Project 1 - Two Category Classification Using Baysian Decision Rule

University of Tennessee, Knoxville
Department of Electrical Engineering and Computer Science
ECE471/571 – Pattern Recognition
Professor: Dr. Hairong Qi

Student: Leandro Henrique Lourenco da Silva
leandrohlsilva@gmail.com

Summary

Abstract

Project 1 is a two-classification problem. By Baysian Decision Rule, this problem should be solved with as accuracy as possible using discriminant functions. Discussions and explanations about methodologies were made in order to clarify outputs and results.

Therefore, using Maximum Likelihood to estimate parameters and Baysian Decision Rules to predict a testing set, it was noticed that the analyzed data has dependent parameters and the classes have different parameters for their distribution. To predict as efficient as possible, the arbitrary discriminant function shows the best result.

Introduction

Background

Project 1 uses a problem of Two Category Classification and a Baysian Decision Rule was used to solve it. The knowledge acquired to complete this project involves Gaussian Distribution, Maximum Likelihood, Likelihood ratio, discriminant functions and notions of Matlab.

Objective

Given a training set, a decision rule must be generated to predict as better as possible a testing set. Each instance has a pair of of two real-valued co-ordinates and a class which is 0 or 1. Assuming that the data is normally distributed, the parameters must be estimated as well as Baysian decision rules. More efficient ways of fitting the testing set is also a goal.

Achievement

A great predict function was obtained by assuming that the data performs 1 Gaussian curve for each class and their covariance matrix are arbitrary. The maximum accuracy obtained was 89.011% from this case (3rd case of discriminant functions).

Using Euclidean Norm (1st case of discriminant functions, where covariance matrix is deviation), the classification rule was not as efficient as the 3rd case (1st case obtained only 71.3287% of accuracy for testing set.

Assuming that the covariance matrix are the same for all classes (2nd case. Dependent features, but independent classes), it was possible to get 88.5115% of accuracy for the testing set.

Technical Approach

Estimated parameters

In order to estimate parameters, I used Maximum Likelihood. The data analyzed is Normally Distributed, so I assumed $N(\mu,\sigma^2)$.

So, the expected value and covariance matrix are, respectively:

$$\vec{m}_i = \begin{bmatrix} m_{i1} = \frac{1}{n_i}\sum_{k=1}^{n_i} x_{k1} \\ M \\ m_{id} = \frac{1}{n_i}\sum_{k=1}^{n_i} x_{kd} \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \cdots & \sigma_{dd} \end{bmatrix} = \begin{bmatrix} \sigma_{1^2} & \cdots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \cdots & \sigma_{d^2} \end{bmatrix}$$

Language, Frameworks and Packages used

C++ in Linux environment was used to manipulate the data. A source code from Dr. Hairong Qi was improved to compute basic operations between matrices. Matlab was also necessary to generate graphs.

Discriminant Functions - Calculations

For the first case ($\Sigma_i = \sigma^2$): The features are independent from each other, what causes an equality of expected value and covariance matrix for all the data, regardless the class or features.

For the second case ($\Sigma_i = \Sigma$): The features are not independent, but they are equally distributed regardless of classes.

For the third and last case ($\Sigma_i = \Sigma$ arbitrary): The features and the classes are not independent, what makes the covariance matrix be different for each class.

The pattern classifier: $g_i(x) = -R(\alpha_i|x)$

$$R(\alpha_i|x) = \sum_{j=1}^{c} \lambda(\alpha_i|\omega_j) P(\omega_j|x)$$

In all the discriminant functions, calculations were made extensively in order to find a smaller loss. Below are listed the best losses found.

$\lambda(\alpha_1|\omega_1) = 0.95$

$\lambda(\alpha_1|\omega_2) = 0.05$

$\lambda(\alpha_2|\omega_1) = 1$

$\lambda(\alpha_2|\omega_2) = 0$

$P(w_i|x) = p(x|w_i)*P(w_i)$

$$p(\vec{x}|w) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1}(\vec{x}-\vec{\mu})\right]$$

