Programming Assignment 1

CS 474

https://github.com/alexander-novo/CS479-PA1

Nikhil Khedekar	Mehar Mangat	Alexander Novotny
10% Code, 0% Report	0% Code, 10% Report	90% Code, 90% Report
Parts 3,4	Error Checking	Parts 1,2,3,4

Due: March 10, 2021
Submitted: March 10, 2021

Contents

1 Parts 1 & 2 **1**
- 1.1 Theory . . . 1
  - 1.1.1 Generating arbitrary multivariate Gaussian-distributed random vectors . . . 1
  - 1.1.2 Bayesian classifier for Gaussian-distributed classes . . . 1
  - 1.1.3 Finding the decision boundary . . . 3
  - 1.1.4 The derivative of the error bound function . . . 3
- 1.2 Implementation . . . 5
- 1.3 Results and Discussion . . . 6
  - 1.3.1 Data Set A . . . 6
  - 1.3.2 Data Set B . . . 8

2 Parts 3 & 4 **10**
- 2.1 Theory . . . 10
- 2.2 Implementation . . . 10
- 2.3 Results and Discussion . . . 11
  - 2.3.1 Data Set A . . . 11
  - 2.3.2 Data Set B . . . 12

1 Parts 1 & 2

1.1 Theory

1.1.1 Generating arbitrary multivariate Gaussian-distributed random vectors

While generating random vectors distributed as a standard normal distribution is simple enough (using a Box-Muller transform or the quantile function), we're still left with transforming it to an arbitrary Gaussian distribution. The key to being able to do this is through the Whitening transform, defined as

$$A_w^\top = \left(\Phi\Lambda^{-1/2}\right)^\top, \tag{1}$$

where $\Sigma = \Phi\Lambda\Phi^{-1}$ is an eigendecomposition (Φ is an orthonormal matrix of eigenvectors of Σ and Λ is a diagonal matrix of eigenvalues of Σ) of the covariance matrix Σ of a multivariate Gaussian distribution. Since Σ is a real symmetric matrix, such an eigendecomposition always exists. Applying this whitening transform to $\vec{X} \sim \mathcal{N}(\vec{0}, \Sigma)$ as

$$\vec{Y} = A_w^\top \vec{X} \tag{2}$$

yields $\vec{Y} \sim \mathcal{N}(\vec{0}, A_w^\top \Sigma A_w)$ and

$$\begin{aligned} A_w^\top \Sigma A_w &= \left(\Phi\Lambda^{-1/2}\right)^\top \left(\Phi\Lambda\Phi^{-1}\right)\left(\Phi\Lambda^{-1/2}\right) \\ &= \left(\Lambda^{-1/2}\right)^\top \Phi^\top \Phi\Lambda\Lambda^{-1/2} \\ &= \Lambda^{-1/2}\Phi^{-1}\Phi\Lambda^{1/2} \\ &= \Lambda^0 \\ &= I, \end{aligned}$$

so $Y \sim \mathcal{N}(\vec{0}, I)$, the standard Gaussian distribution. Then by multiplying both sides of eq. (2) by the inverse transformation

$$\begin{aligned} \left(A_w^\top\right)^{-1} &= \left(\left(\Phi\Lambda^{-1/2}\right)^\top\right)^{-1} \\ &= \left(\left(\Phi\Lambda^{-1/2}\right)^{-1}\right)^\top \\ &= \left(\Lambda^{1/2}\Phi^{-1}\right)^\top \\ &= \Phi\Lambda^{1/2} \\ &= \Phi\Lambda^{1/2}\Phi^{-1}\Phi \\ &= \Sigma^{1/2}\Phi, \end{aligned} \tag{3}$$

we arrive at

$$\vec{X} = \left(A_w^\top\right)^{-1}\vec{Y}. \tag{4}$$

Then we can simply add $\vec{\mu}$ to obtain $\vec{Z} = \vec{X} + \vec{\mu}$ where $\vec{Z} \sim \mathcal{N}(\vec{\mu}, \Sigma)$ is our "goal" distribution.

1.1.2 Bayesian classifier for Gaussian-distributed classes

When designing a classifier, we want to pick discriminant functions for each class that indicate when we should pick that class, i.e. we should pick class ω_i if $g_i(\vec{X}) \geq g_j(\vec{X})$ for all j. For a Bayesian classifier,

these discriminant functions should be the probability that $\vec{X}$ came from the class, but any function of the discriminant that preserves this inequality (monotonic increasing functions) will also work as a discriminant. So, in the general case (referred to as "case 3" in the textbook) of a Bayesian classifier with multivariate Gaussian distribution, we can have (for Y the class of $\vec{X}$)

$$\begin{aligned}
g_i(\vec{x}) &= \ln\Big(P(Y=\omega_i \mid \vec{X}=\vec{x})P(\vec{X}=\vec{x})\Big) \\
&= \ln\left(\left(\frac{P(\vec{X}=\vec{x} \mid Y=\omega_i)P(Y=\omega_i)}{P(\vec{X}=\vec{x})}\right)P(\vec{X}=\vec{x})\right) \\
&= \ln\Big(P(\vec{X}=\vec{x} \mid Y=\omega_i)\Big) + \ln(P(Y=\omega_i)) \\
&= \ln\left(\frac{1}{(2\pi)^{1/d}|\Sigma_i|^{1/2}}\exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu}_i)^\top\Sigma_i^{-1}(\vec{x}-\vec{\mu}_i)\right)\right) + \ln(P(Y=\omega_i)) \\
&= -\frac{1}{d}\ln(2\pi) - \frac{1}{2}\ln|\Sigma_i| - \frac{1}{2}(\vec{x}-\vec{\mu}_i)^\top\Sigma_i^{-1}(\vec{x}-\vec{\mu}_i) + \ln(P(Y=\omega_i)),
\end{aligned}$$

which we can further refine by subtracting the constant $-\frac{1}{d}\ln(2\pi)$:

$$g_i(\vec{x}) = -\frac{1}{2}\ln|\Sigma_i| - \frac{1}{2}(\vec{x}-\vec{\mu}_i)^\top\Sigma_i^{-1}(\vec{x}-\vec{\mu}_i) + \ln(P(Y=\omega_i)). \tag{5}$$

