# CALIFORNIA STATE UNIVERSITY, LONG BEACH

College of Engineering
Department of Computer Engineering and Computer Science
Dr. Thinh V. Nguyen
Spring 2008
**CECS-550/650:** Pattern Recognition

PROJECT 1

Name: ____Foss, Shannon_____

Last, First

- **Dates:** Date assigned: Thursday February 14, 2008. Date due: Wednesday March 6, 2008. Late submissions will receive penalty at 10% per day. This project is worth 30% of the project grade.

- **Objectives**: The objectives of this project includes: (1) to familiarize students with the basic linear classifiers, (2) to perform simulation of various types of classifiers.

- **Project Description:**

  A) Write a computer program to:

  1) generate a set S of N random 2-D data having a 2-D Gaussian distribution with specified mean vector $\boldsymbol{\mu}$ and diagonal covariance matrix $\boldsymbol{\Sigma}$. The set S has two classes $\omega_1$ and $\omega_2$. The following numbers of N should be used: 20, 1000, and 10000. Select values of as appropriate (e.g., non-overlapping, lightly overlapped distributions, heavily overlapped distribution).
  2) randomly divide the set S into two subsets having equal numbers of samples: a subset P used as a training set, and a subset Q used as a test set. Q consists of feature vectors to be classified.
  3) implement the following classifiers: general Bayes classifier, minimum distance classifier (MDC), nearest neighbor (NN) rule (k=1), kNN (k=3, 5, 7), and the perceptron;
  4) compute the classification errors for the above classifiers. Express the error as a percentage; and
  5) estimate execution time for each classifier for large Ns.

  B) Analyze the result. Provide tables or plot curves to show comparisons. Discuss any unexpected results.

  C) Additional operations may be used for extra credit. These may include: (a) extension to higher dimensions and more classes, (b) non-diagonal covariance matrix; (c) loss matrix and likelihood ratio test; and (d) non Gaussian distribution.

- **Useful links**: The following links may be useful.

http://www.dspguru.com/howto/tech/wgn2.htm
http://www.taygeta.com/random/gaussian.html
http://www.taygeta.com/random/weibull.xml
http://www-math.mit.edu/~spielman/ECC/gauss.html
http://helpdesk.graphpad.com/faq/viewfaq.cfm?faq=966 (Generate random Gaussian distribution using Excel)

END OF DOCUMENT

Shannon Foss
March 11, 2008
CECS 550
Project 1

# Pattern Recognition:
# Classifiers

In this project we attempted to implement several different types of classifiers to determine which class a randomly generated point belongs to.

## Introduction

Starting with a randomly generated set of numbers, the classifier algorithms will attempt to correctly categorize the test data into its class after first been trained on similarly generated training data. Depending on the clustering of the generated data, the problem may be very simple or extremely difficult. If the classes of data are already well segregated, it becomes trivial for a classifier to distinguish between them. However, if the classes overlap each other, determining which point belongs to what class can become quite tricky.

## Training and Testing Set Generation

To create a 2-D Gaussian Distribution we will use the Box-Muller Transform. The Box-Muller Transform is a method of generating pairs of independent standard normally distributed random numbers, given a source of uniformly distributed random numbers between 0 and 1. Using the polar coordinate form of the equation:

$$z_0 = u \cdot \sqrt{\frac{-2\ln s}{s}} \quad \text{and} \quad z_1 = v \cdot \sqrt{\frac{-2\ln s}{s}},$$