Experiments and results

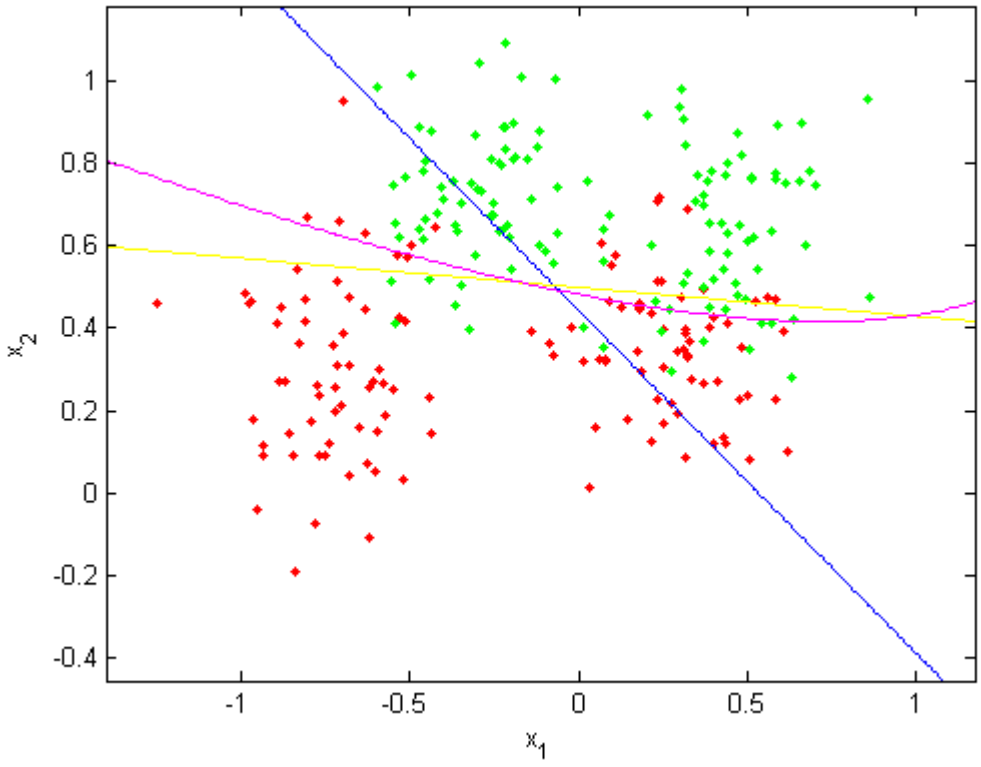
Experiment Design

The experiment was designed to receive a training set (containing 250 instances in this case) in an effort to predict 1000 instances of a testing set.

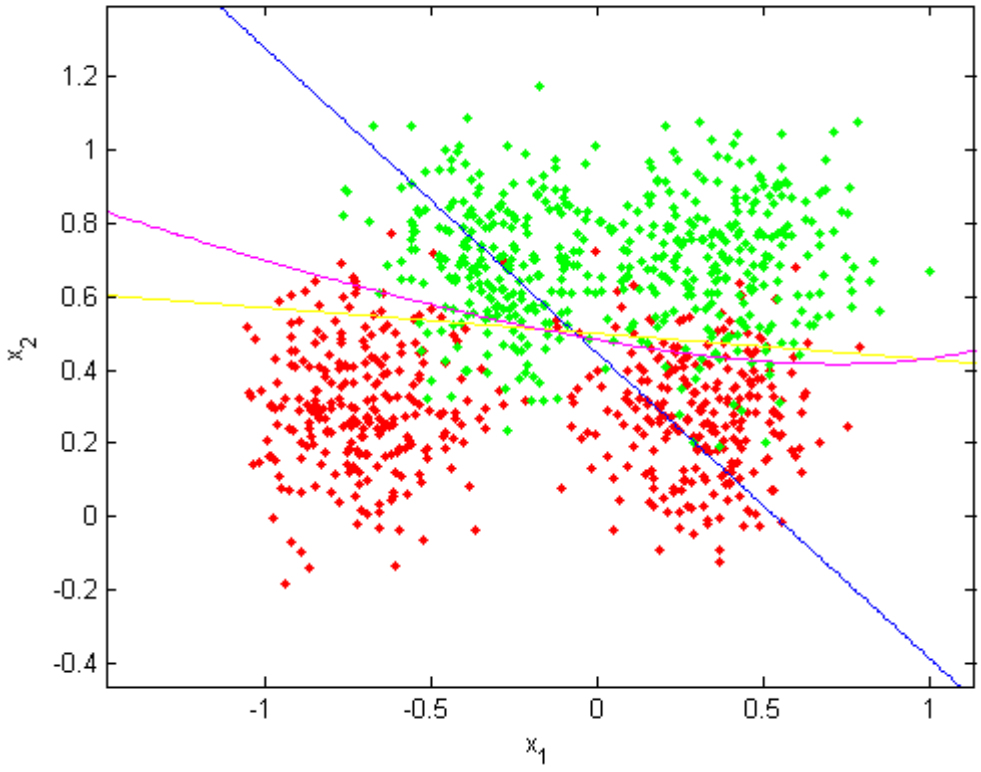
Each instance has 2 features and a classification. In other words, each set is a matrix nxd+1, where n is the quantity of instances and d + 1 is the dimension (features) plus the classification.

