ECE 8527 Homework Number 2: Maximum Likelihood versus Bayesian Estimation

Andrew Powell

February 13, 2014

1 Introduction

The primary goals of Homework 2 are to apply and understand Maximum Likelihood Estimation, Bayesian Estimation, and Principal Component Analysis. Another goal of Homework 2 is to continue to fully understand Bayesian Decision Theory (also known as Maximum Likelihood Classification) by determining the theoretical probability of error and then comparing the theoretical result with an estimation generated from MATLAB simulation.

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(x)} \tag{1}$$

The ultimate goal of Maximum Likelihood Classification is to choose a particular class ω_i for a given feature vector $\mathbf{x}$. The underlining rule behind the Maximum Likelihood Classification is Bayes' Rule as shown in Equation 1. $\mathbf{x}$ is a d dimensional feature vector that is treated as a vector of d random variables and as the basic element of a data set D. The ω_i are the states or categories. The *priors* $P(\omega_i)$ are probabilities that represent prior knowledge about the data. $p(\mathbf{x}|\omega_i)$ is the *likelihood* (or class-conditional density) a certain feature, given the respective state. In the context of Maximum Likelihood and Bayesian Estimation, the $p(\mathbf{x}|\omega_i)$ have known distributions (e.g. the normal distribution) that approximate the likelihoods. The distributions' parameters (e.g. mean and variance) are unknown and each method of estimation has a different approach in determining those parameters from given data, although yielding similar parameters. The *posteriors* $P(\omega_i|\mathbf{x})$ allows for the classification of each feature vector from the data. The *p(x)* merely acts as a scalar to ensure $\sum_{i=1}^{N} P(\omega_i|\mathbf{x}) = 1$. The evidence is discarded for its lack of contribution in the classifier.

$$g_i(\mathbf{x}) = \ln(P(\omega_i|\mathbf{x})) = \ln(p(\mathbf{x}|\omega_i)) + \ln(P(\omega_i)) - \alpha \tag{2}$$

$$\text{If } g_1(\mathbf{x}) > g_2(\mathbf{x}) \text{ then choose } \omega_1\text{, otherwise choose } \omega_2 \tag{3}$$

$$g_i(\mathbf{x}) = -\frac{1}{2}\left[(\mathbf{x}-\mu_i)^t \sum\nolimits_i^{-1} (\mathbf{x}-\mu_i) + d\ln(2\pi) + \ln(|\sum\nolimits_i|)\right] + \ln(P(\omega_i)) \tag{4}$$

In Maximum Likelihood Classification, the classification is done through two or more discriminant functions $g_i(\mathbf{x})$, which are functions of the posteriors $p(\omega_i|x)$ as seen in Equation 2. The constant α is a function of the evidence from Equation 1 and is discarded. For the case in which only two possible states exist, the "Two-Category Case", the discriminant function shown in Equation 3 can be constructed. Since the distribution the most common in Homework 2 is the normal distribution, the likelihood seen in Equation 2 is substituted with the normal distribution $N(\mu, \sum)$ as shown in Equation 4. The MATLAB code developed in order to generate the results for Homework 2 implements the "normal" discriminant function in Equation 4 directly as an anonymous function and then utilizes the Maximum Likelihood classifier in Equation 3 to determine error rate.

Maximum Likelihood and Bayesian Estimation both consider the case in which the likelihood's distribution is not precisely known, however there is given data and a distribution that can potentially approximate the likelihood. In other words, what is a "reasonable" distribution that approximates $p(\mathbf{x}|\omega_i, D)$? D is the given data. The problem of concern is determining the parameter vector θ of the distribution that approximates the likelihood $p(\mathbf{x}|\omega_i, D)$. Maximum Likelihood Estimation tries to find the parameter vector $\hat{\theta}$ that maximizes the likelihood of a set of given data D, whereas Batesian estimation treats the parameter vector θas a random variable whose distribution ideally, with more and more data, is supposed to converge to a parameter vector $\hat{\theta}$.

$$\acute{\mathbf{x}} = \mathbf{A}^t(\mathbf{x} - \mu) \tag{5}$$

Principal Components Analysis basically "whitens" a set of data through a preprocess (i.e. prior to classification) that involves computing the covariance matrix $\sum$ and the mean vector μ for a set of data, building a matrix A from the covariance matrix's eigen values λ_i and vectors e_i, and then finally applying an operation on each feature vector of the data in order to complete the whitening process. "Whitened" indicates the data has been preprocessed in such a way that the preprocessed data set is now decorrelated and thus its covariance matrix $\sum$ is zero for the off-diagonal elements. The operation that transforms each feature vector $\mathbf{x}$ from the original data set to a preprocessed feature vector $\acute{\mathbf{x}}$, whose data set is decorrelated, is shown in Equation 5. $\acute{\mathbf{x}}$ is the preprocessed or transformed feature vector and the matrix $\mathbf{A}$ is a matrix whose columns are the eigen vectors $[e_1, ..., e_d]$. Each eigen vector e_i is determined from the corresponding eigen value λ_i, where the eigen values are sorted starting with the highest eigen value and then decreasing.

Please note all MATLAB source code developed to generate the following results is found in Section 7.1 in this Homework's Appendix.

2 Problem 1

$$\begin{aligned} \mu_1 &= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \mu_2 &= \begin{bmatrix} -1 \\ -1 \end{bmatrix} \end{aligned} \tag{6}$$

Figures 1 to Figure 5 show the results to Problem 1, for which two 2D gaussian probability density functions whose respective mean vectors μ_i are shown in Equation 6 are repeatedly generated so as to produce theoretical probability of errors as a function of the prior probability of class ω_1 and the covariance matrices $\sum_i$. Each figure individually plots the theoretical probability of error $P(error)$ against the prior $P(\omega_1)$. The difference in each figure is simply which covariance matrices were utilized in the generation of the probabilities and in the discriminant functions. Each figure contains 1 or 2 plots. Each plot includes a caption that reveals the covariance matrices applied to generate the respective plots.

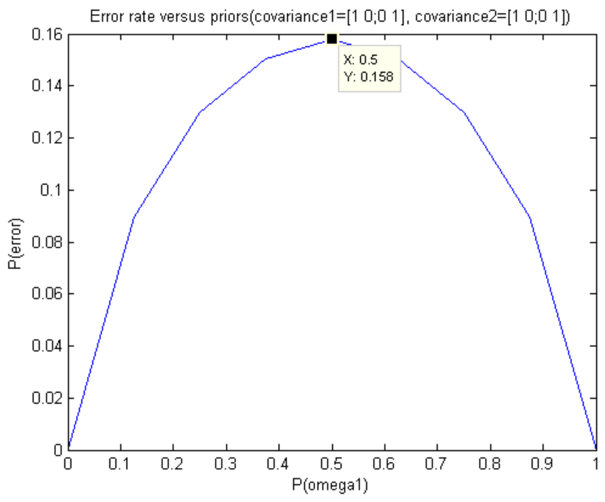
The data tips shown in each plot indicate where the maximum probability of error $P(error)$ is. Unsurprisingly, the highest point of error is always when the priors are equivalent, that is, $P(\omega_1) = P(\omega_2)$. Priors set equal indicate no useful information is gained from the priors and don't contribute to the overall posterior. The likelihoods thus are left to contribute to the classification. A closer examination of the each plot's extremities, when $P(\omega_1) = 0$ and $P(\omega_1) = 1$, reveals another unsurprising observation; $P(error)$ goes to zero at each plot's extremities, suggesting there is simply no way an error can be made if prior knowledge indicates only features from a single class will occur.