By further narrowing down certain cases, we can simplify this discriminant by removing other terms which become constant across classes. For instance, when $\Sigma_i = \Sigma$ (referred to as "case 2" in the textbook), then the $-\frac{1}{2}\ln|\Sigma_i|$ term becomes constant and we have

$$\begin{aligned}
g_i(\vec{x}) &= -\frac{1}{2}(\vec{x}-\vec{\mu}_i)^\top\Sigma^{-1}(\vec{x}-\vec{\mu}_i) + \ln(P(Y=\omega_i)) \qquad (6)\\
&= -\frac{1}{2}(\vec{x}^\top - \vec{\mu}_i^\top)(\Sigma^{-1}\vec{x} - \Sigma^{-1}\vec{\mu}_i) + \ln(P(Y=\omega_i)) \\
&= -\frac{1}{2}(\vec{x}^\top(\Sigma^{-1}\vec{x} - \Sigma^{-1}\vec{\mu}_i) - \vec{\mu}_i^\top(\Sigma^{-1}\vec{x} - \Sigma^{-1}\vec{\mu}_i)) + \ln(P(Y=\omega_i)) \\
&= -\frac{1}{2}(\cancelto{const}{\vec{x}^\top\Sigma^{-1}\vec{x}} - \vec{x}^\top\Sigma^{-1}\vec{\mu}_i - \vec{\mu}_i^\top\Sigma^{-1}\vec{x} + \vec{\mu}_i^\top\Sigma^{-1}\vec{\mu}_i) + \ln(P(Y=\omega_i)) \\
&= -\frac{1}{2}(-((\Sigma^{-1}\vec{\mu}_i)^\top\vec{x})^\top - (\Sigma^{-\top}\vec{\mu}_i)^\top\vec{x} + \vec{\mu}_i^\top\Sigma^{-1}\vec{\mu}_i) + \ln(P(Y=\omega_i)) \\
&= -\frac{1}{2}(-(\Sigma^{-1}\vec{\mu}_i)^\top\vec{x} - (\Sigma^{-1}\vec{\mu}_i)^\top\vec{x} + \vec{\mu}_i^\top\Sigma^{-1}\vec{\mu}_i) + \ln(P(Y=\omega_i))
\end{aligned}$$

(since $(\Sigma^{-1}\vec{\mu}_i)^\top\vec{x}$ is a scalar, and Σ, Σ^{-1} are symmetric)

$$= (\Sigma^{-1}\vec{\mu}_i)^\top\vec{x} - \frac{1}{2}\vec{\mu}_i^\top\Sigma^{-1}\vec{\mu}_i + \ln(P(Y=\omega_i)). \tag{7}$$

Furthermore, if Σ is a scalar matrix $\Sigma = \sigma^2 I$ (referred to as "case 1" in the book), then $\Sigma^{-1} = \frac{1}{\sigma^2}$ and from eq. (6) above, we have

$$\begin{aligned}
g_i(\vec{x}) &= -\frac{1}{2\sigma^2}(\vec{x}-\vec{\mu}_i)^\top(\vec{x}-\vec{\mu}_i) + \ln(P(Y=\omega_i)) \\
&= -\frac{||\vec{x}-\vec{\mu}_i||^2}{2\sigma^2} + \ln(P(Y=\omega_i)). \qquad (8)
\end{aligned}$$

Further simplification can be found in section 2.1.

1.1.3 Finding the decision boundary

It is of interest to find the boundary along which the classifier changes its decision between classes. It is a defined as

$$B = \{\vec{x} \mid g_i(\vec{x}) = g_j(\vec{x}) \geq g_k(\vec{x}) \text{ for some } i \neq j \text{ and all } k\}, \tag{9}$$

and for classifiers which are classifying between only two classes (such as the ones in this assignment), the inequality is trivially satisfied. Therefore, we're interesting in finding the roots $\vec{x}$ such that $g_i(\vec{x}) - g_j(\vec{y}) = 0$. Taking eq. (5) above (the most general case) and reorganizing it in a similar way in eq. (7) to separate out $\vec{x}$ more cleanly, we have

$$\begin{aligned}
g_i(\vec{x}) &= -\frac{1}{2}(\vec{x} - \vec{\mu}_i)^\top \Sigma_i^{-1}(\vec{x} - \vec{\mu}_i) - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)) \\
&= -\frac{1}{2}(\vec{x}^\top - \vec{\mu}_i^\top)(\Sigma_i^{-1}\vec{x} - \Sigma_i^{-1}\vec{\mu}_i) - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)) \\
&= -\frac{1}{2}(\vec{x}^\top(\Sigma_i^{-1}\vec{x} - \Sigma_i^{-1}\vec{\mu}_i) - \vec{\mu}_i^\top(\Sigma_i^{-1}\vec{x} - \Sigma_i^{-1}\vec{\mu}_i)) - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)) \\
&= -\frac{1}{2}(\vec{x}^\top\Sigma_i^{-1}\vec{x} - \vec{x}^\top\Sigma_i^{-1}\vec{\mu}_i - \vec{\mu}_i^\top\Sigma_i^{-1}\vec{x} + \vec{\mu}_i^\top\Sigma_i^{-1}\vec{\mu}_i) - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)) \\
&= -\frac{1}{2}\vec{x}^\top\Sigma_i^{-1}\vec{x} + (\Sigma_i^{-1}\vec{\mu}_i)^\top\vec{x} - \frac{1}{2}\vec{\mu}_i^\top\Sigma_i^{-1}\vec{\mu}_i - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i)),
\end{aligned}$$

