Two Category Classification Using Bayesian Decision Rule

Introduction to Pattern Recognition/ECE-471

William Willie Wells

wwells4@utk.edu

Two Category Classification Using Bayesian Decision Rule

Abstract:

Two category classification using Bayesian decision rule uses maximum likelihood estimation, likelihood ratio, and the discriminant function to classify a synthetic data set with two features. The attached code generates a decision boundary between the two categories using a training set and tests the resultant decision boundary on a synthetic test set. A different decision boundary is generated for each of maximum likelihood estimation, likelihood ratio, and the three cases of the discriminant function. No single method is perfect and loss will always occur but each of these methods result in reasonable decision boundaries.

Introduction:

Humanity has always strived for better ways of performing individual functions. Being able to identify whether an item belongs to one class or another just by identifying its feature set may save lives in situations where time is a factor. Bayesian decision rule attempts to identify an item as belong to a specific class based on its features. This project achieves the designing of a decision rule on a synthetic data set with two categories.

Technical Approach:

Since I recently returned to academic studies from being a part of the military I had to reteach myself some of the finer points of C++. I created many separate functions to implement common routines. The first of these functions finds where the class identifier of the training set changes and returns that index to the function calling it. Function MaP generates a probability density function (pdf) from the subset of separated data. It calculates the constant first, then calculates x-μ, the squared Mahalanobis distance, and finally the entire probability density function. Initially I did not have a separate function for calculating squared Mahalanobis distance but after building a function to deal with the case-2 discriminant I made it a separate function. My version of squared Mahalanobis distant also divides the result by -2. I created a function to generate a generic linear n-by-d matrix centered at a mean that extends an estimated 5 standard deviations (5 standard deviations covers approximately 99% of all data in a Gausian distribution) using 101 iterations. A function compares the two pdf's generated by each class and returns a vector with ones at the index where the pdf's equal or where the sign of their difference changes and zeros everywhere else. I use a separate function getX to get the respective class one x-vector coordinates associated with the index. While designing the function for the case-2 discriminant I also noticed that I was using similar code as in the case-1 discriminant to calculate $\mu^t*(\sum^{-1})^t*x$ and μ^t*x, thus I designed muSigmaX to calculate the final result of $g_1(x)$-$g_2(x)$. I had to modify muSigmaX once more to accommodate the case-3 discriminant to handle the squared Mahalanobis distance x-vector component. I developed functions for each individual discriminant function case. Finished with designing the decision boundary rules I created useClassifier to use the decision boundary rules on the synthetic test set, which classifies a set of test data as one of the two classes. I created a function copyMat to copy the test data into a larger matrix that accommodates the results of the individual tests as well as the initial data within its matrix.   Within the main body of the code I calculate each mean and covariance matrix associated with the individual classes by calling the given functions of the class Matrix used throughout the project.

Experiments and results:

While each method of classifying a two category data set is not perfect each method does a reasonable job at classifying the synthetic data set. This is some sample output from the synthetic test set:

```
Xs      ys       yc       MLE    LR      Case1  Case2  Case3

0.39268 0.090262    0    0    0    1    1    0

0.431409 0.288456    0    0    0    1    1    0

-0.516452 0.501256    1    1    1    0    0    0

-0.116775 0.483405    1    1    1    0    0    0

-0.327961 0.54624    1    0    1    1    1    0

0.161322 0.950856    1    0    0    1    1    1

0.350961 0.686428    1    0    1    1    1    1

0.0902574 0.846995    1    0    0    1    1    1
```

Discussion:

