Project 1

Boling, Kenneth S.[1]
(1)Earth and Planetary Sciences, University of Tennessee, Knoxville, TN 37996
kboling4@vols.utk.edu
ECE 571

Abstract

The most basic types of classifiers using Bayesian statistics are supervised, parametric algorithms that require both a labeled training dataset and an estimation of the parameters of the distribution of the testing dataset. Here a synthetic dataset consisting of a labeled training and unlabeled testing dataset was used to compare the accuracy of four different classifiers. These classifiers were used to define a decision boundary to classify a larger testing dataset. The testing and training data sets were both determined to be able to be represented by a bi-modal Gaussian distribution.

Contents

Abstract 1
Introduction 2
Methods and Technical Approach 3
Maximum likelihood estimation of parameters 3
Case I (Minimum Euclidian Distance) 4
Case II (Minimum Mahalanobis Distance) 5
Case III (Maximum A-Priori Probability) 5
Experiments and Results 6
Classifier performance 9
Discussion 10
References 10
Appendix 10

Introduction

This project implemented several different statistical classifiers. These are supervised classifiers, which use a set of training data where the label or class of each sample is known. These known values are then used to decide which class a given unknown sample from the testing dataset belongs to. In contrast, unsupervised classifiers do not have a predefined set of classes, but rather derive them by clustering unlabeled samples into groups based on their similarity to each other.

All of the classification methods implemented for this project are parametric, meaning that some assumptions about the parameters (such as distribution of data for each class) must be made. For example, the probability density function determined for a set of samples is assumed to be normally distributed around the mean. In contrast, part of Project 2 involved developing a k-Nearest Neighbor classifier, which does not assume anything about the distribution and simply determines the distance between an unclassified testing point and *k*-points in the training set and assigns the unknown to the class with the majority. This is an example of non-parametric classification.

The foundation of these classifiers is based on Bayes' rule:

$$P(\omega_j|x) = \frac{p(x|\omega_j)P(\omega_j)}{p(x)}$$

The *posterior* probability: $P(\omega_j|x)$ is the conditional probability that x belongs to the class ω_j for a given x value. The *likelihood,* or the conditional probability density: $p(x|\omega_j)$ is the probability density function that describes the distribution of the probability of the likely values of a random x. The prior probability: $P(\omega_j)$ is the probability that is assigned based on previously determined relationships or prior knowledge about the data. The prior probability acts as a scaling function of the probability density function between two or more classes. Finally the normalization constant $p(x)$ is the sum of the product of the likelihood and prior probabilities between all possible classes.

The objective of this project was to design a decision rule using a synthetic data set. The provided data set is in the form of two tab delimited text files: *synth.tr* and *synth.te* which contain the training and testing datasets respectively. These files were obtained from: (B.D. Ripley, 1996). The *synth.tr* data set, contains n=250 data points, with values for "xs" and "ys" with a class identifier "yc". 125 belong to class A (yc=0) and 125 belong to class B (yc=1). This data set was used to train the decision rule. The *synth.te* data was then used to test the classifier. The Maximum Likelihood Estimation (MLE) method was first used to attempt to estimate the parameters (**μ**, and **σ)** of a Gaussian distribution.

Methods and Technical Approach

In order to facilitate the use of the discriminate functions it was necessary to develop a script in Python. This script was written in Python version 3.6.2, using the PyCharm development environment. The script developed to accomplish this task employed the use of several open source Python packages from the *SciPy* library (Jones et al., 2014). The *Pandas* (McKinney, 2010) package was used primarily for the initial data loading and conversation from the provided space delimited files, and the *NumPy* (Oliphant, 2006) package was used as the primary tool for data management and processing.