The theoretical probabilities of error are likely to have a certain degree of error, considering the theoretical calculations are done numerically with MATLAB. MATLAB, due to the nature of computers, are going to accumulate some degree of error because of a lack of precision. Another major issue are the two axis defined to generate "all" possible feature vectors for the probability density functions. The period between each point is set as 0.2 in the MATLAB source, but ideally the period should be small as possible to minimize quantization error (*this may or may not be the right word to use here, but I think you know what I mean, Dr. Picone*).

$$P(error) = p(\omega_1|x, x \in \omega_2) + p(\omega_2|x, x \in \omega_1); \tag{7}$$

Equation 7 demonstrates how the theoretical probability of error is calculated. What Equation 7 implies (*or is supposed to imply, rather*) is the probability of error is the sum of the probability of ω_1 being chosen when the feature vector $\mathbf{x}$ really came from ω_2 and the probability of ω_2 being chosen when the feature vector $\mathbf{x}$ really came from ω_1.

Figure 1



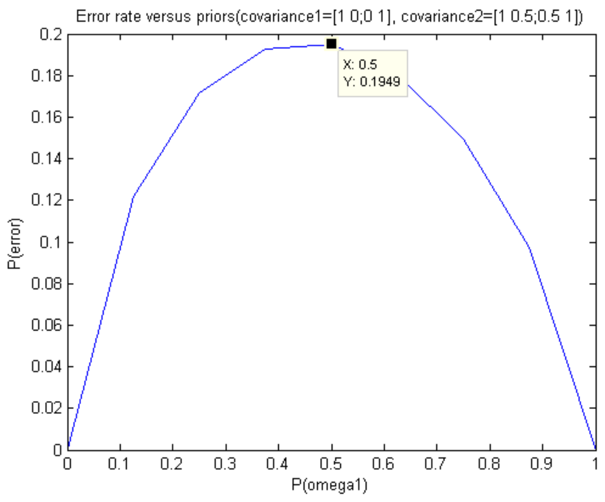
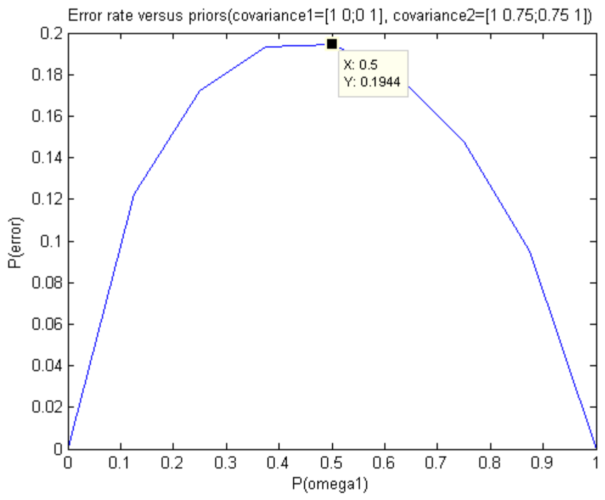


Figure 2



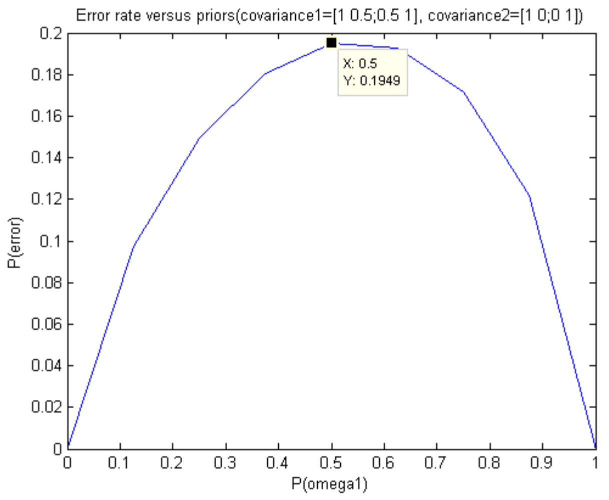
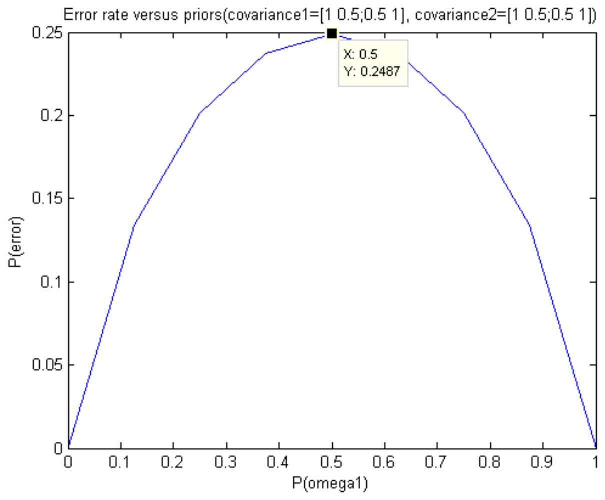


Figure 3



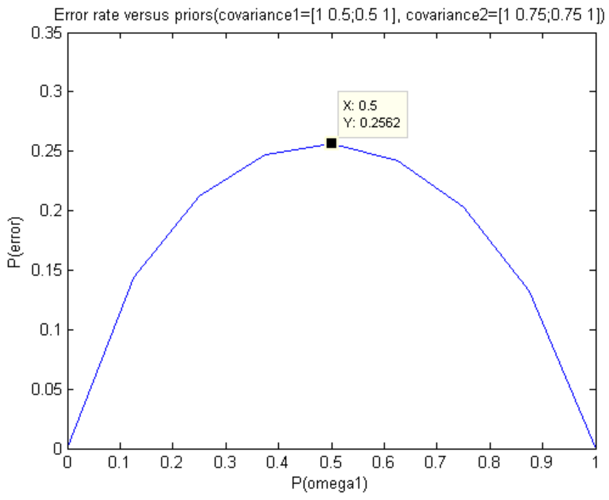
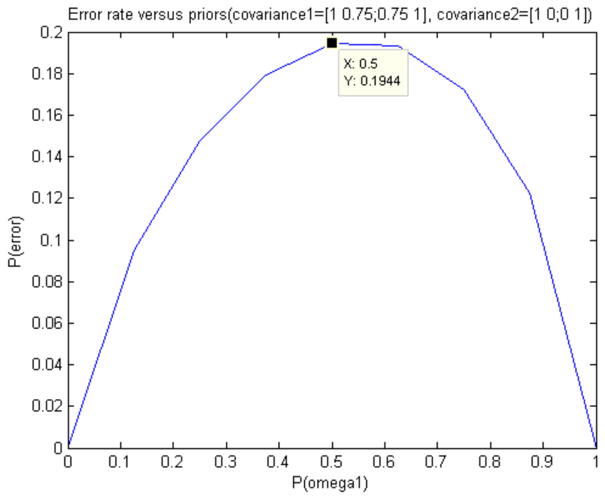