Bayesian decision rule is able to classify a synthetic data set based on boundary decision rules. The efforts of someone working with pattern recognition must be to minimize acceptable loss. As mention earlier in the medical field such methods may save lives. The may yet be better algorithms out there to minimize loss, maximize true positives, and minimizes false negatives.

```
/*
 *  mpp.cpp
 *
 *    Purpose: design a decision rule on a synthetic data set with two features
 *                        Assuming proability density is Gaussian, P(f1)=P(f2)=0.5, and zero-one loss.
 *
 *    Command-line inputs:
 *        - name of the data file (the data file shouldn't contain any
 *          textural information. Head row should be manually deleted if any
 *        - number of features (d)
 *
 *    Author: William Willie Wells
 *
 */
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cmath>
#include "Matrix.h"
#include "Pr.h"

using namespace std;

#define Usage "Usage: ./readwrite filename nr_of_feature \n Note 1: We assume the data file is in the following
format:\n \t\tfeature1 feature2 ... featuren type \n Note 2: All feature values are floating point, all type values are
integer.\n"

// Find where class identifier changes in requested data format
// and return index
int findClassSplit(Matrix &tdata){
	int tf,flag;

	// find where class identifier changes ex: 0-->1
	tf = tdata(0,tdata.getCol()-1);
	for (int i=0; i<tdata.getRow(); i++){
		flag=tdata(i,tdata.getCol()-1);
		if(flag != tf){
			return i;
		}
		tf=flag;
	}
	return 0;
}

// Calculate Mahalanobis distance and return a 1-by-1 matrix
Matrix mDist(Matrix &mux, Matrix &sigma){
	Matrix temp;
	Matrix tmd,tS,tXk;

	// initiate matrices
	temp.createMatrix(mux.getRow(),1);
	tmd.createMatrix(1,1);
	tXk.createMatrix(sigma.getCol(),1);
	tS=inverse(sigma);

	// compute Mahalanobis distance squared / -2
```

```
        for(int i=0;i<mux.getRow();i++){
                for(int j=0;j<mux.getCol();j++){
                        tXk(j,0)=mux(i,j);
                }
                tmd=transpose(tXk)->*tS->*tXk;
                temp(i,0)=(-0.5)*tmd(0,0);
        }

        return temp;
}

// Generate a n-by-d x matrix based on mean and estimated
// std deviation with 2n+1 iterations
Matrix genX(Matrix &mu, Matrix &sigma, int n){
        Matrix xDelta,x;

        // intiate n-by-d x matrix and d-by-1 change in x matrix
        x.createMatrix(2*n+1,sigma.getCol());
        xDelta.createMatrix(sigma.getRow(),1);

        // calculate change in x based on estimated std deviation
        for(int i=0;i<sigma.getCol();i++){
                xDelta(i,0)=5*sqrt(sigma(i,i))/n;
        }

        // create a n-by-d x matrix based on mean and estimated
                          // std deviation
        for(int j=0;j<2*n+1;j++){
                for(int k=0;k<x.getCol();k++){
                        if(j==0){
                                x(j,k)=-1*n*xDelta(k,0)+mu(k,0);
                        }else{
                                x(j,k)=x(j-1,k)+xDelta(k,0);
                        }
                }
        }

        return x;
}

// Retrieve the x-vector coordinates associated with the index
// of a f(x) decision boundary
Matrix getX(Matrix &xBound, Matrix &x){
        Matrix temp;
        int bCount=0;

        // check matching row dimension
        if(xBound.getRow()!=x.getRow()){
                cout << "Matrix rows unegual" << endl;
                exit(1);
        }

        // find amount of boundary points and allocate temp
        for(int i=0;i<x.getRow();i++){
                if(xBound(i,0)!=0){
                        bCount++;
```

```
            }
        }
        temp.createMatrix(bCount,x.getCol());

        // assign x-vector to temp
        for(int j=0;j<x.getRow();j++){
            if(xBound(j,0)!=0){
                for(int k=0;k<x.getCol();k++){
                    temp(2-bCount,k)=x(j,k)*xBound(j,0);
                }
                bCount--;
            }
        }

        return temp;
}

// Calculate multivariate probability density function, return pdf
Matrix MaP(Matrix &mu, Matrix &sigma, Matrix &x){
        Matrix temp, temp1;
        /*const*/ double pi=3.14159265359793;