Maximum likelihood estimation of parameters

In order to use a parametric classification method the parameters must be estimated from the training data. These parameters can then be used to classify the samples in the testing dataset. The parameters of the training set were determined for each class in the training set. The value μ is the mean vector of the variables for each class and is calculated by adding all the samples of a variable and dividing by the total number of samples:

$$\mu_i = \frac{1}{\mathrm{n}} \sum_{i=1}^{n} x_{\mathrm{i}}$$

The μ vector is a column vector containing the mean of each dimension for each class and shown below:

$$\mu_0 = \begin{bmatrix} -0.22147024 \\ 0.32575494 \end{bmatrix} \qquad \mu_1 = \begin{bmatrix} 0.07595431 \\ 0.68296891 \end{bmatrix}$$

The next parameter is the covariance matrix, which is a d by d dimensional matrix where the diagonal values are the variance of the data which describes the way that the data is distributed relative to each axis. If the diagonal elements of a covariance matrix are equal (and the off diagonal elements are equal to zero), the distribution will be hyperspherical around the mean. If the diagonal elements are not equal the distribution will be skewed in the corresponding direction and will be hyperellipsoidal. The off-diagonal elements of the covariance matrix are the covariance of the distribution, and are equal to zero if the variables are totally independent of each other. If the covariance values are not equal to zero the distribution is skewed diagonally from the axis directions either in the positive direction if the covariance if positive, or the negative direction if the covariance values are negative. The covariance matrices for each class were determined using the unbiased equation:

$$\Sigma_{\mathrm{i}} = \frac{1}{\mathrm{n}-1} \sum_{i=1}^{n} [(x_{\mathrm{i}} - \mu_i)^{\mathrm{T}} (x_{\mathrm{i}} - \mu_i)]$$

The covariance matrices for each class were determined for each class and shown below:

$$\Sigma_0 = \begin{bmatrix} 0.27680955 & 0.01122866 \\ 0.01122866 & 0.03611906 \end{bmatrix} \qquad \Sigma_1 = \begin{bmatrix} 0.15974786 & -0.01557501 \\ -0.01557501 & 0.02995842 \end{bmatrix}$$

Once the parameters of each class were determined from the training data a discriminant function was then used classify the testing dataset. In order to use these functions some assumptions were required to be made about the distribution of the testing dataset. These assumptions are what differentiate the following different cases:

Case I (Minimum Euclidian Distance)

The Euclidean distance is the straight linear distance between the mean of a distribution and some point, measured through Euclidean space.

Assumptions:

(1) The features are statistically independent (off-diagonal elements of the covariance matrix (covariance) are equal to zero),
(2) The features have the same variance ($\sigma_{11} = \sigma_{22}$) and consequently, geometrically, the samples fall in equal-size hyper-spherical clusters.
(3) The decision boundary is the hyperplane of d-1 dimension.

Consequently the Euclidean distance for each testing sample is calculated using only the distance from the mean.

$$\mathrm{d} = \left|\left|\mathrm{x} - \mu\right|\right| = \sqrt{\sum_{i=1}^{n} (\mathrm{x_i} - \mu_i)^2}$$

This is the basis of the discriminant function for class 1:

$$g_i(x) = \frac{\left|\left|x - \mu_i\right|\right|^2}{2\sigma^2} + \ln P(\omega_i)$$

Case II (Minimum Mahalanobis Distance)

The Mahalanobis distance is also the distance between the mean of a distribution and a point, but it is measured through a coordinate system that is determined by the variance and covariance of the data. The Mahalanobis distance along the axis of the highest variance will be smaller than the Euclidian distance in Euclidian space across the same area.

Assumptions:

(1) The covariance matrices for all the classes are identical but not a scalar of identity matrix.
(2) Geometrically, the samples fall in hyper-ellipsoidal clusters
(3) Decision boundary is the hyperplane of d-1 dimension

$$d = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

This is the basis of the discriminant function for class 2:

$$g_i(x) = -\frac{1}{2}(x_\mathrm{i} - \mu_i)^\mathrm{T} \Sigma_\mathrm{i} (x_\mathrm{i} - \mu_i) + \ln P(\omega_i)$$

Case III (Maximum A-Priori Probability)

The Maximum A-Priori Probability is the most complex as it assumes that the covariance matrices are different from each category. This will yield a quadratic function in contrast to Case I and Case II which can be expressed as linear.