Assuming that the data is normally distributed, the algorithm divides the training data between classes. The mean and the covariance matrix are calculated and, according to the discriminant function method, the calculations are made.

Graph of training set and decision rules



Graph of testing set and decision rules



Legend

Red dots: Class 1 instances
Green dots: Class 2 instances
Blue line: 1st case of discriminant function (features are independents)
Yellow line: 2nd case of discriminant function (features dependents, but same covariance matrix for all classes)
Pink line: 3rd case of discriminant function (covariance matrix is arbitrary)

According to this graph, the blue line do not predict well the data. It happens because the features are considered independent and they have the same variance. However it is clearly visible that the features are not independent. This is a linear function, that's why it is represented by a line.

The 2nd case (yellow line) can predict much better than the first case, because it considers dependency between features, but classes have the same variance. This case is still a linear function, so it is represented by a line.

The 3rd and last case (pink line) is a quadratic function because their covariance matrices are different due to features and classes are dependents.

Result (output)

Output for Training set

```
Mean 1:
-0.22147
0.325755

Mean 2:
0.0759543
0.682969

Standard Deviation feature 1 = 0.274595
detClass1 = 0.00568553
invClass1 =
13.2622    0
   0 13.2622

probClass1= 0.5
detClass2 = 0.00568553
invClass2 =
13.2622    0
   0 13.2622

probClass2= 0.5
Discriminant Function Case 1 (features independents) - accuracy
= 0.712
Lambda1,1=0.0499998;Lambda1,2=0.95;Lambda2,1=1;Lambda2,2=0
detClass1 = 0.00987202
invClass1 =
3.65873 -1.13742
-1.13742 28.0398

probClass1= 0.5
detClass2 = 0.00987202
invClass2 =
```

```
3.65873 -1.13742
-1.13742 28.0398


probClass2= 0.5
Discriminant Function Case 2 (covariance matrix equal for all features - accuracy = 0.844
Lambda1,1=0.0499998;Lambda1,2=0.95;Lambda2,1=1;Lambda2,2=0
detClass1 = 0.00987202
invClass1 =
3.65873 -1.13742
-1.13742 28.0398


probClass1= 0.5
detClass2 = 0.00454321
invClass2 =
6.59411 3.42819
3.42819 35.1619


probClass2= 0.5
Discriminant Function Case 3 (arbitrary) - accuracy = 0.828
Lambda1,1=0.0499998;Lambda1,2=0.95;Lambda2,1=1;Lambda2,2=0
Process completed
```

Output for Testing set

```
Mean 1:
-0.22147
0.325755

Mean 2:
0.0759543
0.682969

Standard Deviation feature 1 = 0.274595
detClass1 = 0.00568553
invClass1 =
13.2622    0
   0 13.2622

probClass1= 0.5
detClass2 = 0.00568553
invClass2 =
```

```
13.2622    0
   0 13.2622

probClass2= 0.5
Discriminant Function Case 1 (features independents) - accuracy
= 0.713287
Lambda1,1=0.0499998;Lambda1,2=0.95;Lambda2,1=1;Lambda2,2=0
detClass1 = 0.00987202
invClass1 =
3.65873 -1.13742
-1.13742 28.0398

probClass1= 0.5
detClass2 = 0.00987202
invClass2 =
3.65873 -1.13742
-1.13742 28.0398

probClass2= 0.5
Discriminant Function Case 2 (covariance matrix equal for all
features - accuracy = 0.885115
Lambda1,1=0.0499998;Lambda1,2=0.95;Lambda2,1=1;Lambda2,2=0
detClass1 = 0.00987202
invClass1 =
3.65873 -1.13742
-1.13742 28.0398

probClass1= 0.5
detClass2 = 0.00454321
invClass2 =
6.59411 3.42819
3.42819 35.1619

probClass2= 0.5
Discriminant Function Case 3 (arbitrary) - accuracy = 0.89011
Lambda1,1=0.0499998;Lambda1,2=0.95;Lambda2,1=1;Lambda2,2=0
Process completed
```