        double c_g;

        // calculate Gausian constant in multivariate
        c_g=1/sqrt(2*pi);
        for(int d=0;d<sigma.getCol()-2;d++){
            c_g*=c_g;
        }
        c_g/=sqrt(det(sigma));

        // calculate x-mu
        temp1.createMatrix(x.getRow(),x.getCol());
        for(int i=0;i<x.getRow();i++){
            for(int j=0;j<x.getCol();j++){
                temp1(i,j)=x(i,j)-mu(j,0);
            }
        }

        // calculate pdf (probability density function)
        temp=mDist(temp1,sigma);
        for(int k=0; k<temp.getRow(); k++){
            for(int y=0; y<temp.getCol(); y++){
                temp(k,y)=exp(temp(k,y))*c_g;
            }
        }

        return temp;
}

// Compare 2 probability density function classes
// and return decision boundary
Matrix comparePdf(Matrix &f1,Matrix &f2){
        Matrix temp;
        double tminus=0;

        // check matrix dimensions
```

```
	if(f1.getRow() != f2.getRow() || f1.getCol() != f2.getCol()){
		cout << "Matrix dimensions unequal" << endl;
		return temp;
	}

	// find decision boundary at f1-f2=0 or  f1-f2 sign changes
			// from f1(i-1)-f2(i-1), cross 0 but not equal 0
	temp.createMatrix(f1.getRow(),f1.getCol());
	for(int j=0;j<f1.getCol();j++){
		for(int i=0;i<f1.getRow();i++){
			if(i>0){
				if(f1(i,j)==f2(i,j) || (tminus!=0 && signbit(tminus) != signbit(f1(i,j)-f2(i,j)))){
					temp(i,j)=1;
				}
			}
			tminus=f1(i,j)-f2(i,j);
		}
	}

	return temp;
}

// Computes transpose(mu)*transpose(inverse(covariance))*x-vector
// and adds x-vector Mahalanobis distance if applicable
Matrix muSigmaX(Matrix &muSigma, Matrix &x, Matrix &tc, Matrix &tXmd){
	Matrix temp;
	Matrix tRes,tResX,tXk;
	int b=0;

	// initiate matrices
	temp.createMatrix(muSigma.getCol(),muSigma.getCol());
	tRes.createMatrix(x.getRow(),muSigma.getRow());
	tXk.createMatrix(muSigma.getCol(),muSigma.getRow());

	// compute mu*sigma*x or mu*x
	for(int i=0;i<x.getRow();i++){
		for(int j=0;j<x.getCol();j++){
			tXk(j,0)=x(i,j);
		}
		tResX=muSigma->*tXk+tc;
		tRes(i,0)=tResX(0,0)+tXmd(i,0);
		if(signbit(tRes(i,0))!=signbit(tRes(i-1,0))){
			temp(b,0)=x(i,0);
			temp(b,1)=x(i,1);
			b++;
		}
	}

	return temp;
}

// Calculate for case 1 multiple of identity matrix covariance g_1(x)-g_2(x)
Matrix sameSigma(Matrix &mu1,Matrix &mu2, Matrix &x){
	Matrix tC,tMix,tEmpty;

	// calculate constants
```

```
        tC=transpose(mu1)->*mu1-transpose(mu2)->*mu2;
        tC(0,0)*=-0.5;

        // transpose of the difference between means
        tMix=(transpose(mu1)-transpose(mu2));

                        // initiate empty (zero) x-vector Mahalanobis distance
                        // and calculate g_1(x)-g_2(x)
        tEmpty.createMatrix(x.getRow(),1);
        return muSigmaX(tMix,x,tC,tEmpty);
}

// Calculate for case 2 same covariances g_1(x)-g_2(x)
Matrix sameCov(Matrix &mu1, Matrix &mu2, Matrix &sigma, Matrix &x){
        Matrix tMix,tS,tC,tMu1,tMu2,tEmpty;

        // Mahalanobis distance for each mean
        tMu1=transpose(mu1);
        tMu2=transpose(mu2);
        tC=mDist(tMu1,sigma)-mDist(tMu2,sigma);

        // differnce of means*single inverse covariance
        tS=inverse(sigma);
        tMix=(transpose(mu1)-transpose(mu2))->*transpose(tS);

                        // initiate empty (zero) x-vector Mahalanobis distance
                        // and calculate g_1(x)-g_2(x)
        tEmpty.createMatrix(x.getRow(),1);
        return muSigmaX(tMix,x,tC,tEmpty);
}

// Calculate for case 3 differing, arbitrary, covariances g_1(x)-g_2(x)
Matrix arbCov(Matrix &mu1, Matrix &mu2, Matrix &s1, Matrix &s2, Matrix &x){
        Matrix tMix,tC,tX;
        Matrix tS,tS1,tS2,tMu,tMu1,tMu2;

        // Mahalanobis distance for each covariance (x, mean)
        tX=mDist(x,s1)-mDist(x,s2);
        tMu1=transpose(mu1);
        tMu2=transpose(mu2);
        tMu=mDist(tMu1,s1)-mDist(tMu2,s2);

        // calculate constants
        tS.createMatrix(1,1);
        tS(0,0)=(det(s1))/(det(s2));
        tC.createMatrix(1,1);
        tC(0,0)=(log(tS(0,0)))*(-0.5)+tMu(0,0);

        // differnce of individual means*inverse covariance
        tS1=inverse(s1);
        tS2=inverse(s2);