Assumptions:

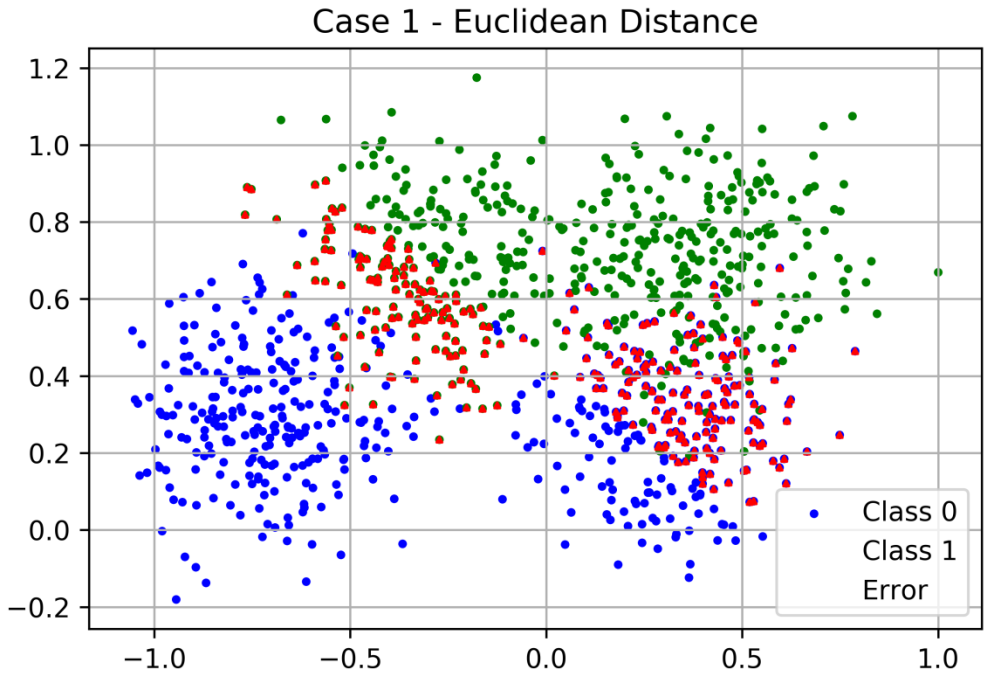
(1) The covariance matrices are different from each category
(2) Decision boundary is hyperquadratic for d-2 Gaussian

$$g_i(x) = -\frac{1}{2}(x_\mathrm{i} - \mu_i)^\mathrm{T} \Sigma_\mathrm{i} (x_\mathrm{i} - \mu_i) - \frac{1}{2}\ln|\Sigma_\mathrm{i}| + \ln P(\omega_i)$$

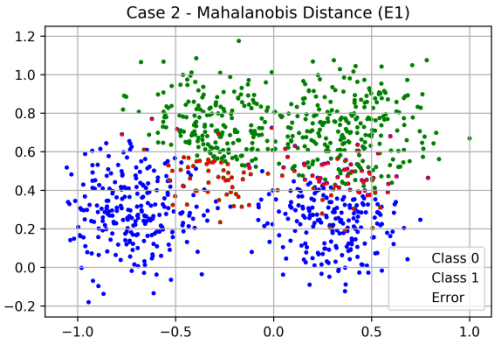
A function was defined for each of these cases in python. A second function was also created for evaluating the accuracy of each classifier and produce a plot showing the locations of the data points in the testing set. The incorrectly classified points were indicated.

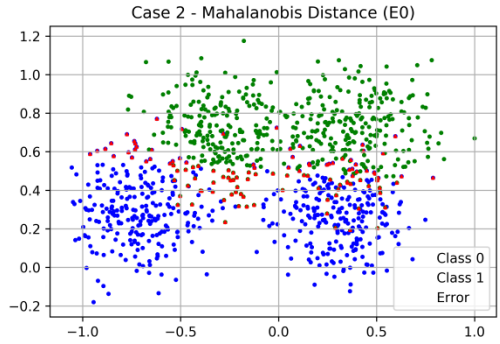
Experiments and Results

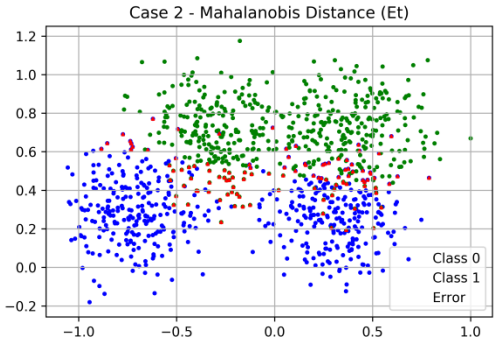
The training data set was used to train the classifiers for each discriminant function case (see included Jupiter Notebook for the complete code). The minimum Euclidean distance classifier Case 1 was the lowest in terms of accuracy:



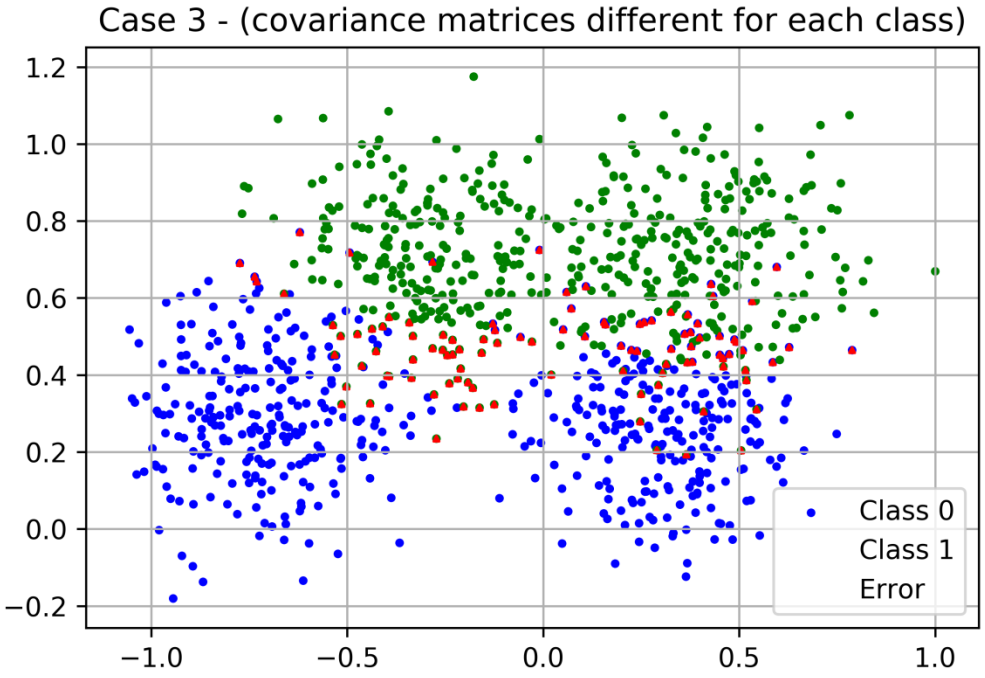
The classifier for Case 2 (Mahalanobis Distance) takes the variance of the data into account. This proved to perform much better than simply using the Euclidean Distance in case 1. This case assumes that the covariance matrix for each class are equal, and the results obtained by using the covariance matrices for each class separately as well as the total samples are shown in figure 2.