Performance Evaluation

In order to take advantage of the matrix lambda and increase the accuracy of the decision rule, a algorithm was created to find the best match.
As showed above, the best lambda matrix found is [0.05 0.95; 1 0].
The first discriminant function case got 71.3287% of accuracy in predicting classes.
The second discriminant function case got 88.5115% of accuracy in predicting classes.
The most complex function got 89.011% of accuracy in predicting classes.

Discussion

So far, all the necessary tools to complete this project was taught, like Maximum

Likelihood estimation, Bayes Rule, Bayes Decision Rule, Likelihood ratio, etc.

Reference

Dr. Hairong Qi's Matrix source Code - http://web.eecs.utk.edu/~qi/ece471-571/matrixcpp.htm

ECE 471 (presentations) - http://web.eecs.utk.edu/~qi/ece471-571/syllabus.htm

More about Maximum Likelihood - http://mathworld.wolfram.com/MaximumLikelihood.html

Appendix

Matlab code

```
Tr = […] %training set
class1 = tr(1:124,1:2)
class2 = tr(125:250,1:2)
mean1 = [-.22147;.325755]
mean2 = [0.0759543;.682969]
detClass1 = 0.00568553
invClass1 = [13.2622 0; 0 13.2622]
detClass2 = detClass1
invClass2 = invClass1
probClass1 = .5
probClass2 = .5


%scatt1 = scatter(class1(:,1),class1(:,2))


plot(class1(:,1), class1(:,2), 'r.', class2(:,1), class2(:,2), 'g.')
hold on

syms x1;
```

```
syms x2;
feat = [x1;x2]


g1 = (1/(2*pi*sqrt(detClass1)))*exp(-.5*transpose(feat - mean1)*invClass1*(feat - mean1))*probClass1
g2 = (1/(2*pi*sqrt(detClass2)))*exp(-.5*transpose(feat - mean2)*invClass2*(feat - mean2))*probClass2
strG1 = char(g1)
strG2 = char(g2)
expression = sprintf('0 = %s - %s', strG1, strG2);
pl1 = ezplot(expression, [-2 1.5 -2 1.5])

detClass1 = 0.00987202
invClass1 = [3.65873 -1.13742; -1.13742 28.0398]
detClass2 = 0.00987202
invClass2 = invClass1

g1 = (1/(2*pi*sqrt(detClass1)))*exp(-.5*transpose(feat - mean1)*invClass1*(feat - mean1))*probClass1
g2 = (1/(2*pi*sqrt(detClass2)))*exp(-.5*transpose(feat - mean2)*invClass2*(feat - mean2))*probClass2
strG1 = char(g1)
strG2 = char(g2)
expression = sprintf('0 = %s - %s', strG1, strG2);
pl2 = ezplot(expression, [-2 1.5 -2 1.5])


detClass1 = 0.00987202
invClass1 = [3.65873 -1.13742; -1.13742 28.0398]
detClass2 = 0.00454321
invClass2 = [6.59411 3.42819; 3.42819 35.1619]

g1 = (1/(2*pi*sqrt(detClass1)))*exp(-.5*transpose(feat - mean1)*invClass1*(feat - mean1))*probClass1
g2 = (1/(2*pi*sqrt(detClass2)))*exp(-.5*transpose(feat - mean2)*invClass2*(feat -
```

```
mean2))*probClass2
strG1 = char(g1)
strG2 = char(g2)
expression = sprintf('0 = %s - %s', strG1, strG2);
pl3 = ezplot(expression, [-2 1.5 -2 1.5])


set(pl1, 'Color', 'b');
set(pl2, 'Color', 'y');
set(pl3, 'Color', 'm');
```