which gives us

$$g_i(\vec{x}) - g_j(\vec{x}) = \vec{x}^\top W_{ij}\vec{x} + \vec{w}_{ij}^\top\vec{x} + w_{0_{ij}} = 0, \tag{10}$$

$$W_{ij} = -\frac{1}{2}\left(\Sigma_i^{-1} - \Sigma_j^{-1}\right), \tag{11}$$

$$\vec{w}_{ij} = \Sigma_i^{-1}\vec{\mu}_i - \Sigma_j^{-1}\vec{\mu}_j, \tag{12}$$

$$\begin{aligned}
w_{0_{ij}} = &\left(-\frac{1}{2}\vec{\mu}_i^\top\Sigma_i^{-1}\vec{\mu}_i - \frac{1}{2}\ln|\Sigma_i| + \ln(P(Y = \omega_i))\right) - \\
&\left(-\frac{1}{2}\vec{\mu}_j^\top\Sigma_j^{-1}\vec{\mu}_j - \frac{1}{2}\ln|\Sigma_j| + \ln(P(Y = \omega_j))\right).
\end{aligned} \tag{13}$$

This is in the general form of a quadric, which in two dimensions looks like

$$ax^2 + bx + cy^2 + dx + ey + f = 0,$$

where

$$W_{ij} = \begin{pmatrix} a & \frac{1}{2}b \\ \frac{1}{2}b & c \end{pmatrix}, \tag{14}$$

$$\vec{w}_{ij} = \begin{bmatrix} d \\ e \end{bmatrix}, \tag{15}$$

$$w_{0_{ij}} = f. \tag{16}$$

1.1.4 The derivative of the error bound function

When trying to minimize the error bound function for 2-class multivariate Gaussian Bayesian classifiers,

$$\begin{aligned}
f(\beta) &= (P(Y = \omega_1))^\beta (P(Y = \omega_2))^{1-\beta} \exp(-k(\beta)), \\
k(\beta) &= \frac{\beta(1-\beta)}{2}(\vec{\mu}_1 - \vec{\mu}_2)^\top[(1-\beta)\Sigma_1 + \beta\Sigma_2]^{-1}(\vec{\mu}_1 - \vec{\mu}_2) + \frac{1}{2}\ln\left(\frac{|(1-\beta)\Sigma_1 + \beta\Sigma_2|}{|\Sigma_1|^{1-\beta}|\Sigma_2|^\beta}\right),
\end{aligned} \tag{17}$$

it is useful to know the derivative, $f'(\beta)$. However, since this involves many chain rules, product rules, quotient rules, and matrix derivative rules, we should first break the derivatives up into parts. Note that

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\beta}\Big[a^{f(\beta)}\Big] &= \frac{\mathrm{d}}{\mathrm{d}\beta}\Big[\exp\Big(\ln\Big(a^{f(\beta)}\Big)\Big)\Big] \\ &= \frac{\mathrm{d}}{\mathrm{d}\beta}[\exp(f(\beta)\ln(a))] \\ &= \exp(f(\beta)\ln(a))\frac{\mathrm{d}}{\mathrm{d}\beta}[f(\beta)\ln(a)] \\ &= a^{f(\beta)}\bigg(\ln(a)\frac{\mathrm{d}f}{\mathrm{d}\beta}\bigg), \end{aligned} \tag{18}$$

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\beta}\Big[[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}\Big] &= -[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}\frac{\mathrm{d}}{\mathrm{d}\beta}[(1-\beta)\Sigma_1+\beta\Sigma_2][(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1} \\ &= -[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}(-\Sigma_1+\Sigma_2)[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}, \end{aligned} \tag{19}$$

and by Jacobi's Formula,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\beta}|(1-\beta)\Sigma_1+\beta\Sigma_2| &= |(1-\beta)\Sigma_1+\beta\Sigma_2|\,\mathrm{tr}\left([(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}\frac{\mathrm{d}}{\mathrm{d}\beta}[(1-\beta)\Sigma_1+\beta\Sigma_2]\right) \\ &= |(1-\beta)\Sigma_1+\beta\Sigma_2|\,\mathrm{tr}\Big([(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}(-\Sigma_1+\Sigma_2)\Big). \end{aligned} \tag{20}$$

Finally, if we rewrite

$$\begin{aligned} f(\beta) &= f_1(\beta)g_1(\beta)h_1(\beta), \\ f_1(\beta) &= (P(Y=\omega_1))^{\beta}, \\ g_1(\beta) &= (P(Y=\omega_2))^{1-\beta}, \\ h_1(\beta) &= \exp(-k(\beta)), \\ k(\beta) &= \frac{1}{2}f_2(\beta)g_2(\beta)h_2(\beta)+\frac{1}{2}\ln\bigg(\frac{f_3(\beta)}{g_3(\beta)h_3(\beta)}\bigg), \\ f_2(\beta) &= \beta, \\ g_2(\beta) &= 1-\beta, \\ h_2(\beta) &= (\vec{\mu}_1-\vec{\mu}_2)^{\top}[(1-\beta)\Sigma_1+\beta\Sigma_2]^{-1}(\vec{\mu}_1-\vec{\mu}_2), \\ f_3(\beta) &= |(1-\beta)\Sigma_1+\beta\Sigma_2|, \\ g_3(\beta) &= |\Sigma_1|^{1-\beta}, \\ h_3(\beta) &= |\Sigma_2|^{\beta}, \end{aligned}$$