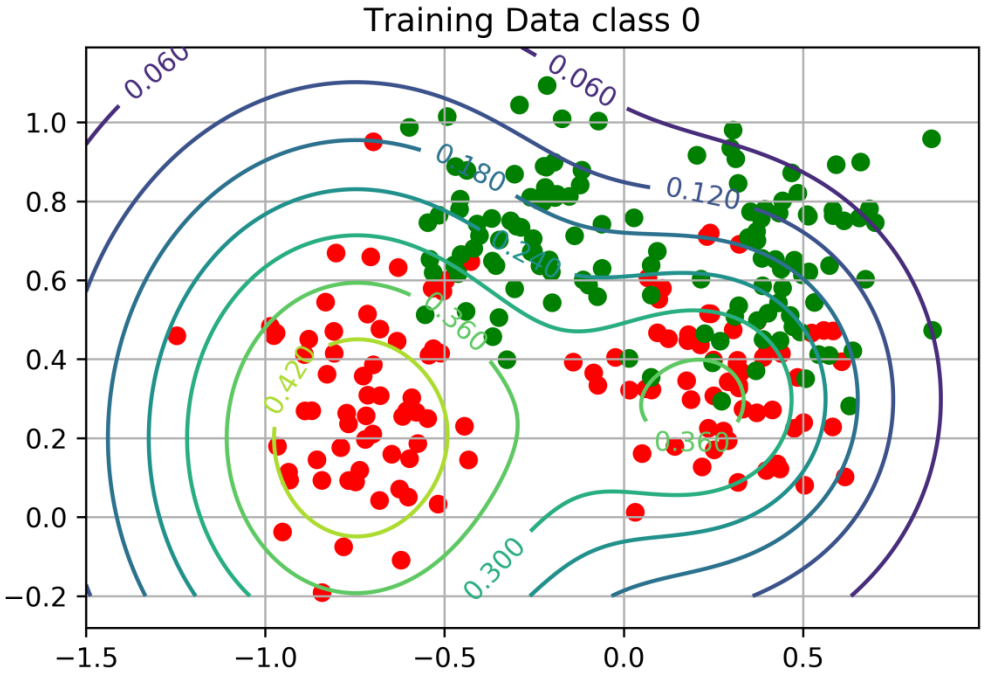


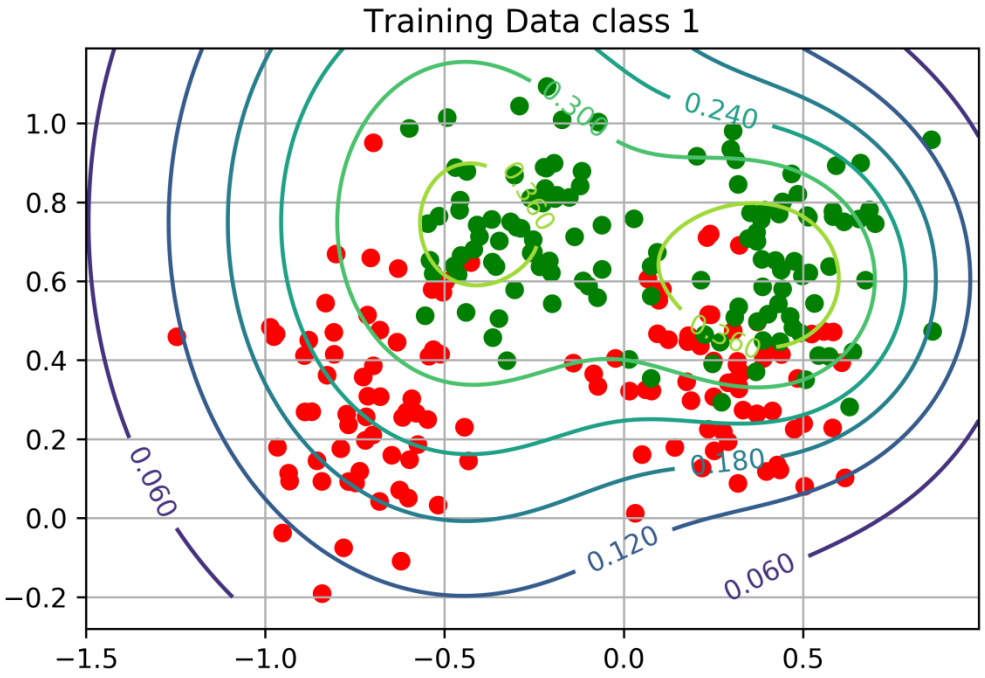


Finally the classifier for case 3 yielded a very slightly higher accuracy than case 2.