project1.cpp

```
/*
 *    ECE471 - Pattern Recognition
 *
 *    Undergrad: Leandro Henrique Lourenco da Silva
 *    ID: 000356622
 *    Date: 01/25/12
 *
 */
#include <iostream>
#include <fstream>
#include <cmath>
#include <cstdlib>
#include "Matrix.h"
#include "Pr.h"
#include <vector>

using namespace std;

#define Usage "Usage: ./project1 trainingData testData\n"

#define PI 3.14159

int main(int argc, char **argv)
{
```

```
	// check to see if the number of argument is correct
	if (argc < 3) {
		cout << Usage;
		exit(1);
	}

	//read data and fill matrix
	FILE* rawData = fopen(argv[1], "r");
	float xs, ys;
	int yc;
	vector<float> xsClass1, ysClass1, xsClass2, ysClass2;

	int class1Instances = 0, class2Instances = 0;

	while (!feof(rawData)) {
		fscanf(rawData, "%f %f %d\n", &xs, &ys, &yc);
		if (yc == 0) {
			xsClass1.insert(xsClass1.end(), xs);
			ysClass1.insert(ysClass1.end(), ys);
			class1Instances++;
		} else {
			xsClass2.insert(xsClass2.end(), xs);
			ysClass2.insert(ysClass2.end(), ys);
			class2Instances++;
		}
	}

	double probClass1 = ((double)class1Instances/(double)
(class1Instances+class2Instances));
	double probClass2 = ((double)class2Instances/(double)
(class1Instances+class2Instances));

	//filling vector with data
	Matrix matClass1(xsClass1.size(), 2);
	for (int i=0; i<matClass1.getRow(); i++) matClass1(i, 0) = xsClass1[i];
	for (int i=0; i<matClass1.getRow(); i++) matClass1(i, 1) = ysClass1[i];
	Matrix matClass2(xsClass2.size(), 2);
	for (int i=0; i<matClass2.getRow(); i++) matClass2(i, 0) = xsClass2[i];
	for (int i=0; i<matClass2.getRow(); i++) matClass2(i, 1) = ysClass2[i];
```

```
	//cout << "matClass1" << endl << matClass1 << endl;
	//cout << "matClass2" << endl << matClass2 << endl;


	//vector mean1 represents class1

	Matrix mean1 = mean(matClass1, 2);
	Matrix mean2 = mean(matClass2, 2);
	cout << "Mean 1:" << endl << mean1 << endl;
	cout << "Mean 2:" << endl << mean2 << endl;


	//to calculate discrimant function type1, we assume that features are
independent
	double stdDeviation = 0;

	for (int i = 0; i < matClass1.getRow(); i++) {
		Matrix line = subMatrix(matClass1, i, 0, i, 1);
		stdDeviation += pow((line(0, 0) - mean1(0, 0)), 2);
	}

	stdDeviation /= matClass1.getRow();

	cout << "Standard Deviation feature 1 = " << stdDeviation << endl;


	//calculating SIGMA matrix (covariance Matrix)

	//Matrix sigmaClass1 = cov(matClass1, 2);
	Matrix sigmaClass2;
	Matrix sigmaClass1(2,2);
	sigmaClass1(0,0) = stdDeviation*stdDeviation;
	sigmaClass1(1,1) = sigmaClass1(0,0);
	sigmaClass1(0,1) = 0;
	sigmaClass1(1,0) = 0;

	Matrix invClass1 = inverse(sigmaClass1);
	Matrix invClass2 = inverse(sigmaClass1);
```

```
	//Now test if test set fits our decision rule

	Matrix testSet = readData(argv[2], 3);

	//calculating likelihood ratio first

	double detClass1 = det(sigmaClass1);
	double detClass2 = detClass1;


	//Im going to create a new matrix to keep predicted class and error

	int testCase = 0;
	while (testCase < 3) {

		float bestLambda1 = 1.0;
		float bestLambda2 = 1.0;
		double bestAccuracy = 0;

		for (float lambdaClass1 = 1.0; lambdaClass1 > 0; lambdaClass1 -= 0.05)
{
			bestLambda1 = 1.0;
			bestLambda2 = 1.0;
			bestAccuracy = 0;
			for (float lambdaClass2 = 1.0; lambdaClass2 > 0; lambdaClass2 -=
0.05) {


				Matrix predictedMatrix(testSet.getRow(), 2);

				int correctGuesses = 0;

				for (int i = 0; i < testSet.getRow(); i++) {
					Matrix line(1,2);
					line = subMatrix(testSet, i, 0, i, 1);
					line = transpose(line);
```

```
                        Matrix mahalanobis = ((line - mean1));
                        mahalanobis = transpose(mahalanobis);
                        mahalanobis = mahalanobis*invClass1;
                        mahalanobis = mahalanobis*(line - mean1);
                        double varMahalanobis = (-0.5)*mahalanobis(0, 0);