Figure 4



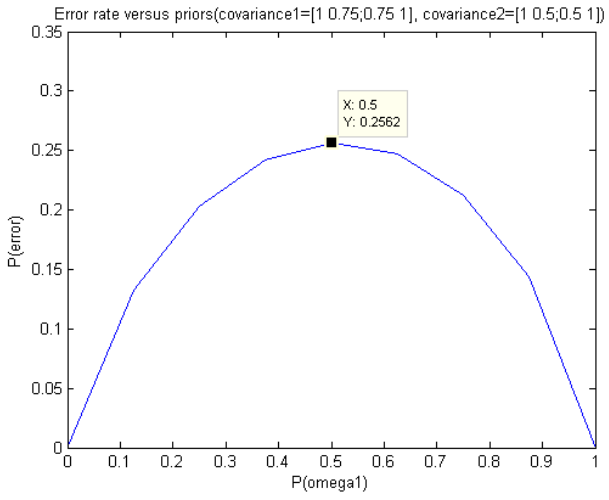
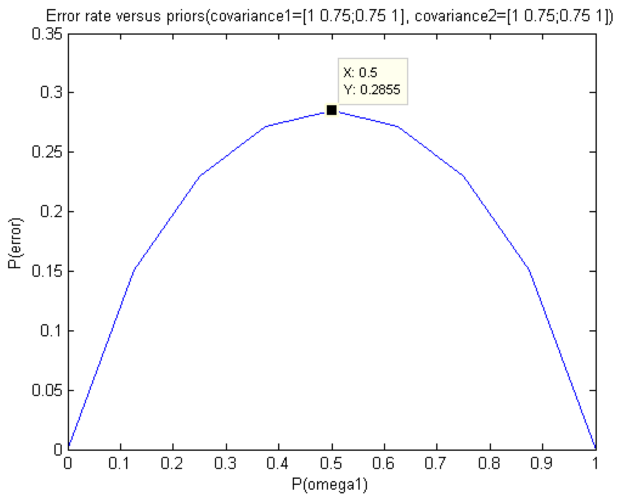


Figure 5



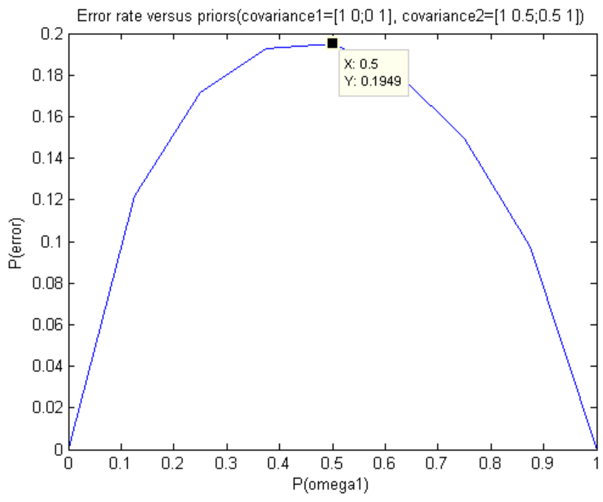
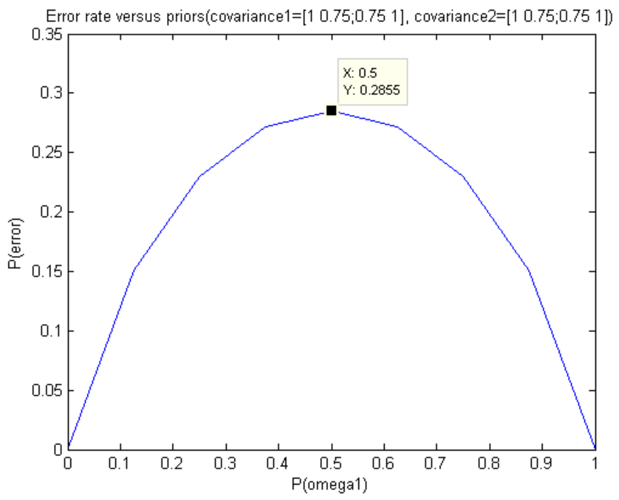


Figure 6



3 Problem 2

$$\sum\nolimits_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \sum\nolimits_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{8}$$

$$\sum\nolimits_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \sum\nolimits_2 = \begin{bmatrix} 1 & .5 \\ .5 & 1 \end{bmatrix} \tag{9}$$

In according to the instructions for Problem 2, a Maximum Likelihood Classifier is created and applied to randomly generated data from the classes described in Problem 1. The priors are set to equal each other, that is, $P(\omega_1) = P(\omega_2) = .5$. Due to the length of time to run simulations on MATLAB, the two cases show in Equations 8 and 9 are considered. The number of feature vectors generated for each class, n, are $1{\times}10^2$, $1{\times}10^3$, $1{\times}10^4$, and $1{\times}10^5$. The estimated probability of errors are calculate for each length n, utilizing the Maximum Likelihood Classifier constructed from Equations 3 and 4. The simulated results for Problem 2 are shown as Listing 1. The theoretical results are shown in the plots under Figure 1.

```
Problem 2

Data size: 100
Error rate: 0.2
Covariance1: [1 0;0 1]
Covariance2: [1 0;0 1]

Data size: 1000
Error rate: 0.146
Covariance1: [1 0;0 1]
Covariance2: [1 0;0 1]

Data size: 10000
Error rate: 0.1557
Covariance1: [1 0;0 1]
Covariance2: [1 0;0 1]

Data size: 100000
Error rate: 0.15916
Covariance1: [1 0;0 1]
Covariance2: [1 0;0 1]

Data size: 1000000
Error rate: 0.15702
Covariance1: [1 0;0 1]
Covariance2: [1 0;0 1]

Data size: 100
Error rate: 0.2
Covariance1: [1 0;0 1]
Covariance2: [1 0.5;0.5 1]

Data size: 1000
Error rate: 0.171
Covariance1: [1 0;0 1]
Covariance2: [1 0.5;0.5 1]

```

```
Data size: 10000
Error rate: 0.1874
Covariance1: [1 0;0 1]
Covariance2: [1 0.5;0.5 1]

Data size: 100000
Error rate: 0.1945
Covariance1: [1 0;0 1]
Covariance2: [1 0.5;0.5 1]

Data size: 1000000
Error rate: 0.19565
Covariance1: [1 0;0 1]
Covariance2: [1 0.5;0.5 1]
```

Listing 1: Maximum Likelihood Classification results generated in MATLAB

Unsurprisingly, as the number of the feature vectors are included, the simulated error rates tend to converge to the theoretical probability of errors as shown in Figure 1.

4 Problem 3

$$\mu_1 = \begin{bmatrix} -2 \\ -5 \end{bmatrix} \qquad \sum\nolimits_1 = \begin{bmatrix} 2 & 1.5 \\ 1.5 & 2.0 \end{bmatrix} \tag{10}$$

$$\mu_2 = \begin{bmatrix} 3 \\ 6 \end{bmatrix} \qquad \sum\nolimits_2 = \begin{bmatrix} 3 & -2 \\ -2 & 3 \end{bmatrix} \tag{11}$$

Two data sets are generated from two 2D gaussian distributions. The first data set for class ω_1 corresponds to the 2D gaussian distribution whose parameters are shown in Equation 10, whereas the second data set for class ω_2 corresponds to the 2D gaussian distribution whose parameters are shown in Equation 11. It is assumed the length n for each data set is 1×10^4 feature vectors $\mathbf{x}$ and both priors for Maximum Likelihood Classification are $P(\omega_1) = P(\omega_2) = .5$. The density function of the data generated is shown in Figure 8.