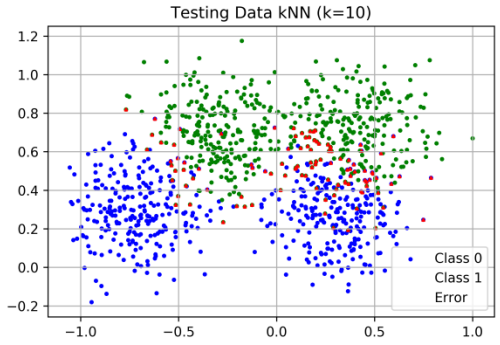


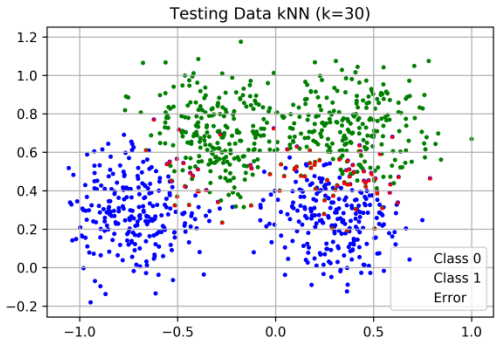
A bimodal probability density function was generated for each class of the training data set using the python code in the appendix. The results of this process are shown below in 2d and 3d plots:

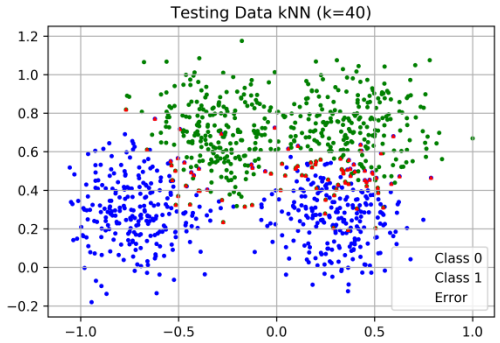


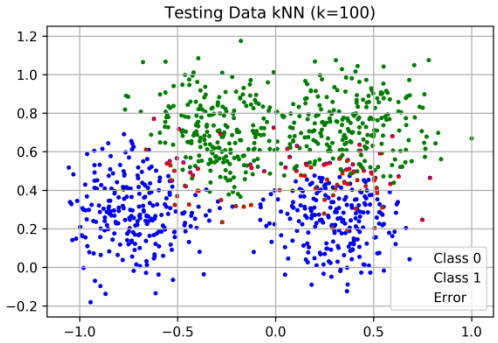


Results of the kNN classifier from project 2.









Classifier performance

Each of the classifiers used in this project were compared in terms of their accuracy and run time. The performance of each classifier are shown in Table 1a.

	Classifier	Classification Accuracy	Time (s)
One-Modal	Case 1 (Euclidean)	71.3%	0.007
	Case 2 (Mahalanobis)	Σt (89.2%) Σ0 (88.5 %) Σ1 (88.2%)	0.086 0.127 0.089
	Case 3 (MAP)	89.1%	0.117
Bi-Modal	Case 3 (MAP)		
	kNN	91.4% (k=30)	6.855

k=	Accuracy
1	85.00%
2	84.70%
3	86.60%
4	85.30%
5	87.00%
6	86.10%
7	88.90%
8	87.30%
9	88.80%
10	89.30%

Discussion

The results of this project illustrate the differences in the effectiveness of several classifiers. All three methods of determining these boundaries are assuming a one-modal Gaussian distribution. Since the actual distribution is bi-modal, the example where the values for parameters (**μ**, and **σ)** are estimated for bi-modal distributions of the points may be more accurate.

One thing that was interesting is the processing time for the discriminant functions compared to the kNN classifier from project two, which was an order of magnitude slower. This illustrates just how much more computationally intensive kNN is compared to using a discriminant function.

References

B.D. Ripley, 1996, Datasets used in the book Pattern Recognition and Neural Networks by B.D. Ripley (1996):, http://www.stats.ox.ac.uk/pub/PRNN/ (accessed January 2019).

Jones, E., Oliphant, T., and Peterson, P., 2014, {SciPy}: open source scientific tools for {Python}

McKinney, W., 2010, Data structures for statistical computing in python, *in* Proceedings of the 9th Python in Science Conference, Austin, TX, v. 445, p. 51–56.

Oliphant, T.E., 2006, A guide to NumPy: Trelgol Publishing USA, v. 1.

Appendix

(please refer to the Jupyter Notebook file included with this document for all code and step by step instructions)