                        double probIsClass1 = (1.0/
(2*PI*detClass1))*exp(varMahalanobis)*(probClass1);


                        mahalanobis = (line - mean2);
                        mahalanobis = transpose(mahalanobis);
                        mahalanobis = mahalanobis*invClass2;
                        mahalanobis = mahalanobis*(line - mean2);
                        varMahalanobis = (-0.5)*mahalanobis(0, 0);

                        double probIsClass2 = (1.0/
(2*PI*detClass2))*exp(varMahalanobis)*(probClass2);

                        double aux = probIsClass1;
                        probIsClass1 = lambdaClass1*probIsClass1 + (1 -
lambdaClass1)*probIsClass2;
                        probIsClass2 = lambdaClass2*probIsClass2 + (1 -
lambdaClass2)*aux;

                        int predictedClass = 0;
                        if (probIsClass2 > probIsClass1) predictedClass = 1;

                        if (predictedClass == testSet(i, 2)) correctGuesses+
+;

                        double error = min(probClass2, probClass1);

                        predictedMatrix(i, 1) = error;
                        predictedMatrix(i, 0) = predictedClass;

                    }

                    double acc = ((double)correctGuesses/
```

```
(double)testSet.getRow());
                        if (acc > bestAccuracy) {
                              bestAccuracy = acc;
                              bestLambda1 = lambdaClass1;
                              bestLambda2 = lambdaClass2;
                        }

                  }

            }

            cout << "detClass1 = " << detClass1 << endl;
            cout << "invClass1 = " << endl << invClass1 << endl;
            cout << "probClass1= " << probClass1 << endl;
            cout << "detClass2 = " << detClass2 << endl;
            cout << "invClass2 = " << endl << invClass2 << endl;
            cout << "probClass2= " << probClass2 << endl;

            if (testCase == 0) {
                  cout << "Discriminant Function Case 1 (features independents) -
accuracy = "
                  << bestAccuracy << " Lambda1,1=" << bestLambda1 << ";Lambda1,2="
<< (1 - bestLambda1) <<
                  ";Lambda2,1=" << bestLambda2 << ";Lambda2,2=" << (1 -
bestLambda2) << endl;

                  //calculating new covariance Matrix (case 2: covariance matrix is
equal)
                  sigmaClass1 = cov(matClass1, 2);
                  sigmaClass2 = sigmaClass1;
                  detClass1 = det(sigmaClass1);
                  detClass2 = detClass1;
                  invClass1 = inverse(sigmaClass1);
                  invClass2 = invClass1;

            } else if (testCase == 1) {
                  cout << "Discriminant Function Case 2 (covariance matrix equal
for all features - accuracy = "
                  << bestAccuracy << " Lambda1,1=" << bestLambda1 << ";Lambda1,2="
<< (1 - bestLambda1) <<
```

```
                  ";Lambda2,1=" << bestLambda2 << ";Lambda2,2=" << (1 -
bestLambda2) << endl;
                  //calculating new covariance matrices (case 3: covariance matrix
are different)
                  //sigma1 and invClass1 are already calculated
                  sigmaClass2 = cov(matClass2, 2);
                  invClass2 = inverse(sigmaClass2);
                  detClass2 = det(sigmaClass2);


            } else if (testCase == 2) {
                  cout << "Discriminant Function Case 3 (arbitrary) - accuracy = "
                        << bestAccuracy << " Lambda1,1=" << bestLambda1 <<
";Lambda1,2=" << (1 - bestLambda1) <<
                  ";Lambda2,1=" << bestLambda2 << ";Lambda2,2=" << (1 -
bestLambda2) << endl;
            }




            testCase++;
      }


      cout << "Process completed" << endl;


      //writeData(predictedMatrix, "predictedValues.txt");


      return 0;
}
```