4.1 A

The error rate determined with a Maximum Likelihood Classifier implemented in MATLAB is shown in Listing 2.

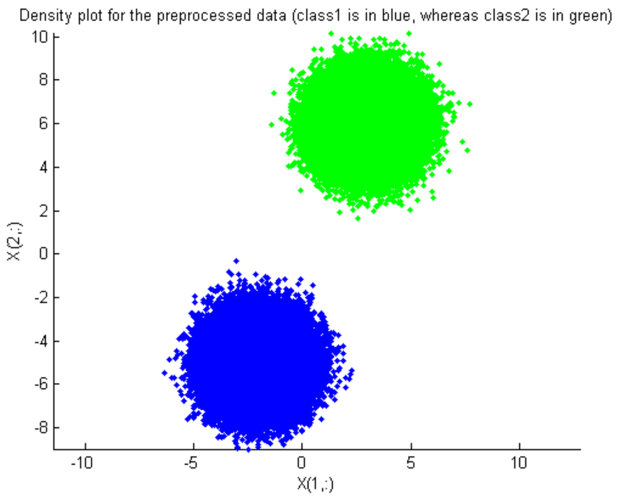
```
Problem 3

ML Classification:
Data size: 100000
Error rate: 2e-05 % basically zero
Mean1: [-2;-5]
Covariance1: [2 1.5;1.5 2]
Mean2: [3;6]
Covariance2: [3 -2;-2 3]
```

Listing 2: Maximum Likelihood Classification results generated in MATLAB

4.2 B and C

For each data set, Principal Component Analysis is performed by first computing the covariance matrix, determining the eigen vectors, concatenating the eigen vectors as column vectors in matrix A with each column corresponding to the next lowest eigen value, and then applying the operation as shown in Equation 5. The resulting preprocessed data set is then normalized such that its variances are equal to 1. Finally, the mean vectors μ are added back to each preprocessed feature vector $\acute{\mathbf{x}}$ in the preprocessed data set. The density plot for both preprocessed data sets are shown in Figure 7.

Figure 7: The density function of both data sets after whitening



The classes are chosen based on the Euclidean distances between each preprocessed feature vector $\acute{\mathbf{x}}$ and the mean vectors of each data set; the class whose mean vector μ is the shortest Euclidean distance away from the preprocessed feature vector $\acute{\mathbf{x}}$ is decided. The error rate is the ratio between the number of incorrect decisions and the total number of preprocessed feature vectors, i.e. $n = 1 \times 10^4$. The results are displayed in Listing 3. The error rate determined after Principal Component Analysis is nearly the same as the error rate determined from the Maximum Likelihood Classifier created for Part A.

```
Problem 3

PCA Classification?:
Data size: 100000
Error rate: 0
Mean1: [-1.9964557135527;-4.99506275580752]
Covariance1: [0.999999999999997 2.10411466895766e-14;2.10411466895766e-14 1]
Mean2: [2.99828728695589;6.00268204090308]
Covariance2: [0.999999999999986 -7.12889747234158e-16;-7.12889747234158e-16 1.00000]
```

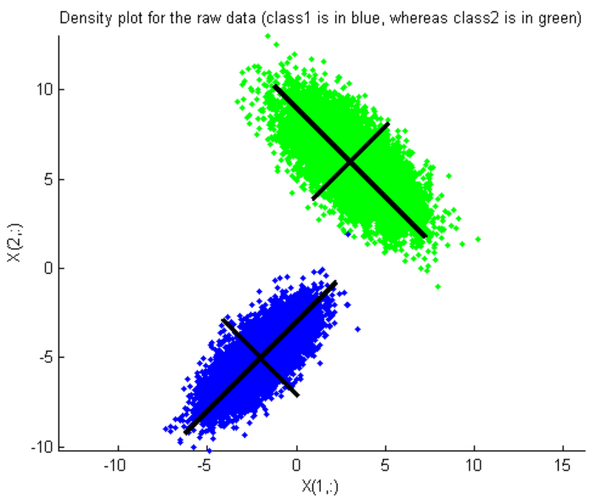
Listing 3: Principal Component Analysis results generated in MATLAB

4.3 D

Figure 8 contains a plot of raw data's density function. The black lines are the eigen vectors, the *Principal Components*, computed from performing the Principal Component Analysis on each data set. The mean vectors estimated from each raw data set are added to the eigen vector in order to shift them over their respective data sets. The actual eigen vectors are shown in Listing

4. Unfortunately, due to limited time to properly finish Homework 2, the support regions are not shown in the figures. Even without the support regions, the shape of each data set's distribution is still easily observable. For each data set, the eigen vectors, or the Principal Components, are aligned in the direction with the most variance.

Figure 8: The density function of both data sets prior (the black lines are the eigen vectors scaled by an arbitrary amount and then summed with the means)



```
Eigen Matrices (columns are the Principal Components):
A1: [-0.70763 0.70659;-0.70659 -0.70763]
A2: [-0.70911 -0.70509;0.70509 -0.70911]
```

Listing 4: Eigen Matrices A

5 Problem 4

For Problem 4, Bayesian Estimation is compared with Maximum Likelihood Estimation by generating a series of 1D normally distributed data sets whose sizes are n features (or data points) and then finding an estimated mean (i.e. $\hat{\mu}$ for Maximum Likelihood Estimation and μ_n for Bayesian Estimation) that converges towards the true mean μ. It is given the number of data points n should range from 100 to 100×10^3 and, for Bayesian Estimation, the probability density of the mean μ for each data set, $p(\mu|D)$ where D represents each data set, is normally distributed. Considering the example from the slides and the book do not consider the case in which the variance σ^2 is unknown for Bayesian Estimation, the variance σ^2 is assumed known and thus chosen such that the simulation produces the most "interesting" results. What is considered "interesting" is later made clear.

$$\begin{aligned} \mu &= 0 \\ \sigma^2 &= 10 \times 10^9 \\ n &= \left[100 : 5 \times 10^3 : (100 \times 10^3 + 100)\right] \end{aligned} \tag{12}$$

Equation 12 displays the parameters for the generation of the data. The mean μ is selected as 0 in order to make interpretation of the results easier. The variance σ^2 is chosen as a large value in order to see the difference between the Bayesian and Maximum Likelihood Estimation;

the larger the true variance σ^2 and the "uncertainty" σ_0^2 is, the larger difference is between the two forms of estimation. n simpl shows how the length of each data set increases a little bit above the value 100×10^3. Please note n from Equation 12 is in MATLAB's notation for an incrementing vector.

$$\begin{aligned} \mu_0 &= 0 \\ \sigma_0^2 &= 10 \times 10^9 \end{aligned} \tag{13}$$

Equation 13 shows the parameters for $p(\mu|D)$ approximated as $N(\mu_0, \sigma_0^2)$, which is the shorthand notation for the normal distribution. Since μ is considered a random variable in Bayesian Estimation, $p(\mu|D)$ is the mean's probability density function with respect to each data set D. μ_0 is prior knowledge of the true mean μ and is assumed to equal 0. It is also assumed the degree of "uncertainty", represented as the variance σ_0^2 for $p(\mu|D)$, is also known. Again, the reason why the "uncertainty" σ_0^2 is chosen as such a large value is to force a larger difference between the Maximum Likelihood and Bayesian estimated means.