**Equation 1**: Box-Muller Transform

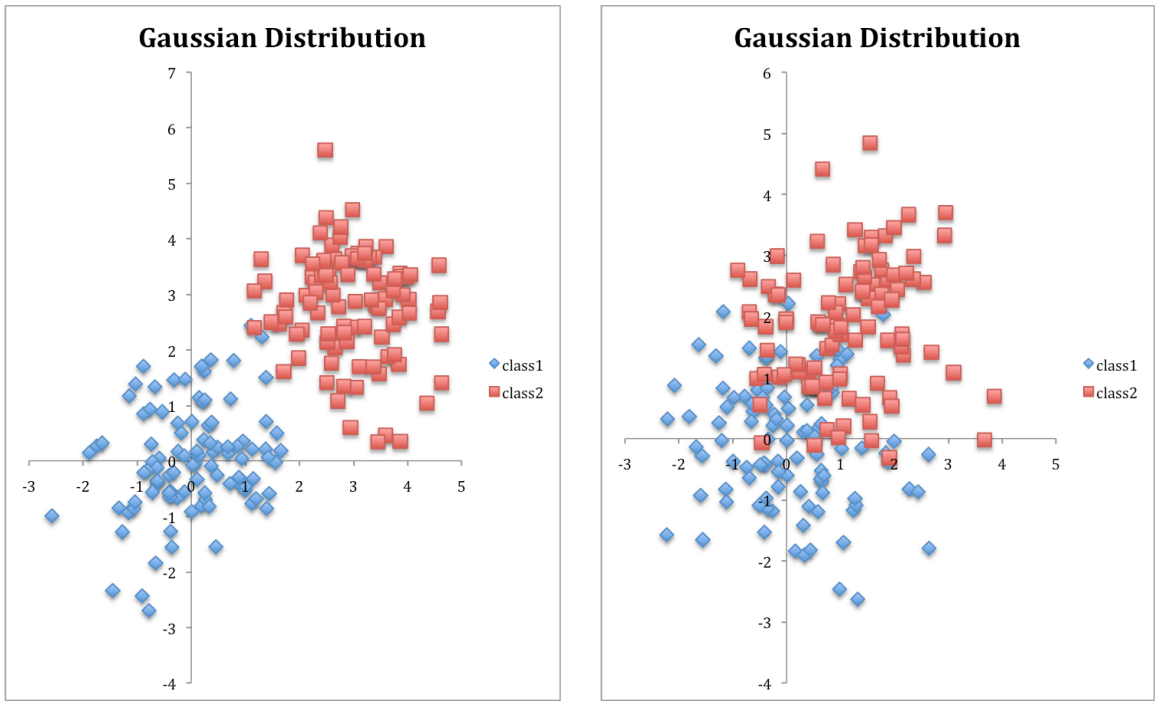
we may generate random points given uniformly distributed random numbers u and v ( $s=u^2+v^2$). To change the location of the mean, we simply multiply the x-component of the mean by $z_0$ and the y-component by $z_1$. Also, to change the standard deviation, we add the x-component of the standard deviation to $z_0$ and the y-component to $z_1$.

Using this information we may create sets of points classified according to the mean. By changing the locations of the means and magnitude of the standard deviation, we can create overlapping classes (Figure 1).

Overlapping classes makes the problem of classifying a point more difficult in that the classifiers are less able to distinguish which class a point could belong to when they lie within the overlapping area. Different strategies must be employed to help the classifiers distinguish between one set or the other.

There are three different types of overlap used for this project:

1. No overlap - $\omega_1=(0,0)$, $\sigma_1=(1,1)$ and $\omega_2=(4,4)$, $\sigma_2=(1,1)$.
2. Moderate overlap - $\omega_1=(0,0)$, $\sigma_1=(1,1)$ and $\omega_2=(2,3)$, $\sigma_2=(1,1)$.
3. Heavy overlap - $\omega_1=(0,0)$, $\sigma_1=(1,1)$ and $\omega_2=(1,1)$, $\sigma_2=(1,1)$.




(a) (b)

**Figure 1**: (a) Two 2-D Gaussian Distributions, with centers c1:(0,0) and c2:(3,3) and standard deviation of 1.0, demonstrating little to no overlap. (b) Two 2-D Gaussian Distributions, with centers c1:(0,0) and c2:(1,2) and a standard deviation of 1.0, demonstrating a great deal of overlap.

A training and a testing set may now be created from these classified points. A training set is used to teach the classifying algorithm how the classes are segregated. The testing set is used to determine how well the points were classified. The set is divvied up randomly between the training set and testing set and any of the classifying algorithms may then be run on them.

There are three different set sizes used in this project:

1. Small – size = 20
2. Medium – size = 1000
3. Large – size = 10000

**Bayesian Classifier**

The Bayesian Classifier is used to determine the class of a point based on the mean and standard deviation of the class you are comparing it to. You calculate $g_i(\mathbf{x})$ for each class in your set using the following equation:

$$g_i(\bar{x}) = -\frac{1}{2}(\bar{x} - \bar{\mu}_i)^T \Sigma_i^{-1}(\bar{x} - \bar{\mu}_i) + \ln P(\omega_i) + c_i$$

**Equation 2**: Bayesian Discriminate Function

After calculating the Bayesian Discriminate Function for each class, you classify **x** according to whichever g was the greatest. If $g_i(\mathbf{x}) > g_j(\mathbf{x})$ then the point belongs to class i.