        tMix=(transpose(mu1))->*transpose(tS1)-(transpose(mu2))->*transpose(tS2);

                        // calculate g_1(x)-g_2(x)
        return muSigmaX(tMix,x,tC,tX);
}
```

```
// Apply decision boundary rule to test set
Matrix useClassifier(Matrix &data,Matrix &dRule,Matrix &rDat,int n){
    Matrix temp;

    // test for 2-by-2 matrix vs 1-by-2 matrix
    // classify data points based on boundary rule
    if(dRule(1,0)==0){
        for(int k=0;k<data.getRow();k++){
            if(data(k,0)<dRule(0,0) && data(k,1)<dRule(0,1)){
                rDat(k,n)=0;
            }else{
                rDat(k,n)=1;
            }
        }
    }else{
        for(int i=0;i<data.getRow();i++){
            if(data(i,0)>dRule(0,0) && data(i,1)> dRule(0,1) && data(i,0)<dRule(1,0) &&
data(i,1)<dRule(1,1)){
                rDat(i,n)=1;
            }else{
                rDat(i,n)=0;
            }
        }
    }
    temp=rDat;

    return temp;
}

// Copy values of one Matrix into another without losing size
Matrix copyMat(Matrix &data, Matrix &rDat){
    Matrix temp;

    for(int i=0;i<data.getRow();i++){
        for(int j=0;j<data.getCol();j++){
            rDat(i,j)=data(i,j);
        }
    }
    temp=rDat;

    return temp;
}

// Two category classification using Baysian decision rule
int main(int argc, char **argv){
    int d,mcut;
    int n=50;

    // error check input format
    if(argc<3){
        cout << Usage;
        exit(1);
    }
    Matrix tdat,tdat1,tdat2,tedat,teDatC,teC;// input data
    Matrix mu,mu1,mu2,sigma,s1,s2;// mean + covariance of individual classes and entire set
    Matrix p1,p2,qbp,qbxp;// maximum a-posterior
    Matrix lr, lro, qbl,qbxl;// likelihood ratio
    Matrix x1,x2,x;// feature grid matrices centered at mean
    Matrix ide,ave,arb;// discriminant functions

    // read in data
    d=atoi(argv[2]);
    tdat = readData(argv[1], d);
    tedat =readData(argv[3],d);

    // separate data into distinct classes
    mcut = findClassSplit(tdat);
    tdat1=subMatrix(tdat,1,0,mcut-1,d-1);
    tdat2=subMatrix(tdat,mcut,0,tdat.getRow()-1,d-1);
    teDatC=subMatrix(tedat,1,0,tedat.getRow()-1,1);
    teC=subMatrix(tedat,1,0,tedat.getRow()-1,2);

    // mean + covariance of individual classes and entire set
    mu1=mean(tdat1,d-1);
    mu2=mean(tdat2,d-1);
    s1=cov(tdat1,d-1);
    s2=cov(tdat2,d-1);
    mu=mean(tdat,d-1);
    sigma=cov(tdat,d-1);

    // evenly spaced feature grid centered at the mean
    x=genX(mu,sigma,n);
    x1=genX(mu1,s1,n);
    x2=genX(mu2,s2,n);

    // Maximum a posterior
    p1=MaP(mu1,s1,x1);
    p2=MaP(mu2,s2,x2);
    qbp=comparePdf(p1,p2);// returns p(x) not x
    qbxp=getX(qbp,x1);// returns x from p(x)

    // likelihood ratio
    lr=p1/p2;
    lro.createMatrix(lr.getRow(),lr.getCol());
    lro.initMatrix(1.0);
    qbl=comparePdf(lr,lro)*p2;// returns p(x) not x
    qbxl=getX(qbl,x1);// returns x from p(x)

    //discriminant functions cases 1-3
    //case 1
    ide=sameSigma(mu1,mu2,x);// returns x boundary single value

    //case 2
    ave=sameCov(mu1,mu2,sigma,x);// returns x boundary single x-vector

    //case 3
    arb=arbCov(mu1,mu2,s1,s2,x);// returns x boundary

    // use rules on test data
    Matrix cData;
    cData.createMatrix(teC.getRow(),teC.getCol()+5);
```

```
        cData=copyMat(teC,cData);
        cData=useClassifier(teDatC,qbxp,cData,teC.getCol());
        cData=useClassifier(teDatC,qbxl,cData,teC.getCol()+1);
        cData=useClassifier(teDatC,ide,cData,teC.getCol()+2);
        cData=useClassifier(teDatC,ave,cData,teC.getCol()+3);
        cData=useClassifier(teDatC,arb,cData,teC.getCol()+4);

        // output cData (results of test) to a file
        writeData(cData, "synth.txt");

        // output desicion boundarys to the command line
        cout << "MAP decision boundary:" << endl;
        cout << qbxp << endl;
        cout << "Likelihood Ratio:" << endl;
        cout << qbxl << endl;
        cout << "Discriminant Function Case 1:" << endl;
        cout << ide << endl;
        cout << "Discriminant Function Case 2:" << endl;
        cout << ave << endl;
        cout << "Discriminant Function Case 3:" << endl;
        cout << arb << endl;

        return 0;
}
```