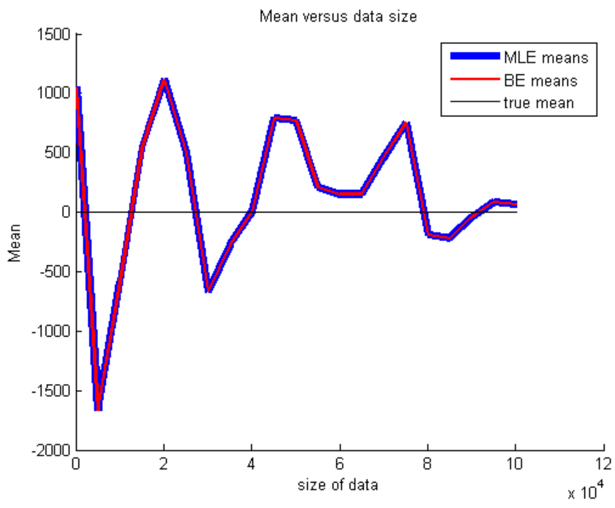
$$\begin{aligned} \mu_n &= (\frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2})\bar{x_n} + \frac{\sigma^2}{n\sigma_0^2 + \sigma^2}\mu_0 \\ \bar{x_n} &= \frac{1}{n}\sum_{k=1}^{n} x_k \end{aligned} \tag{14}$$

$$\sigma_n^2 = \frac{\sigma_0^2\sigma^2}{n\sigma_0^2 + \sigma} \tag{15}$$

$$\hat{x} = \frac{1}{n}\sum_{k=1}^{n} x_k \tag{16}$$

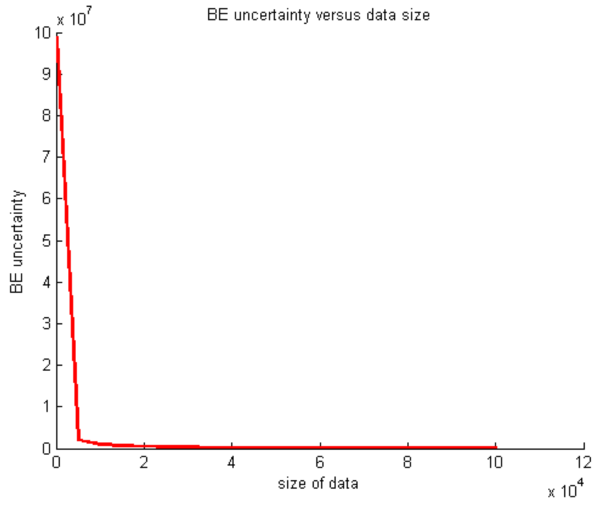
Implemented directly in the MATLAB source are the Equations 14, 15, and 16. Equation 14 shows how the Bayesian estimated mean μ_n is determined. Equation 15 determines the "uncertainty" of the Bayesian estimate. The derivation of 14 results from the relation $p(\mu|D) = \frac{p(D|u)p(u)}{p(D)} \sim N(\mu_n, \sigma_n^2)$, where $p(D|\mu)$ is approximately $\prod_{k=1}^{n} p(x_k|\mu) \sim \prod_{k=1}^{n} N(\mu, \sigma^2)$ and the prior $p(\mu)$ is approximately $N(\mu_0, \sigma_0^2)$. Equation 16 determines the estimated mean $\hat{\mu}$, which is the mean μ that maximizes the probability density function $p(\mu|D)$ as part of Maximum Likelihood Estimation.

Figure 9



The plot in Figure 9 is generated from the MATLAB simulation and visually illustrates how much the estimated means vary with respect to the amount of data produced. The plot appears to imply the estimated means are always the same and converge to the true mean μ.

Figure 10



The plot in Figure 10 is also of importance, as it visually shows how the "uncertainty" diminishes as the data set's size is increased.

Data Sizes n	MLE Means	BE Means	|MLE Means − BE Means|
100.0	1052.5	1042.1	10.421
5100.0	−1663.6	−1663.3	0.32614
10100.0	−583.98	−583.92	0.057814
15100.0	544.58	544.55	0.036063
20100.0	1118.2	1118.2	0.05563
25100.0	502.55	502.53	0.020021
30100.0	−667.74	−667.72	0.022183
35100.0	−275.1	−275.09	0.0078373
40100.0	2.5359	2.5358	0.000063237
45100.0	792.05	792.03	0.017562
50100.0	768.91	768.89	0.015347
55100.0	208.16	208.15	0.0037778
60100.0	149.37	149.37	0.0024853
65100.0	153.47	153.47	0.0023574
70100.0	459.47	459.46	0.0065543
75100.0	751.06	751.05	0.010001
80100.0	−188.77	−188.77	0.0023567
85100.0	−220.67	−220.67	0.002593
90100.0	−41.132	−41.132	0.00045651
95100.0	85.552	85.551	0.00089959
100100.0	62.317	62.317	0.00062254

(17)

The table shown in Figure 17 (*I think*) reveals the most "interesting" information regarding how the estimated means change as the data sets increase in size, and how the "uncertainty" in the data's expected value (i.e. the variance σ^2) and in the mean (i.e. the σ_0^2) impact the Bayesian estimated mean. If σ^2 and σ_0^2 are *both* increased to incredibly large values, the difference between the two estimates grow larger and larger. But, only increasing one of the two forms of "uncertainty" did not appear to impact the Bayesian estimates at all.

6 Problem 5

$$\begin{aligned} \mu &= 1 \\ \sigma^2 &= 1 \\ n &= \begin{bmatrix} 1\times10^2 & 1\times10^3 & 1\times10^4 & 1\times10^5 \end{bmatrix} \end{aligned} \tag{18}$$

$$\begin{aligned} \mu_0 &= 0 \\ \sigma_0^2 &= 1 \end{aligned} \tag{19}$$

Finally, for Problem 5, a series of 1D data sets are created with the parameters shown in Equation 18. The true mean μ is unknown and is instead considered a random variable normally distributed with the parameters shown in Equation 19.

$$p(x|D) \sim N(\mu_n, \sigma^2 + \sigma_n^2) \tag{20}$$

An objective of Problem 5 is to generate a series of plots that reveal how the probability of getting a particular data point x, given a particular data set D, changes with respect to a series of data sets. This probability is determine from the probability density function $p(x|D)$. As mentioned, the series of data points each have a different lengths n (i.e. the number of data points contained within each data set). Another objective is to show how the Bayesian estimated mean μ_n changes with respect to each data set D whose length is n. Equation 20 shows how the probability density of $p(x|D)$ is generated in the MATLAB source.

The plots included in Figures 11 to 13 summarize the results generated from the MATlAB source.