Once all of the points in the testing data have been classified, you compare the results with the actual classifications for the test set and calculate the error.

| Results | Overlap = None | Overlap = Moderate | Overlap = Heavy |
|---|---|---|---|
| Size = 20 | 100% matches | 95 – 100% | 65 – 95% |
| Size = 1000 | 99.3 – 100 % matches | 95.4 – 97.9% | 73.9 – 77.5% |
| Size = 10000 | 99.63 – 99.87% matches | 96.11 – 96.87% | 75.22 – 77.32% |

**Table 1**: Results of Bayesian Classifier

The table shows that the Bayesian Classifier did very well for small to moderate overlap, heavier overlap, however, did not classify as well. The Bayesian Classifier ran very quickly, often less than a second even for the largest size.

**Minimum Distance Classifier**

The Minimum Distance Classifier is a very simple algorithm. It is the Euclidean Distance from the point you are testing to the mean of the class you are examining.

$$d_{\in} = \left\| \bar{x} - \bar{\mu}_i \right\|$$

**Equation 3**: Euclidean Distance

Compare the distances from that point to all of the class means, the shortest distance found is the one that gets assigned. Then repeat the process for all of the test points until they are all classified.

Evaluate the function by comparing the actual classifications to the ones found.

| Results | Overlap = None | Overlap = Moderate | Overlap = Heavy |
|---|---|---|---|
| Size = 20 | 100% | 85 – 100% | 60 – 100% |
| Size = 1000 | 99.6 – 99.9% | 95.3 – 97.2% | 74.3 – 78% |
| Size = 10000 | 99.72 – 99.83% | 96.15 – 96.87% | 75.16 – 77.26% |

**Table 2**: Results of Minimum Distance Classifier

The table shows that the Minimum Distance Classifier did very well on the low to moderate overlap, but much like the Bayesian, it dropped out with the heavier overlap. Also like the Bayesian Classifier, the Minimum Distance Classifier ran very quickly, with hardly a pause on the largest sizes.

**Nearest Neighbor**

The Nearest Neighbor Method finds the k nearest classified (training) points to the point being evaluated (test point), with k being a given number greater than one. The algorithm uses the Euclidean Distance (Equation 3) to find the k shortest paths from the point being evaluated. It then determines the classification of each of the k paths and counts each class and the class with the highest count is assigned to that point.

Once all of the test points are classified, the algorithm is evaluated by comparing the actual classifications to the ones that were calculated.

| Results k = 1 | Overlap = None | Overlap = Moderate | Overlap = Heavy |
|---|---|---|---|
| Size = 20 | 95 – 100% | 75 – 100% | 50 – 70% |
| Size = 1000 | 99.6 – 99.8% | 93.5 – 96% | 65.9 – 70.5% |
| Size = 10000 | 99.6 – 99.68 | 95.8 – 96.2% | 67.25 – 67.89% |

**Table 3**: Results of k=1 Nearest Neighbor

| Results k = 3 | Overlap = None | Overlap = Moderate | Overlap = Heavy |
|---|---|---|---|
| Size = 20 | 100% | 85 – 100% | 45 – 75% |
| Size = 1000 | 99.3 – 99.8% | 94.8 – 97% | 68.3 – 71% |
| Size = 10000 | 99.71 – 99.83% | 95.92 – 96.22% | 71.05 – 72.06% |

**Table 4**: Results of k=3 Nearest Neighbor

| Results k = 5 | Overlap = None | Overlap = Moderate | Overlap = Heavy |
|---|---|---|---|
| Size = 20 | 100% | 80 – 100% | 55 – 85% |
| Size = 1000 | 99.4 – 99.9 | 95.5 – 96.7% | 70.9 – 74.7% |
| Size = 10000 | 99.75 – 99.84% | 95.99 – 96.3% | 72.97 – 73.09% |

**Table 5**: Results of k=5 Nearest Neighbor

| Results k = 7 | Overlap = None | Overlap = Moderate | Overlap = Heavy |
|---|---|---|---|
| Size = 20 | 100% | 85 – 95% | 40 – 80% |
| Size = 1000 | 99.2 – 99.9% | 95 – 97.1% | 72.2 – 77.7% |
| Size = 10000 | 99.63 – 99.83% | 95.94 – 96.47% | 72.9 – 74.59% |

**Table 6**: Results of k=7 Nearest Neighbor

The table shows that the k nearest neighbors did very well on the lowest amount of overlap, but started dropping out on the moderate overlap and a great deal on the heavy overlap. The higher value of k often helped to bring about higher matches, but not a great deal more. The timing on k Nearest Neighbor was often very slow, and could take up to 20-25 seconds to complete the largest size sets.