then we have

$$
\begin{aligned}
f'(\beta) &= f_1'(\beta)g_1(\beta)h_1(\beta) + f_1(\beta)g_1'(\beta)h_1(\beta) + f_1(\beta)g_1(\beta)h_1'(\beta), \qquad (21)\\
f_1'(\beta) &\overset{(18)}{=} (P(Y=\omega_1))^{\beta}\ln(P(Y=\omega_1)),\\
g_1'(\beta) &\overset{(18)}{=} -(P(Y=\omega_2))^{1-\beta}\ln(P(Y=\omega_2)),\\
h_1'(\beta) &= -\exp(-k(\beta))k'(\beta),\\
k'(\beta) &= \frac{1}{2}f_2'(\beta)g_2(\beta)h_2(\beta) + f_2(\beta)g_2'(\beta)h_2(\beta) + f_2(\beta)g_2(\beta)h_2'(\beta))\\
&\quad + \frac{\left(\frac{f_3'(\beta)g_3(\beta)h_3(\beta) - f_3(\beta)g_3'(\beta)h_3(\beta) - f_3(\beta)g_3(\beta)h_3'(\beta)}{g_3(\beta)^2 h_3(\beta)^2}\right)}{2\left(\frac{f_3(\beta)}{g_3(\beta)h_3(\beta)}\right)},\\
f_2'(\beta) &= 1,\\
g_2'(\beta) &= -1,\\
h_2'(\beta) &\overset{(19)}{=} -(\vec{\mu}_1 - \vec{\mu}_2)^{\top}[(1-\beta)\Sigma_1 + \beta\Sigma_2]^{-1}(-\Sigma_1 + \Sigma_2)[(1-\beta)\Sigma_1 + \beta\Sigma_2]^{-1}(\vec{\mu}_1 - \vec{\mu}_2),\\
f_3'(\beta) &\overset{(20)}{=} |(1-\beta)\Sigma_1 + \beta\Sigma_2|\operatorname{tr}\left([(1-\beta)\Sigma_1 + \beta\Sigma_2]^{-1}(-\Sigma_1 + \Sigma_2)\right),\\
g_3'(\beta) &\overset{(18)}{=} -|\Sigma_1|^{1-\beta}\ln|\Sigma_1|,\\
h_3'(\beta) &\overset{(18)}{=} |\Sigma_2|^{\beta}\ln|\Sigma_2|.
\end{aligned}
$$

1.2 Implementation

All matrix calculations are done using the Eigen C++ library (`http://eigen.tuxfamily.org/`). Random standard multivariate Gaussian-distributed vectors are generated using the C++11 `<random>` library, and then the transform described in eq. (3) is applied to get the correct distribution. Eigen provides a library (`SelfAdjointEigenSolver`) for quickly finding the eigendecomposition of symmetric matrices. This library includes a way to retrieve Φ and a way to calculate $\Sigma^{1/2}$.

A classifier is then implemented by automatically detecting the case and choosing from three different discriminant functions (described in eqs. (5), (7) and (8)) based on the detected case. The prior probabilities for each class is calculated as the quotient of the sample size for the class and the sum of the sample sizes for all classes.

The inverse variance matrices and variance determinants are pre-computed based on the case detected. If the case is 1, then the determinant is not needed and the inverse matrix is calculated by Eigen very easily using its `Diagonal` library, which presumably just finds the reciprocal of each diagonal element. If the case is 2, then the determinant is still not needed, but the matrices are now arbitrary and more difficult to find inverses for. However, since covariance matrices are symmetric positive semi-definite, we can use the Cholesky decomposition, which is provided by Eigen's `LLT` library. Finally, if the case is 3, then each covariance matrix is still positive semi-definite, but we need to calculate the determinant of each one (see eq. (5)), and the Cholesky decomposition does not help with that. Instead we use the LU decomposition, which gives both the matrix inverse and determinant easily via Eigen's `PartialPivLU` library, but is considerably slower than calculating the Cholesky decomposition.

Then each vector generated in the sample is classified and checked against the class it was generated from. If they mismatch, the mismatch is tallied and the classifier continues until each sample is classified. The error rate is then calculated as the quotient of the number of mismatched vectors and the total number of vectors. The Bhattacharyya bound is calculated from eq. (17) with $\beta = 0.5$. Then, a hybrid iterative root-finding algorithm that uses the secant method (or the bisection method if the secant method attempts to diverge and to set the second initial guess for the secant method) with an initial

guess of $\beta = 0.5$ is run on eq. (21) to find the Chernoff Bound. This should always work since there should be only one local minimum in $(0, 1)$ and $\beta = 0.5$ should be quite close. Finally, the decision boundary quadric parameters are calculated using eqs. (14) to (16) and then written out into an output file, where there will be used by plotting software to plot the boundary.

To make plots, `gnuplot` was used. To plot the decision boundary, `gnuplot` first plots the quadric as a surface, then finds the contour curve at $z = 0$ according to eq. (10). This curve is the decision boundary. In addition, the joint pdf is pre-calculated and each point on the surface is classified according to the more likely class. This is included as a surface which lies above the sample scatter plots and decision boundary to illustrate that the decision boundary lines up with which class is actually more likely at each point.

1.3 Results and Discussion

1.3.1 Data Set A

The results of classifying data set A can be found in listing 1. We can see that the misclassification rate for class 1 is significantly higher than the misclassification rate for class 2 - this is due to the prior probability of class 1 being lower than the prior probability of class 2, so the Bayes classifier is correctly trading higher classification error from class 1 into more classification error reduction from class 2. Also, the overall misclassification rate is correctly much lower than both the Bhattacharyya bound and the Chernoff bound, although the Chernoff bound does not improve on the Bhattacharyya bound that much.

Listing 1: Output from classifying data set A.

```
Classifying data set "A" - 2 classes.
Detected case 1

Prior probabilities:
0.3
0.7

Misclassification rates:
0.02755
0.0102571

Overall misclassification rate:
0.015445

Bhattacharyya Bound:
0.0483

Chernoff Bound:
0.0473463 @ beta = 0.547072
```

A plot of the samples generated as well as the misclassified points can be found in figs. 1 and 2, respectively. Note that the decision boundary agrees with both the misclassified points (as each class only has misclassified points on one side of the boundary) and the colored boundary along the joint pdf, where the most likely class changes.