Figure 11: $p(x|D^n)$, where $n = 1 \times 10^2$ for the left plot and $n = 1 \times 10^3$ for the right plot

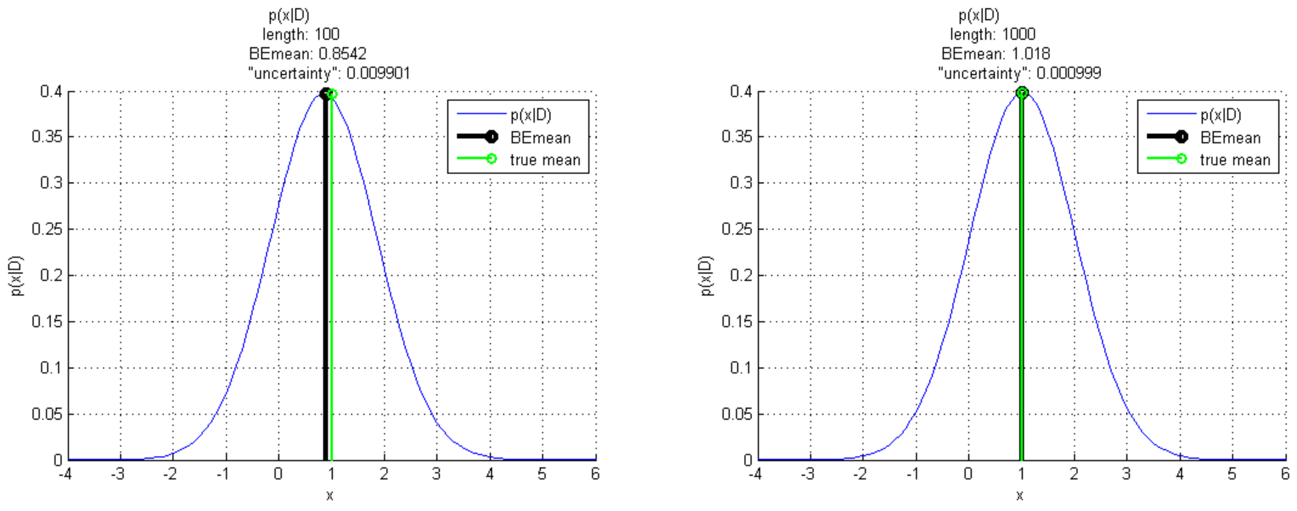
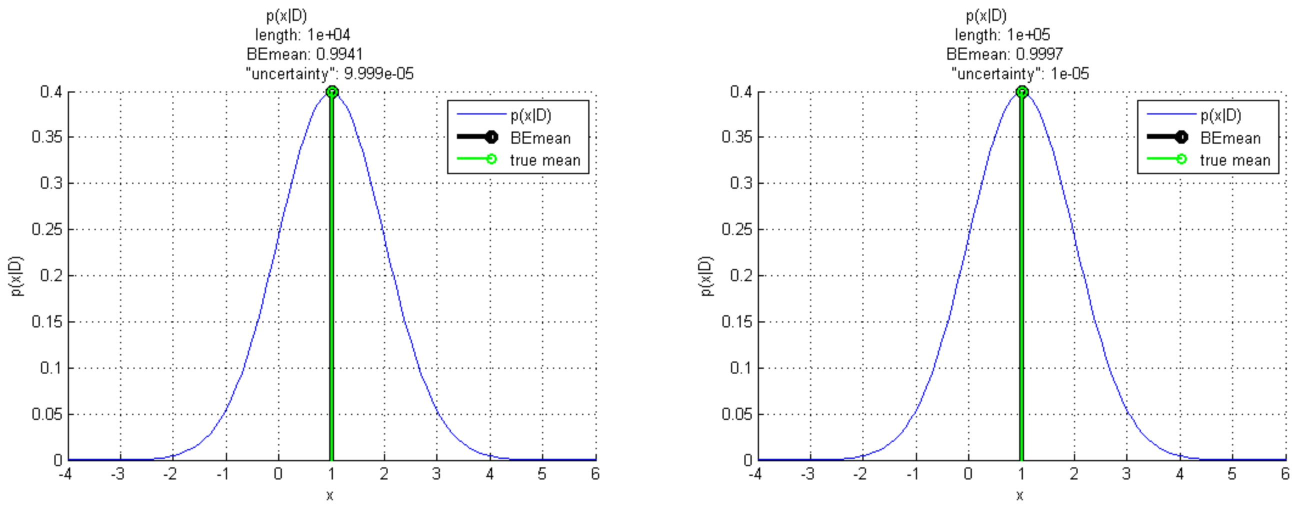


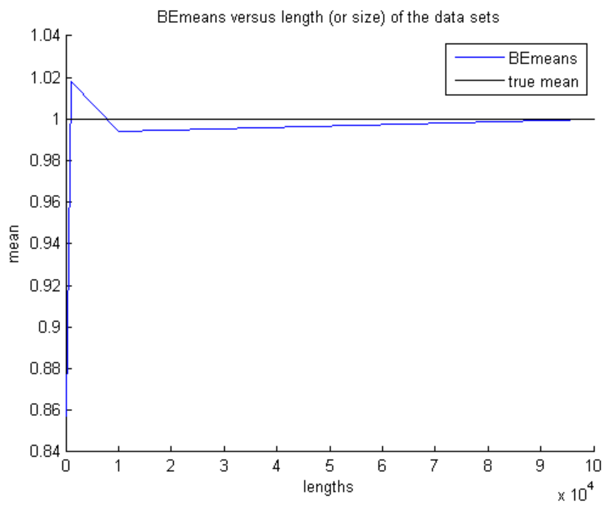
Figure 12: $p(x|D^n)$, where $n = 1 \times 10^4$ for the left plot and $n = 1 \times 10^5$ for the right plot



An initial glance at the plots in Figure 11 and 12 reveal wrongly choosing the mean based on prior knowledge μ_0 impacts the Bayesian estimated mean μ_n, especially when n is very low. Not shown for this problem are the simulations for which the prior mean μ_0 is changed to a value whose distance from the true mean μ is very large, that is, $|\mu_0 - \mu| > 1 \times 10^4$. The Bayesian estimated mean μ_n never really got close to the incorrect prior mean μ_0; however, the distance between the Bayesian estimated mean μ_n and the true mean μ is relatively a lot larger than compared to when $\mu_0 = 0$.

The plot in Figure 13 simply demonstrates the obvious: as the data sets D increase in size n, the more the Bayesian estimated mean μ_n converges to the true mean μ. Though, it is given the mean based on the data is approximately normally distributed with known "uncertainty" σ_0^2, i.e. $p(\mu|D^n) \sim N(\mu_n, \sigma_n^2)$.