**Perceptron**

The perceptron algorithm consists of several rules:

$$w^{*T}x > 0 \quad \forall x \in \omega_1$$
$$w^{*T}x < 0 \quad \forall x \in \omega_2$$

**Equation 4**: Assuming that the classes $\omega_1$ and $\omega_2$ are linearly separable, then there exists a hyperplane $w^{*T}x = 0$ which divides the two classes.

$$w(t+1) = w(t) + \rho x_{(t)} \text{ if } x_{(t)} \in \omega_1 \text{ and } \omega^T(t)x_{(t)} \le 0$$
$$w(t+1) = w(t) - \rho x_{(t)} \text{ if } x_{(t)} \in \omega_2 \text{ and } \omega^T(t)x_{(t)} \ge 0$$
$$w(t+1) = w(t) \text{ otherwise}$$

**Equation 5**: Reward and Punishment schemes to correctly converge to a weight that determines a hyperplane that fits the classes.

Applying these rules to the training set will determine a hyperplane that will correctly divide the two class sets.

| Results | Overlap = None | Overlap = Moderate | Overlap = Heavy |
|---|---|---|---|
| Size = 20 | 95 – 100% | 80 – 100% | 40 – 90% |
| Size = 1000 | 99.3 – 99.9% | 56.7 – 96.3% | 59.4 – 75.1% |
| Size = 10000 | 99.36 – 99.8% | 84.08 – 96.33% | 65.11 – 74.71% |

**Table 7**: Results of Perceptron

The table shows that the perceptron algorithm does very well for low to moderate overlap, but again, starts dropping out on the heavier overlaps, and its matches can vary widely. The timing on the perceptron algorithm was not as good as others, but it was better than the Nearest Neighbors. Perceptron often timed in between 5 and 10 seconds for the largest training sets.

**Hybrid**

I created and implemented a hybrid classifier that is a cross between the Nearest Neighbor Method and the Bayesian Classifier. This method uses Bayes' Theorem as a basis:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

**Equation 6**: Bayes' Theorem – Where P(A) is the prior probability of A, P(B) is the prior probability of B, P(B|A) is the conditional probability of B given A, and P(A|B) is the conditional probability of A given B.

First we find the probability that the new point is either in $\omega_1$ or $\omega_2$, which is the number of items in the class divided by the total number items, this gives us our prior probabilities. Now we need to determine a posterior probability for both of the classes. I decided to draw a box, based on the $\sigma$ size, around the point being classified. Somewhat like Nearest Neighbor, if there are any points from $\omega_1$ or $\omega_2$ in the box, we count them to determine the posterior probability of the point. Probabilities for both classes are determined, and the one that is greater is what is classified to the point.

This process is repeated for all of the testing points until they are all classified. The algorithm is evaluated by comparing the found classifications to the actual classifications.

| Results | Overlap = None | Overlap = Moderate | Overlap = Heavy |
|---|---|---|---|
| Size = 20 | 85 – 95% | 75 – 95% | 55 – 80% |
| Size = 1000 | 99.4 – 99.8% | 95.6 – 97% | 74.5 – 76.7% |
| Size = 10000 | 99.66 – 99.76% | 96.24 – 96.66% | 76.03 – 76.65% |

**Table 8**: Results of Hybrid Classifier

The table shows that, again, the classifier did very well on the low to moderate overlap, but dropped out on the heavier overlap. The timing on the Hybrid Classifier was quite fast and was comparable to the Bayesian Classifier.

**Conclusion**

It seems as though that the best algorithms were quick and had an overall higher match rate than the slower algorithms. Bayesian, Minimum Distance and the Hybrid were all comparable in their matches and in their timing. Perceptron had a good percentage of matches but it was slightly slower, whereas the k Nearest Neighbor was the slowest of them all, and did not generate any better match outcomes than the faster algorithms.