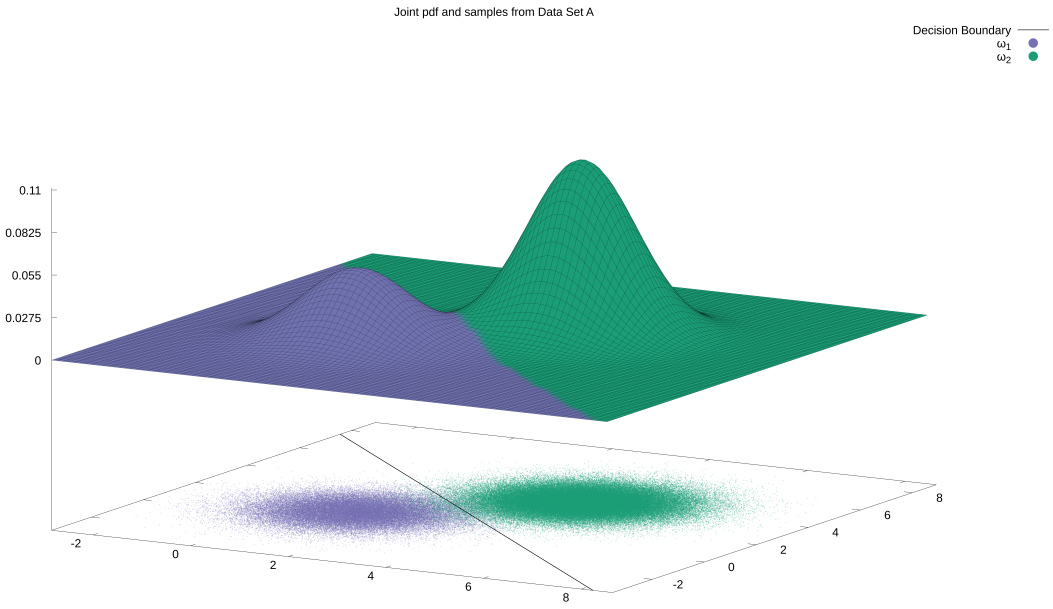

Figure 1: A plot of the samples from Data Set A, the classification boundary, and the joint pdf. The joint pdf is colored according to which class is more likely at that point.

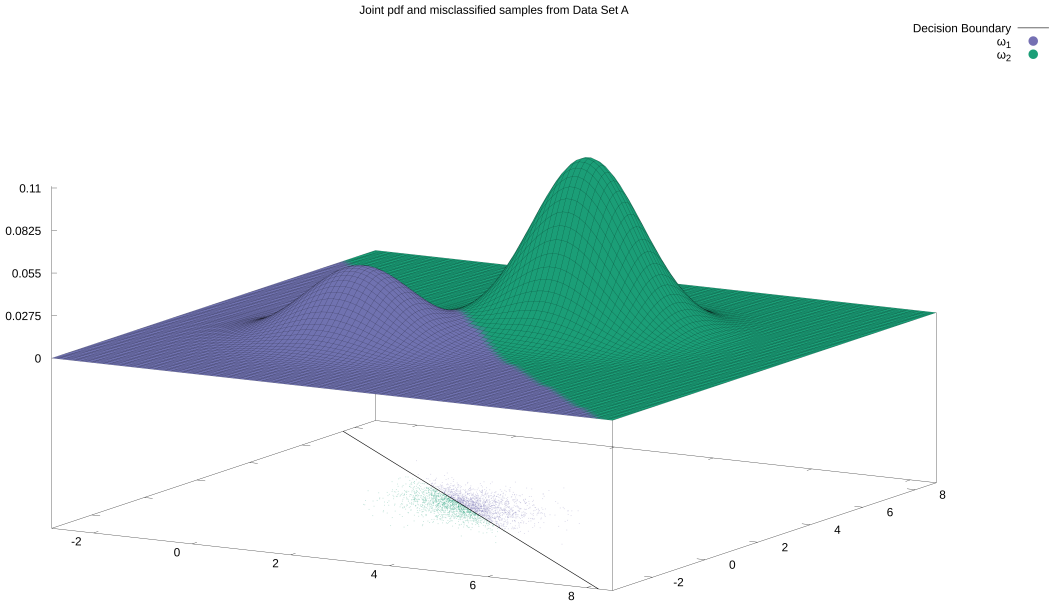

Figure 2: A plot of the misclassified points from Data Set A. Note that each class only has misclassified points on one side of the boundary.

A plot of the error bound function from eq. (17) can be found in fig. 3. Note that the Chernoff bound is only slightly lower than the Bhattacharyya bound, but β^* is correctly placed at the root $f'(\beta^*) = 0$, implying that the Chernoff bound is indeed the minimum.