Figure 13



7 Appendix

7.1 MATLAB source

```
function Homework2Script
close all;

Problem1;
Problem2;
Problem3;
Problem4;
Problem5;

end

function Problem1

% get discriminant function
g = getDiscriminantFunction;

% create anonymous function
%generateGrvPdf = @(X, mean, covariance)mvnpdf(X', mean', covariance)';

% generate a large combination of data vectors
dx = .2;
x1 = -10:dx:10;
x2 = -10:dx:10;
[X1, X2] = meshgrid(x1, x2);
X = [X1(:) X2(:)]';

% generate shared data between problem 1 and 2
[mean1, mean2, covariances] = Problem1And2Data;

% determine theoretical results
priors = 0:1/8:1;
for covariance1 = covariances
    for covariance2 = covariances

        % get probabilities
        grv1 = generateGrvPdf(X, mean1, covariance1{1});
        grv2 = generateGrvPdf(X, mean2, covariance2{1});
        % create error rate vector
        error = zeros(1, numel(priors));

        for prior = priors

            % determine error individual error rates for the given prior
            error1 = 0;
            error2 = 0;
            for n=1:numel(X(1,:))

                % features from state 1
                g1 = g(X(:,n), mean1, covariance1{1}, prior);
                g2 = g(X(:,n), mean2, covariance2{1}, 1-prior);
                error1 = error1 + (g1 <= g2)*grv1(n);

                % features from state 2
                g1 = g(X(:,n), mean1, covariance1{1}, prior);
                g2 = g(X(:,n), mean2, covariance2{1}, 1-prior);
                error2 = error2 + (g1 > g2)*grv2(n);
            end

            % add the errors to determne error rates
            % please note the priors themselves are multiplied to the
            % errors to effectively "reduce" the respective data, in
            % accordance to the priors
            error(priors == prior) = (error1*prior + error2*(1-prior))*2*dx^2;
        end

        % plot results
        figure;
        plot(priors, error);
        title(['Error rate versus priors(covariance1=' mat2str(covariance1{1}) ...
            ', covariance2=' mat2str(covariance2{1}) ')']);
        xlabel('P(omega1)');
        ylabel('P(error)');
    end
end

end

function Problem2

% generate shared data between problem 1 and 2
[mean1, mean2, covariances] = Problem1And2Data;

% set the other parameters of the problem
prior = .5;
lengths = [1e2, 1e3, 1e4, 1e5, 1e6];

% state the problem
disp(char('Problem 2', ' '));

for covariance1 = covariances(1)
  for covariance2 = covariances(1:2)
    for length = lengths

        % determine error rate
        errorrate = twoMLClassifier( ...
            mean1, covariance1{1}, ...
            mean2, covariance2{1}, ...
            prior, length);
        % generate and display the error rate
        disp(char(['Data size: ' num2str(length)], ...
            ['Error rate: ' num2str(errorrate)], ...
            ['Covariance1: ' mat2str(covariance1{1})], ...
            ['Covariance2: ' mat2str(covariance2{1})], ...
            ' '));
    end
  end
end

end

function Problem3

% generate the data
mean1 = [-2 -5]';
mean2 = [3 6]';
covariance1 = [2 1.5; 1.5 2];
covariance2 = [3 -2; -2 3];
prior1 = .5; % assumption
length = 10e4;

% generate error rate and data
[errorrateMLC, grv1X, grv2X] = twoMLClassifier( ...
    mean1, covariance1, ...
    mean2, covariance2, ...
    prior1, length);

% display the problem number
disp(char('Problem 3', ' '));

% create anonymous function for displaying results
precision = 5;
displayResult = @(header, length, errorrate, ...
    mean1, covariance1, mean2, covariance2) ...
    disp(char(header, ...
    ['Data size: ' num2str(length)], ...
    ['Error rate: ' num2str(errorrate)], ...
    ['Mean1: ' mat2str(mean1, precision)], ...
    ['Covariance1: ' mat2str(covariance1, precision)], ...
    ['Mean2: ' mat2str(mean2, precision)], ...
    ['Covariance2: ' mat2str(covariance2, precision)], ...
    ' '));

% display the error rate
displayResult('ML Classification: ', length, errorrateMLC, ...
    mean1, covariance1, mean2, covariance2);

% perform principal components analysis on the data sets
[grv1Xpp, A1] = performPrincipalComponentsAnalysis(grv1X);
[grv2Xpp, A2] = performPrincipalComponentsAnalysis(grv2X);

% get estimated meanas with MLE
[mean1e, covariance1e] = getMaximumLikelihoodEstimations(grv1Xpp);
[mean2e, covariance2e] = getMaximumLikelihoodEstimations(grv2Xpp);

% determine new error rate with the preprocessed data
getDistanceFromMean = @(X, mean)pdist([X' ; mean'],'euclidean');
errorratePCA = 0;
for n = 1:numel(grv1X(1,:))
    errorratePCA = errorratePCA + ...
         (getDistanceFromMean(grv1Xpp(:,n), mean1e) >= ...
        getDistanceFromMean(grv1Xpp(:,n), mean2e));
    errorratePCA = errorratePCA + ...
        (getDistanceFromMean(grv2Xpp(:,n), mean1e) < ...
        getDistanceFromMean(grv2Xpp(:,n), mean2e));
end
errorratePCA = errorratePCA/(length*2);

% display eigen vector matrices
disp(char('Eigen Vectors (i.e. Principal Components): ', ...
    ['eigen matrix1: ' mat2str(A1, precision)], ...
    ['eigen matrix1: ' mat2str(A2, precision)], ...
    ' '));

% display the error rate
displayResult('PCA Classification?: ', length, errorratePCA, ...
    mean1e, covariance1e, mean2e, covariance2e);

% set up eigen vectors
A1plot = {6*[[-A1(1,1) A1(1,1)]; [-A1(2,1) A1(2,1)]] + repmat(mean1e, 1, 2), ...
   3*[[-A1(1,2) A1(1,2)]; [-A1(2,2) A1(2,2)]] + repmat(mean1e, 1, 2)};
A2plot = {6*[[-A2(1,1) A2(1,1)]; [-A2(2,1) A2(2,1)]] + repmat(mean2e, 1, 2), ...
   3*[[-A2(1,2) A2(1,2)]; [-A2(2,2) A2(2,2)]] + repmat(mean2e, 1, 2)};

% plot density function
figure;
hold on
plot(grv1X(1,:), grv1X(2,:), '.b');
plot(grv2X(1,:), grv2X(2,:), '.g');
plot(A1plot{1}(1,:), A1plot{1}(2,:), '-k', 'LineWidth',3);
plot(A1plot{2}(1,:), A1plot{2}(2,:), '-k', 'LineWidth',3);
plot(A2plot{1}(1,:), A2plot{1}(2,:), '-k', 'LineWidth',3);
plot(A2plot{2}(1,:), A2plot{2}(2,:), '-k', 'LineWidth',3);
title(['Density plot for the raw data (class1 is in blue, whereas ' ...
    'class2 is in green)']);
xlabel('X(1,:)');
ylabel('X(2,:)');
axis equal
hold off

% plot density function
figure;
hold on
plot(grv1Xpp(1,:), grv1Xpp(2,:), '.b');
plot(grv2Xpp(1,:), grv2Xpp(2,:), '.g');
title(['Density plot for the preprocessed data (class1 is in blue, whereas ' ...
    'class2 is in green)']);
xlabel('X(1,:)');
ylabel('X(2,:)');
axis equal
hold off

end

function Problem4

% let's just work with a standard normal distribution for simplicity's sake

% theses anonymous functions generate the best guess mean with Bayesian
% Estimation and the uncertainty of the guess
[getBayesianEstimationMean, getBayesianEstimationUncertainty] = ...
    getBayesianEstimationFunctions;

% this anonymous function generates the uncertainty of the BE-determined
% mean
```

```matlab

% the data here is for generating the data for the 1D gaussian distribution
