const char \*, // the file name

int); // the number of columns of the matrix

Matrix readData(const char \*, // the file name

int, // the number of columns

int); // the number of rows (or samples)

Matrix readData(const char \*); // read data file to a matrix with 1 row

void writeData(Matrix &, const char \*); // write data to a file

Matrix readImage(const char \*, // read the image from a file

int \*, // the number of rows (or samples)

int \*); // the number of columns

void writeImage(const char \*, // write the image to a file

Matrix &, // the matrix to write

int, // the number of rows

int); // the number of columns

////////////////////////

// distance calculation

double euc(const Matrix &, // Euclidean distance between two vectors

const Matrix &);

double mah(const Matrix &, // the Mahalanobis distance, input col vec

const Matrix &C, // the covariance matrix

const Matrix &mu); // the mean (a col vector)

// Estimates calculation

Estimate estimateCalculation(const Matrix &train, int c);

////////////////////////

// classifiers

// maximum a-posteriori probability (mpp)

int mpp(const Matrix &train, // the training set of dimension mx(n+1)

// where the last col is the class label

// that starts at 0

const Matrix &test, // one test sample (a col vec), nx1

const int, // number of different classes

const int, // caseI,II,III of the discriminant func

const Matrix &Pw); // the prior prob, a col vec

// likelyhood ratio (lr)

int lr(Matrix \*mu, Matrix \*cov, Matrix &teData, int classes, Matrix &Pw);

#endif

1. **/lib/Estimate.cpp**

// definition of member functions of class Estimate

#include <iostream>

#include <cstdlib>

#include <cstdio>

#include <cmath>

#include <iomanip>

#include "Matrix.h"

#include "Estimate.h"

using namespace std;

// empty constructor

Estimate::Estimate()

{

means = (Matrix \*) new Matrix [0];

covs = (Matrix \*) new Matrix [0];

nf=0;

ncl=0;

}

// parameterized constructor

Estimate::Estimate(int nft,int c)

{

nf=nft;

ncl=c;

means=(Matrix \*) new Matrix[ncl];

for (int i=0; i<ncl; i++)

means[i].createMatrix(nf, 1);

covs=(Matrix \*) new Matrix[ncl];

for (int i=0; i<ncl; i++)

covs[i].createMatrix(nf, nf);

}

// destructor

Estimate::~Estimate()

{

if(means){delete [] means;}

if(covs){delete [] covs;}

}

// getter for means which is array or pointer to mean matrix

Matrix \* Estimate::getMean()

{

return means;

}

// setter for means

void Estimate::setMean(Matrix \*mean)

{

for(int i=0;i<ncl;i++)

means[i]=mean[i];

}

// getter for covs

Matrix \* Estimate::getCovariance()

{

return covs;

}

// setter for covs

void Estimate::setCovariance(Matrix \*cov)

{

for(int i=0;i<ncl;i++)

covs[i]=cov[i];

}

// alternate way to calculate estimates

void Estimate::calculateEstimates(const Matrix &train, int c)

{

double varavg;

Matrix tmp;

int nctr;

int nf,i, j;

// get the size of input raw data

nctr = train.getCol();

nf = nctr-1;

// the mean is an array of nfx1 matrix of size c and the cov is an array of nf x nf matrix of size c

means = (Matrix \*) new Matrix [c];

for (i=0; i<c; i++)

means[i].createMatrix(nf, 1);

covs = (Matrix \*) new Matrix [c];

for (i=0; i<c; i++)

covs[i].createMatrix(nf, nf);

// calculate the mean and covariance

for (i=0; i<c; i++) {

tmp = getType(train, i);

covs[i] = cov(tmp, nf);

means[i] = mean(tmp, nf);

}

}

1. **/lib/EstimateCalculation.cpp**

// function to calculate estimates means and covariances

#include <iostream>

#include <cstdlib>

#include <cmath>

#include "Matrix.h"

#include "Pr.h"

using namespace std;

Estimate estimateCalculation(const Matrix &train, int c)

{

double varavg;

Matrix \*means, \*covs, tmp;

int nctr;

int nf,i, j;

// get the size of input raw data

nctr = train.getCol();

nf = nctr-1;

// the mean is an array of nfx1 matrix of size c and the cov is an array of nf x nf matrix of size c

means = (Matrix \*) new Matrix [c];

for (i=0; i<c; i++)

means[i].createMatrix(nf, 1);

covs = (Matrix \*) new Matrix [c];

for (i=0; i<c; i++)

covs[i].createMatrix(nf, nf);

// calculate the mean and covariance

for (i=0; i<c; i++) {

tmp = getType(train, i);

covs[i] = cov(tmp, nf);

means[i] = mean(tmp, nf);

}

Estimate est(nf,c);

est.setMean(means);

est.setCovariance(covs);

return est;

}

1. **/lib/lr.cpp**

// returns likely class value for the given sample data with possibility of different classess

// it uses derived rule for likelihood ratio after taking log to both sides and then taking all parameters to one side

// so ratio is now tranformed to substraction such that

// likelihood1-likelihood2>0

// if it's true then sample belongs to 1st class or else it belongs to 2nd class

#include <iostream>

#include <cstdlib>

#include <cmath>

#include "Pr.h"

using namespace std;

int lr(Matrix \*mu, Matrix \*cov,Matrix &teData, int classes,Matrix &Pw)

{

// classification supposing zero-one loss

// the input sample and mean need to be column vectors

Matrix disc(1,classes);

int likelyClass=0;

for(int i=0;i<classes;i++)

disc(0,i)=-0.5\*mtod(transpose(teData-mu[i])->\*inverse(cov[i])->\*(teData-mu[i])) - 0.5\*log(det(cov[i])) + log(Pw(i,0));

for(int i=0;i<classes-1;i++)