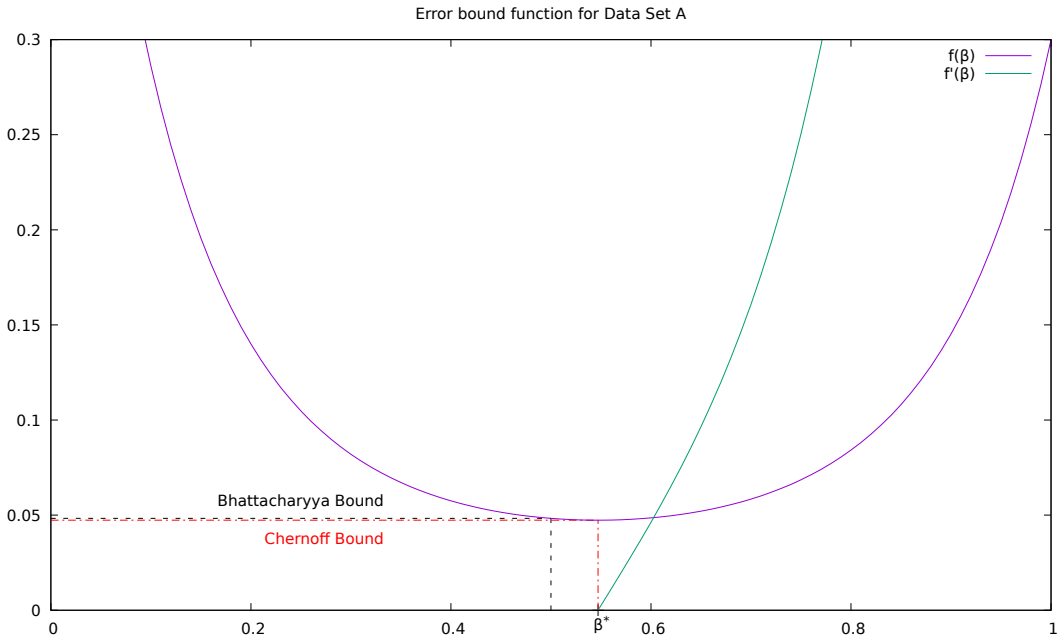

Figure 3: A plot of the error bound function $f(\beta)$ for sample A. Both the Bhattacharyya bound and the Chernoff bound (at β^*) are marked.

1.3.2 Data Set B

The results of classifying data set B can be found in listing 2. Note that the covariance matrices of the different classes are different, so the classifier correctly chooses case 3. Once again, the misclassification rate for class 1 is much higher than the misclassification rate for class 2 due to the disparity in prior probability. The disparity is even larger in data set B than data set A, so the difference in error rate is also larger. Once again, the overall misclassification rate is much lower than both theoretical bounds, and this time the difference between the Chernoff bound and the Bhattacharyya bound is just about unnoticeable.

Listing 2: Output from classifying data set B.

```
Classifying data set "B" - 2 classes.
Detected case 3

Prior probabilities:
0.2
0.8

Misclassification rates:
0.13545
0.0576625

Overall misclassification rate:
0.07322

Bhattacharyya Bound:
0.140853

Chernoff Bound:
0.140702 @ beta = 0.521114
```

A plot of the samples generated as well as the misclassified points can be found in figs. 4 and 5, respectively. Note that the decision boundary is elliptical this time - this is due to the fact that the case 3 discriminant was used. It once again agrees with the misclassified points and the joint pdf.

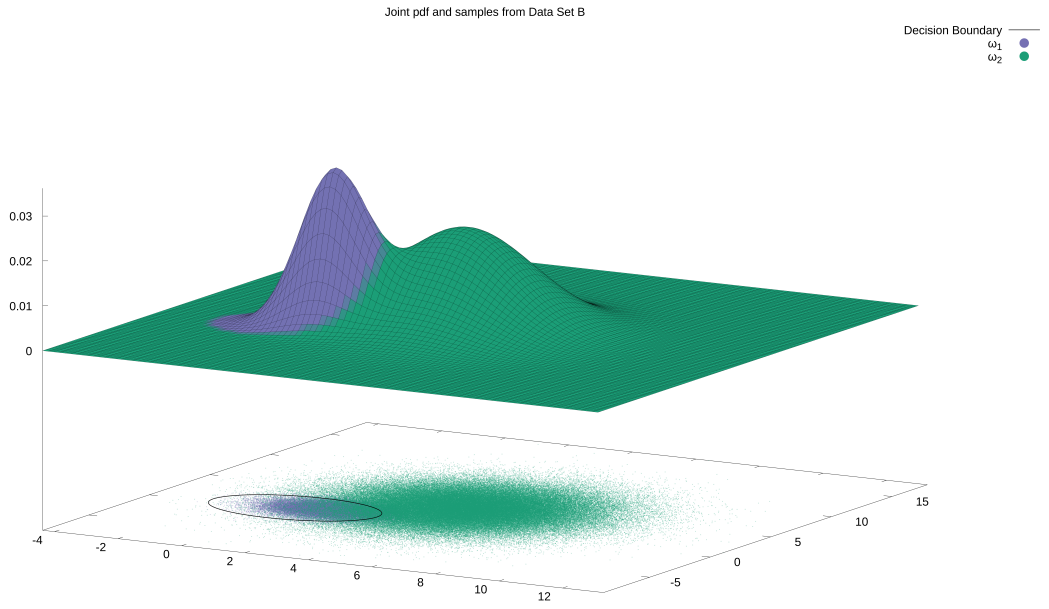

Figure 4: A plot of the samples from Data Set B, the classification boundary, and the joint pdf. The joint pdf is colored according to which class is more likely at that point.

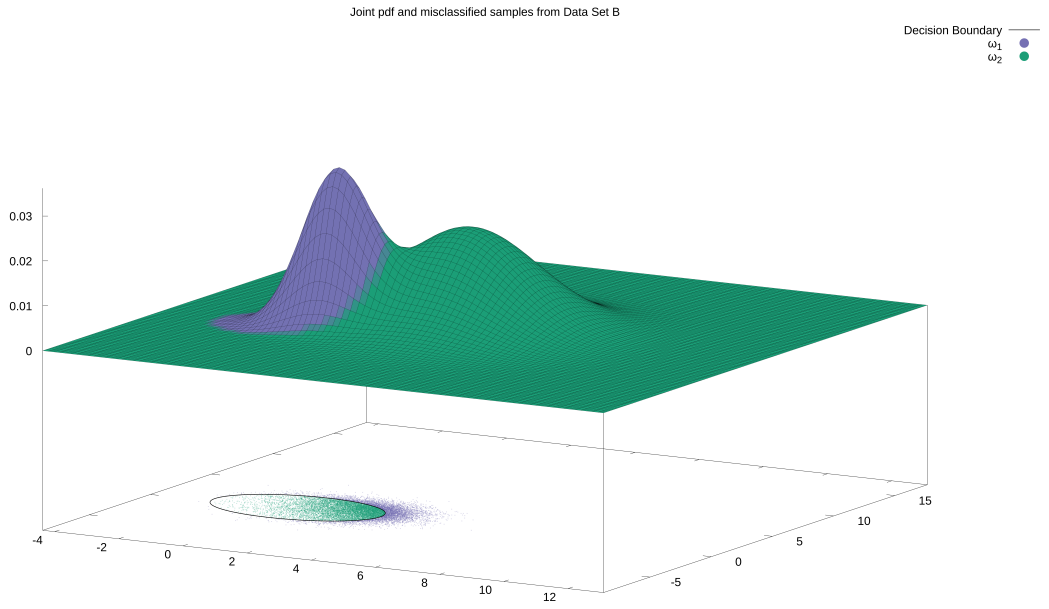

Figure 5: A plot of the misclassified points from Data Set B. Note that each class only has misclassified points on one side of the boundary.