% represented by the random variable X
mean = 0;
variance = 10e9;
lengths = 100:5000:(100e3+100);

% For Bayesian Estimation, there is a degree of uncertainty about the
% parameter(s) that govern the distribution of the likelihood p(x|omega_i,D).
% The parameter(s) are thus viewed as random variables whose distribution
% (hopefully) converges to the true parameter(s) with more data.
%
% In the case of Problem 4, the mean of X's normal distribution is
% uncertain. It is given the mean is normally distributed,
% i,e. N(mean0, variance0). mean0 is the prior knowledge about the mean,
% whereas variance0 is the degree of uncertainty.
mean0 = 0;
variance0 = 10e9;

MLEmeans = zeros(1, numel(lengths));
BEmeans = zeros(1, numel(lengths));
BEuncertainty = zeros(1, numel(lengths));

for length = lengths

    % generate the data
    X = generateGrv(mean, variance, length);

    % determine the Maximum Likelihood of the mean and the "best guess" of
    % the mean with Bayesian Estimation
    MLEmeans(length == lengths) = getMaximumLikelihoodEstimations(X);
    BEmeans(length == lengths) = getBayesianEstimationMean( ...
        length, ...
        mean0, variance0, ...
        variance,X);

    % determine the Bayesian Estimation uncertainty with respect thhe size
    % of the data
    BEuncertainty(length == lengths) = getBayesianEstimationUncertainty( ...
        length, variance0, variance);
end

% generate plots
figure;
hold on
plot(lengths, MLEmeans, 'b','LineWidth',5);
plot(lengths, BEmeans, 'r','LineWidth',2);
plot(lengths, repmat(mean, numel(lengths)), 'k');
title('Mean versus data size');
legend('MLE means','BE means','true mean');
ylabel('Mean');
xlabel('size of data');
hold off

figure;
hold on
plot(lengths, BEuncertainty, 'r','LineWidth',2);
plot(lengths, repmat(mean, numel(lengths)), 'k');
title('BE uncertainty versus data size');
ylabel('BE uncertainty');
xlabel('size of data');
hold off
```

```
% save data
% generate differences
convertMatrixToLatexWithPrecision([lengths' MLEmeans' BEmeans' abs(MLEmeans'-BEmeans')], 5)
end

function Problem5

[getBayesianEstimationMean, getBayesianEstimationUncertainty] = ...
    getBayesianEstimationFunctions;


mean = 1;
variance = 1;
mean0 = -1000;
variance0 = 1;
lengths = [1e2, 1e3, 1e4, 1e5];

pdfX = mean-5:.1:mean+5;
BEmeans = zeros(1, numel(lengths));

for length = lengths
    X = generateGrv(mean, variance, length);

    BEmean = getBayesianEstimationMean( ...
        length, ...
        mean0, ...
        variance0, ...
        variance, ...
        X);
    BEmeans(length == lengths) = BEmean;
    BEuncertainty = getBayesianEstimationUncertainty( ...
        length, ...
        variance0, ...
        variance);

    pdfXGivenD = generateGrvPdf(pdfX, BEmean, variance+BEuncertainty);

    figure;
    hold on
    [maxPdfXGivenD, maxIndex] = max(pdfXGivenD);
    maxPdfX = pdfX(maxIndex);
    plot(pdfX, pdfXGivenD, 'b-');
    stem(maxPdfX, maxPdfXGivenD, 'k','LineWidth',3);
    stem(mean, maxPdfXGivenD, 'g','LineWidth',2);
    legend('p(x|D)','BEmean','true mean');
    grid on
    title(char( ...
        'p(x|D)', ...
        ['length: ' num2str(length, 4)], ...
        ['BEmean: ' num2str(BEmean, 4)], ...
        ['"uncertainty": ' num2str(BEuncertainty, 4)]));
    ylabel('p(x|D)');
    xlabel('x');
    hold off
end

figure;
hold on
plot(lengths, BEmeans,'b');
plot(lengths, repmat(mean,1,numel(lengths)), 'k');
legend('BEmeans','true mean');
title('BEmeans versus length (or size) of the data sets');
```

```
ylabel('mean');
xlabel('lengths');
hold off

end

function latexMatrix = convertMatrixToLatexWithPrecision(M,p)

d = digits(p);
latexMatrix = latex(sym(vpa(M)));
digits(d);

end

function X = generateGrv(mean, covariance, length)

    X = mvnrnd(mean', covariance, length)';

end

function pdf = generateGrvPdf(X, mean, covariance)

pdf = mvnpdf(X', mean', covariance)';

end

function [errorrate, grv1X, grv2X] = twoMLClassifier( ...
    mean1, covariance1, ...
    mean2, covariance2, ...
    prior1, length)

% get discriminant function
g = getDiscriminantFunction;

% generate the data
%generateGrv = @(mean, covariance, length)mvnrnd(mean', covariance, length)';
grv1X = generateGrv(mean1, covariance1, length);
grv2X = generateGrv(mean2, covariance2, length);

% estimate mean and variance of the data
[mean1e, covariance1e] = getMaximumLikelihoodEstimations(grv1X);
[mean2e, covariance2e] = getMaximumLikelihoodEstimations(grv2X);

% determine error rate for equal priors
error1 = 0;
error2 = 0;
for n=1:length
    error1 = error1 ...
        + (g(grv1X(:,n), mean1e, covariance1e, prior1) ...
        <= g(grv1X(:,n), mean2e, covariance2e, 1-prior1));
    error2 = error2 ...
        + (g(grv2X(:,n), mean1e, covariance1e, prior1) ...
        > g(grv2X(:,n), mean2e, covariance2e, 1-prior1));
end

% generate and display the error rate
errorrate = (error1+error2)/length;

end

function [Xpp, A] = performPrincipalComponentsAnalysis(X)

% determine the mean and covariance from MLE
```

```
[mean, covariance] = getMaximumLikelihoodEstimations(X);

% obtain eigen vectors
[V, ~] = eig(covariance);

% flip eigen vector matrix such that the columns of the eigen vector matrix
% correspond to a descending eigen values
A = fliplr(V);

% apply transformaion
Xpp = zeros(size(X));
for n = 1:numel(X(1,:))
    Xpp(:,n) = A'*(X(:,n)-mean);
end

% normalize variances
[~, covariancepp] = getMaximumLikelihoodEstimations(Xpp);
for n = 1:numel(mean)
    Xpp(n,:)=Xpp(n,:)/(covariancepp(n,n)^(1/2));
end

% add back the mean
Xpp = Xpp + repmat(mean, 1, numel(Xpp(1,:)));

end

function g = getDiscriminantFunction

% discriminant function for normal distributions
% g_i(x)=ln(p(x|omega_i)P(omega_i))
g = @(X, mean, covariance, prior) ...
    -1/2*(X-mean)'*covariance^(-1)*(X-mean) ...
    -numel(mean)/2*log(2*pi) ...
    -1/2*log(det(covariance)) ...
    +log(prior);

end

function [getBayesianEstimationMean, getBayesianEstimationUncertainty] = ...
    getBayesianEstimationFunctions

getBayesianEstimationMean = @(length,mean0,variance0,variance,X) ...
     (length*variance0/(length*variance0+variance)) ...
     *getMaximumLikelihoodEstimations(X) ...
     + (variance/(length*variance0+variance))*mean0;
getBayesianEstimationUncertainty = @(length,variance0,variance) ...
     (variance0*variance)/(length*variance0+variance);

end

function [mean1, mean2, covariances] = Problem1And2Data

% mean vectors
mean1 = [1 1]';
mean2 = [-1 -1]';

% covariances
covariances = {eye(2),[1 .5; .5 1],[1 .75; .75 1]};

end

function [mean, covariance] = getMaximumLikelihoodEstimations(X)
    mean = 1/numel(X(1,:))*sum(X,2);
    covariance = 0;
    for n =1:numel(X(1,:))
        covariance = covariance + (X(:,n)-mean)*(X(:,n)-mean)';
    end
    covariance = covariance/numel(X(1,:));
end
```

Listing 5: MATLAB source