{

if((disc(0,likelyClass)-disc(0,i+1))>0)

{

likelyClass=i;

}

else

{

likelyClass=i+1;

}

}

return likelyClass;

}

1. **/example/testLr.cpp**

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include "Matrix.h"

#include "Pr.h"

using namespace std;

#define Usage "Usage: ./testLr training\_set test\_set classes features \n\t training\_set: the file name for training set\n\t test\_set: the file name for test set\n\t classes: number of classes\n\t features: number of features (dimension)\n\t"

int main(int argc, char \*\*argv)

{

int nrTr, nrTe, // number of rows in the training and test set

nc; // number of columns in the data set; both the

// training and test set should have the same

// column number

Matrix Tr, Te;

// check to see if the number of argument is correct

if (argc < 5) {

cout << Usage;

exit(1);

}

int classes = atoi(argv[3]); // number of classes

int nf = atoi(argv[4]); // number of features (dimension)

// read in data from the data file

nc = nf+1; // the data dimension; plus the one label column

Tr = readData(argv[1], nc);

nrTr = Tr.getRow(); // get the number of rows

Te = readData(argv[2], nc);

nrTe = Te.getRow(); // get the number of columns

// prepare the labels and error count

Matrix labelLR(nrTe, 1); // a col vector to hold result for LR

int errCountLR = 0; // calcualte error rate for MPP

// assign prior probability

Matrix Pw(classes, 1);

for (int i=0; i<classes; i++)

Pw(i,0) = (float)1/classes; // if assuming equal prior probability

//Pw(0,0)=0.8;Pw(1,0)=1-Pw(0,0); // to change probability for class 0 and 1 in two class case

// calculate Estimates

Estimate est=estimateCalculation(Tr,classes);

Matrix \*mean=est.getMean();

Matrix \*cov=est.getCovariance();

//est.calculateEstimates(Tr,classes);

// to test two modal performance

/\*mean[0](0,0)=-0.75;mean[0](1,0)=0.2;

cov[0](0,0)=0.25;cov[0](0,1)=0;cov[0](1,0)=0;cov[0](1,1)=0.3;

mean[1](0,0)=0.3;mean[1](1,0)=0.3;

cov[1](0,0)=0.1;cov[1](0,1)=0;cov[1](1,0)=0;cov[1](1,1)=0.1;

Pw(0,0)=0.8;Pw(1,0)=0.2;\*/

// perform classification

for (int i=0; i<nrTe; i++) {

// classify one test sample at a time, get one sample from the test data

Matrix sample = transpose(subMatrix(Te, i, 0, i, nf-1));

// get the size of input raw data

int nctr = Tr.getCol();

int nrte = sample.getRow();

if (nctr != (nrte+1)) {

cout << "MPP: "

<< "Training and testing set do not have same number of features\n";

exit(3);

}

// call lr to perform classification

labelLR(i,0) = lr(mean, cov, sample, classes, Pw);

// check if the classification result is correct or not

if (labelLR(i,0) != Te(i,nf))

errCountLR++;

}

// calculate accuracy

cout << "The error rate using Likelihood ratio is = " << (float)errCountLR/nrTe << endl;

return 0;

}

1. **/example/testEstimate.cpp**

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include "Matrix.h"

#include "Pr.h"

using namespace std;

#define Usage "Usage: ./testLr training\_set test\_set classes features \n\t training\_set: the file name for training set\n\t test\_set: the file name for test set\n\t classes: number of classes\n\t features: number of features (dimension)\n\t"

int main(int argc, char \*\*argv)

{

int nrTr, nrTe, // number of rows in the training and test set

nc; // number of columns in the data set; both the

// training and test set should have the same

// column number

Matrix Tr, Te;

// check to see if the number of argument is correct

if (argc < 5) {

cout << Usage;

exit(1);

}

int classes = atoi(argv[3]); // number of classes

int nf = atoi(argv[4]); // number of features (dimension)

// read in data from the data file

nc = nf+1; // the data dimension; plus the one label column

Tr = readData(argv[1], nc);

nrTr = Tr.getRow(); // get the number of rows

Te = readData(argv[2], nc);

nrTe = Te.getRow(); // get the number of columns

// prepare the labels and error count

Matrix labelLR(nrTe, 1); // a col vector to hold result for LR

int errCountLR = 0; // calcualte error rate for MPP

// assign prior probability

Matrix Pw(classes, 1);

for (int i=0; i<classes; i++)

Pw(i,0) = (float)1/classes; // if assuming equal prior probability

// calculate Estimates

Estimate est=estimateCalculation(Tr,classes);

Matrix \*mean=est.getMean();

Matrix \*cov=est.getCovariance();

//est.calculateEstimates(Tr,classes);

// to test two modal performance

/\*mean[0](0,0)=-0.75;mean[0](1,0)=0.2;

cov[0](0,0)=0.25;cov[0](0,1)=0;cov[0](1,0)=0;cov[0](1,1)=0.3;

mean[1](0,0)=0.3;mean[1](1,0)=0.3;

cov[1](0,0)=0.1;cov[1](0,1)=0;cov[1](1,0)=0;cov[1](1,1)=0.1;\*/

// display estimates

for(int i=0;i<classes;i++)

{

cout<<"Mean["<<i<<"] = "<<mean[i]<<endl;

cout<<"Covariance["<<i<<"] = "<<cov[i]<<endl;

}

return 0;

}