A plot of the error bound function from eq. (17) can be found in fig. 6. The Chernoff bound is imperceptibly lower than the Bhattacharyya bound, but once again β^* is placed correctly at the zero-crossing of $f'(\beta)$.

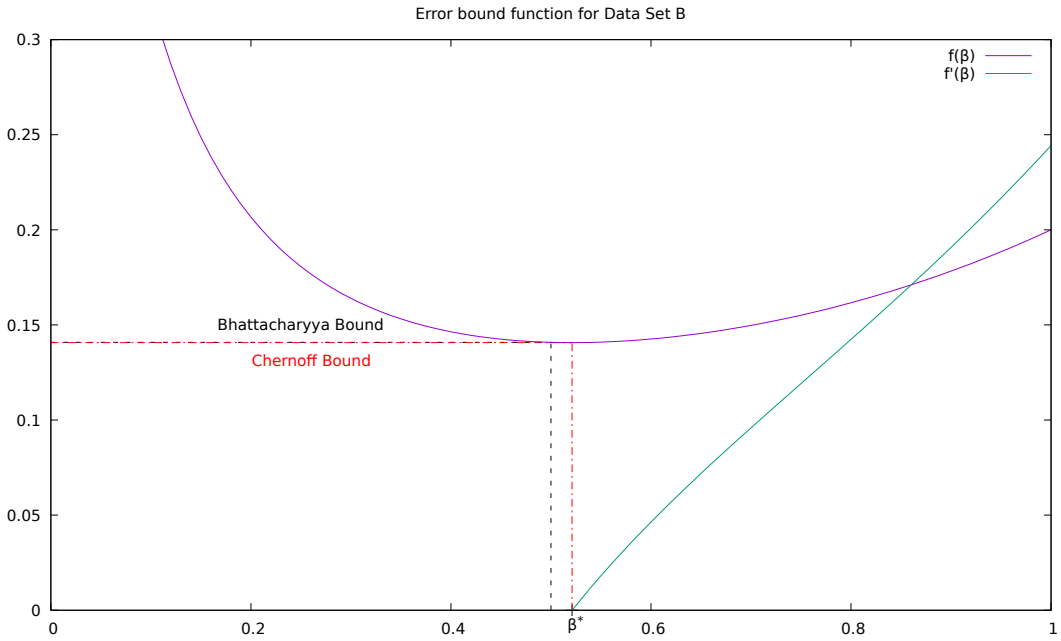

Figure 6: A plot of the error bound function $f(\beta)$ for sample B. Both the Bhattacharyya bound and the Chernoff bound (at β^*) are marked.

2 Parts 3 & 4

2.1 Theory

Continuing on from eq. (8), if the prior probabilities $P(Y = \omega_i)$ are the same for all classes ω_i, then they, the variance σ^2, and the squared norm are all constant (with respect to i) increasing functions (with respect to $g_i(\vec{x})$) and can be removed, leading us to the discriminant

$$g_i(\vec{x}) = -||\vec{x} - \vec{\mu}_i||, \tag{22}$$

the so-called "Euclidean distance" classifier, as it primarily involves using the Euclidean norm $||\cdot||$.

2.2 Implementation

The implementation for these parts are mostly similar to the implementation for the previous parts (as described in section 1.2), except any reference to cases, covariance matrices, and priors were removed, as the Euclidean distance classifier does not need any of these details. The error bounds checking sections were also removed, as they were not required for these parts. Instead, a single discriminant function implementing eq. (22) was used. Decision boundary code was kept the same, except an assumption was made about all priors and variances being equal.

2.3 Results and Discussion

2.3.1 Data Set A

The results of classifying data set A can be found in listing 3. As expected from what was discussed in section 1.3.1, the misclassification rates between classes are very similar, due to the classifier no longer compensating for differences in prior probabilities. This also leads to the misclassification rate being a little bit higher (1.6735% v.s. 1.5445%), so this classifier is not optimum, but the difference in error is still pretty small.

Listing 3: Output from classifying data set A.

```
Classifying data set "A" - 2 classes.
Misclassification rate for class 1:
0.0167

Misclassification rate for class 2:
0.01675

Overall misclassification rate:
0.016735
```

A comparison of the decision boundaries between the Bayes classifier and the Euclidean distance classifier can be found in fig. 7. Note that both decision boundaries are linear, although slightly different. This means that the change in discriminant only has a marginal effect on the misclassification rate. In fig. 8, we can see the additional misclassified points.

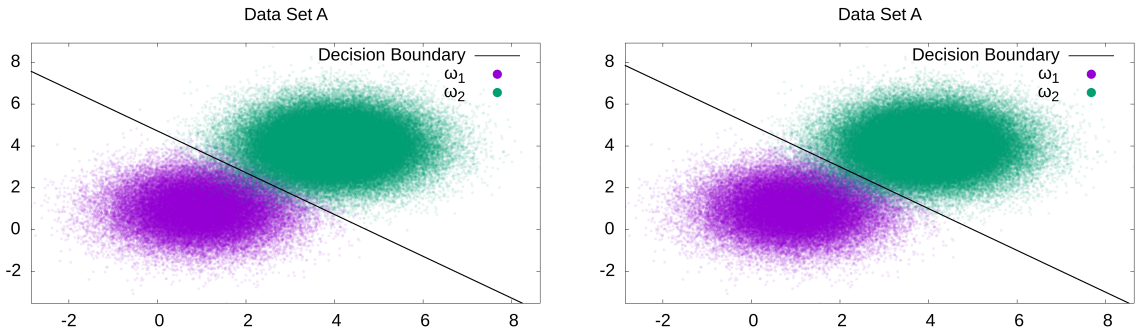

Figure 7: A comparison of the decision boundaries between the Bayes classifier (left) and the Euclidean distance classifier (right) for data set A.

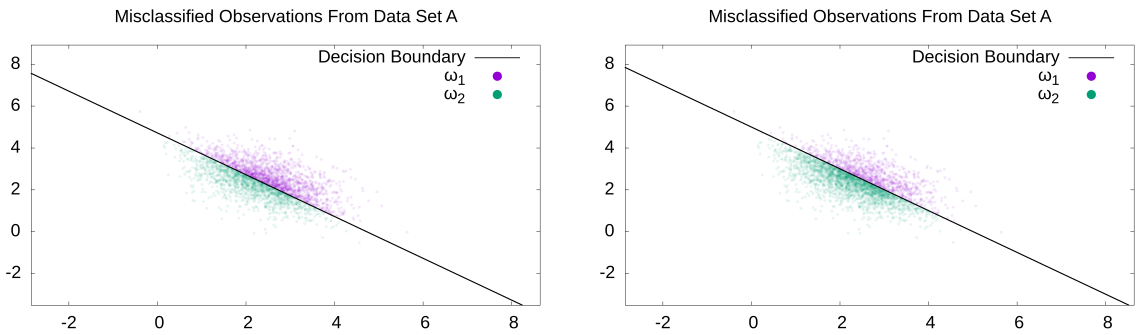

Figure 8: A comparison of the misclassified points from the Bayes classifier (left) and the Euclidean distance classifier (right) for data set A.

2.3.2 Data Set B

The results of classifying data set A can be found in listing 4. This time, there is a big difference in misclassification rates between classes *and* between classifiers (7.322% v.s. 16.075%). This can be attributed to the large difference in decision boundary (linear v.s. elliptical), which can be seen in fig. 9. Therefore, not only is the Euclidean distance classifier not an *optimum* classifier, it is not a good approximation of one either. The additional misclassified points can be seen in fig. 10.

Listing 4: Output from classifying data set B.

```
Classifying data set "B" - 2 classes.
Misclassification rate for class 1:
0.01685

Misclassification rate for class 2:
0.196725

Overall misclassification rate:
0.16075
```

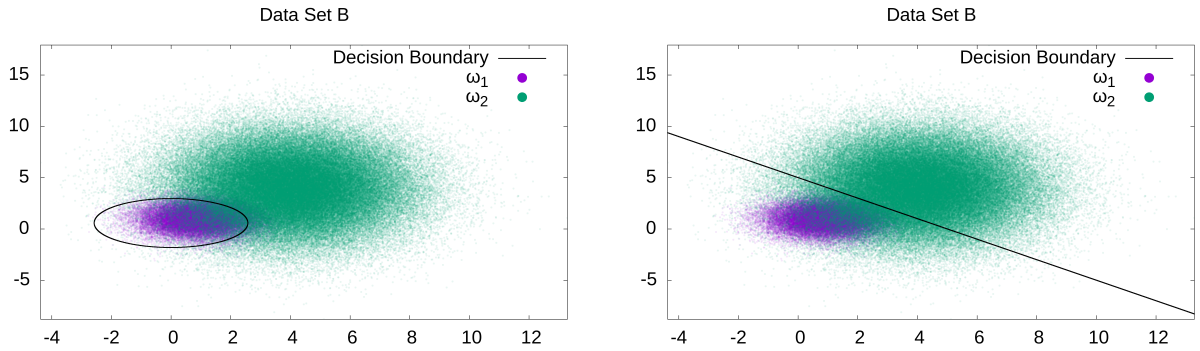

Figure 9: A comparison of the decision boundaries between the Bayes classifier (left) and the Euclidean distance classifier (right) for data set B.

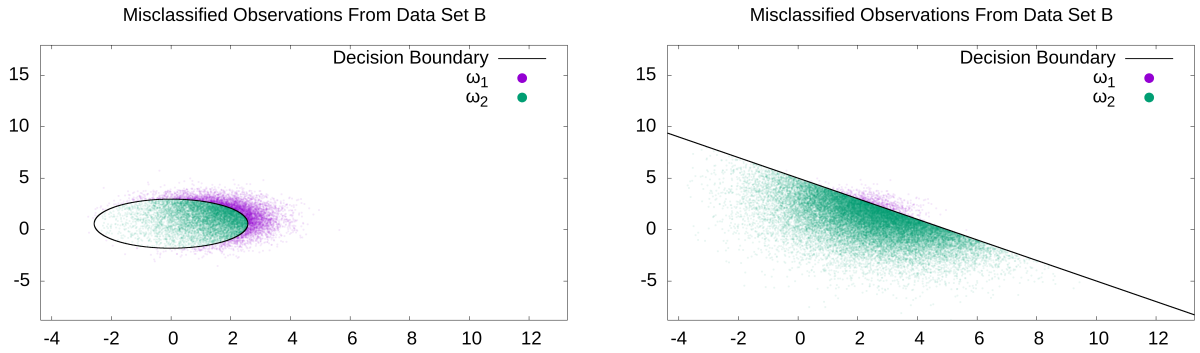

Figure 10: A comparison of the misclassified points from the Bayes classifier (left) and the Euclidean distance classifier (right) for data set B.