CS 189 Introduction to Machine Learning
Spring 2024 Jonathan Shewchuk

HW3

Due: Friday, February 23 at 11:59 pm

- Homework 3 consists of coding assignments and math problems.
- We prefer that you typeset your answers using LaTeX or other word processing software. If you haven't yet learned LaTeX, one of the crown jewels of computer science, now is a good time! Neatly handwritten and scanned solutions will also be accepted.
- In all of the questions, **show your work**, not just the final answer.
- The assignment covers concepts on Gaussian distributions and classifiers. Some of the material may not have been covered in lecture; you are responsible for finding resources to understand it.
- **Start early; you can submit models to Kaggle only twice a day!**

Deliverables:

1. Submit your predictions for the test sets to Kaggle as early as possible. Include your Kaggle scores in your write-up. The Kaggle competition for this assignment can be found at
   - MNIST: https://www.kaggle.com/competitions/cs189-hw3-mnist-spring-2024/
   - SPAM: https://www.kaggle.com/competitions/cs189-hw3-spam-spring-2024/
2. Write-up: Submit your solution in **PDF** format to "Homework 3 Write-Up" in Gradescope.
   - On the first page of your write-up, please list students with whom you collaborated
   - Start each question on a new page. If there are graphs, include those graphs on the same pages as the question write-up. DO NOT put them in an appendix. We need each solution to be self-contained on pages of its own.
   - **Only PDF uploads to Gradescope will be accepted**. You are encouraged use LaTeX or Word to typeset your solution. You may also scan a neatly handwritten solution to produce the PDF.
   - **Replicate all your code in an appendix**. Begin code for each coding question in a fresh page. Do not put code from multiple questions in the same page. When you upload this PDF on Gradescope, *make sure* that you assign the relevant pages of your code from appendix to correct questions.
   - While collaboration is encouraged, *everything* in your solution must be your (and only your) creation. Copying the answers or code of another student is strictly forbidden. Furthermore, all external material (i.e., *anything* outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that consequences of academic misconduct are *particularly severe*!

3. Code: Submit your code as a .zip file to "Homework 3 Code". The code must be in a form that enables the readers to compile (if necessary) and run it to produce your Kaggle submissions.
    - **Set a seed for all pseudo-random numbers generated in your code.** This ensures your results are replicated when readers run your code. For example, you can seed `numpy` with `np.random.seed(42)`.
    - Include a README with your name, student ID, the values of random seed you used, and instructions for compiling (if necessary) and running your code. If the data files need to be anywhere other than the main directory for your code to run, let us know where.
    - Do **not** submit any data files. Supply instructions on how to add data to your code.
    - Code requiring exorbitant memory or execution time might not be considered.
    - Code submitted here must match that in the PDF Write-up. The Kaggle score will not be accepted if the code provided a) does not compile/run or b) runs but does not produce the file submitted to Kaggle.

1 Honor Code

Declare and sign the following statement (Mac Preview, PDF Expert, and FoxIt PDF Reader, among others, have tools to let you sign a PDF file):

"I certify that all solutions are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

Signature: Sajal R.

I did not collaborate with any other students on this assignment.

2 Gaussian Classification

Let $f_{X|Y=C_i}(x) \sim \mathcal{N}(\mu_i, \sigma^2)$ for a two-class, one-dimensional ($d = 1$) classification problem with classes C_1 and C_2, $P(Y = C_1) = P(Y = C_2) = 1/2$, and $\mu_2 > \mu_1$.

1. Find the Bayes optimal decision boundary and the corresponding Bayes decision rule by finding the point(s) at which the posterior probabilities are equal. Use the 0-1 loss function.

2. Suppose the decision boundary for your classifier is $x = b$. The Bayes error is the probability of misclassification, namely,

$$P_e = P((C_1 \text{ misclassified as } C_2) \cup (C_2 \text{ misclassified as } C_1)).$$

   Show that the Bayes error associated with this decision rule, in terms of b, is

$$P_e(b) = \frac{1}{2\sqrt{2\pi}\sigma}\left(\int_{-\infty}^{b} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right) dx + \int_{b}^{\infty} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right) dx\right).$$

3. Using the expression above for the Bayes error, calculate the optimal decision boundary b^* that minimizes $P_e(b)$. How does this value compare to that found in part 1? *Hint: $P_e(b)$ is convex for $\mu_1 < b < \mu_2$.*

(1) Find Bayes optimal decision boundary:

$$P(y = C_1 \mid x) = P(y = C_2 \mid x)$$

$$\frac{P(x \mid y=C_1)\,P(y=C_1)}{P(x)} = \frac{P(x \mid y=C_2)\,P(y=C_2)}{P(x)}$$

★ The two $P(x)$ cancel out

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma}\right)^2\right) \cdot \frac{1}{2} = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma}\right)^2\right) \cdot \frac{1}{2}$$

$$-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma}\right)^2 = -\frac{1}{2}\left(\frac{x-\mu_2}{\sigma}\right)^2$$

$$(x-\mu_1)^2 = (x-\mu_2)^2$$

$$x^2 - 2x\mu_1 + \mu_1^2 = x^2 - 2x\mu_2 + \mu_2^2$$

$$2x(\mu_2 - \mu_1) = \mu_2^2 - \mu_1^2$$

(continued on next page)

$$2x = \frac{(\mu_2+\mu_1)(\mu_2-\mu_1)}{(\mu_2-\mu_1)}$$

$$x = \frac{\mu_2+\mu_1}{2}$$ ✯ This is our Bayes decision boundary

Then, the corresponding Bayes decision rule is:

✯ Classify $X=x$ as belonging to class C_1 if $x < \frac{\mu_1+\mu_2}{2}$

✯ Classify $X=x$ as belonging to class C_2 if $x \geq \frac{\mu_1+\mu_2}{2}$

(2) Based on the Bayes decision rule found in part (1):

Given that $b = \frac{\mu_1+\mu_2}{2}$:

$$P(\text{classified as } c_2 \mid \text{it's } c_1) = \int_{\left(\frac{\mu_1+\mu_2}{2}\right)}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma}\right)^2\right) dx$$

$$= \int_{b}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right) dx$$

$$P(\text{classified as } c_1 \mid \text{it's } c_2) = \int_{-\infty}^{\left(\frac{\mu_1+\mu_2}{2}\right)} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma}\right)^2\right) dx$$

$$= \int_{-\infty}^{b} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right) dx$$

The Bayes error is given as

$$P_e = P((C_1 \text{ misclassified as } C_2) \cup (C_2 \text{ misclassified as } C_1))$$

$$= P(C_1 \text{ misclassified as } C_2) + P(C_2 \text{ misclassified as } C_1)$$

$$= P(C_1 \mid C_2) \cdot P(C_2) + P(C_2 \mid C_1) \cdot P(C_1)$$

$$= (0.5)\, P(C_1 \mid C_2) + (0.5)\, P(C_2 \mid C_1)$$

$$= \frac{1}{2}\left[\int_b^{\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right) dx + \int_{-\infty}^{b} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right) dx\right]$$

$$= \frac{1}{2\sqrt{2\pi}\sigma}\left(\int_{-\infty}^{b} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right) dx + \int_b^{\infty} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right) dx\right) \checkmark$$

(3) To calculate the optimal b^* that minimizes $P_e(b)$, we should compute $\frac{\partial P_e(b)}{\partial b} = 0$ and solve for b:

$$P_e(b) = \frac{1}{2\sqrt{2\pi}\sigma}\left(\int_{-\infty}^{b} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right) dx + \int_b^{\infty} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right) dx\right)$$

$$\frac{\partial P_e(b)}{\partial b} = \frac{1}{2\sqrt{2\pi}\sigma}\left(\exp\left(-\frac{(b-\mu_2)^2}{2\sigma^2}\right) - \exp\left(-\frac{(b-\mu_1)^2}{2\sigma^2}\right)\right) = 0$$

$$\exp\left(-\frac{(b-\mu_2)^2}{2\sigma^2}\right) = \exp\left(-\frac{(b-\mu_1)^2}{2\sigma^2}\right)$$

(continued on next page)

$$-\frac{(b-\mu_2)^2}{2\sigma^2} = -\frac{(b-\mu_1)^2}{2\sigma^2}$$

$$(b-\mu_2)^2 = (b-\mu_1)^2$$

$$b^* = \frac{\mu_2 + \mu_1}{2}$$

* This optimal decision boundary that minimizes $P_e(b)$ is equal to the Bayes optimal decision boundary we found in part 1.

3 Classification and Risk

Suppose we have a classification problem with classes labeled $1, \ldots, c$ and an additional "doubt" category labeled $c+1$. Let $r : \mathbb{R}^d \to \{1, \ldots, c+1\}$ be a decision rule. Define the loss function

$$L(r(x) = i, y = j) = \begin{cases} 0 & \text{if } i = j \quad i, j \in \{1, \ldots, c\}, \\ \lambda_r & \text{if } i = c+1, \\ \lambda_s & \text{otherwise,} \end{cases}$$

where $\lambda_r \geq 0$ is the loss incurred for choosing doubt, $\lambda_s \geq 0$ is the loss incurred for making a misclassification, and at least one of them is nonzero. Hence the risk of classifying a new data point x as class $i \in \{1, 2, \ldots, c+1\}$ is

$$R(r(x) = i|x) = \sum_{j=1}^{c} L(r(x) = i, y = j)\, P(Y = j|x).$$

To be clear, the actual label Y can *never* be $c+1$.

1. Show that the following predictor obtains the minimum risk when $\lambda_r \leq \lambda_s$:
   - Choose class i if $P(Y = i|x) \geq P(Y = j|x)$ for all j and $P(Y = i|x) \geq 1 - \lambda_r/\lambda_s$;
   - Choose doubt otherwise.
2. What happens if $\lambda_r = 0$? What happens if $\lambda_r > \lambda_s$? Explain why this is consistent with what one would expect intuitively.

1 Case 1: $P(y=i \mid x) \geq P(Y=j \mid x)\ \forall j$ and $P(y=i \mid x) \geq 1 - \frac{\lambda_r}{\lambda_s}$

★ We are not choosing the doubt class in this case

$$R(r(x)=i \mid x) = \sum_{j=1}^{c} L(r(x)=i,\ y=j)\, P(Y=j \mid x)$$

$$= \underbrace{\lambda_s (1 - P(y=i \mid x))}_{\text{If we incorrectly classify point } x, \text{ the loss is } \lambda_s} + \underbrace{0 (P(y=i \mid x))}_{\text{If we correctly classify point } x, \text{ the loss is zero}}$$

$$= \lambda_s (1 - P(y=i \mid x))$$

(continued on next page)

Now, introduce some function $f(x)$ which outputs a value between 1 and $c+1$ (f classifies some point x as belonging to some class C_1 or C_2 or ... or C_{c+1}, similar to what $r(x)$ does but it's a different decision rule.

* based on office hours hint

subcase 1: $f(x) = j$ where $j = c+1$

$$R(f(x) = c+1 \mid x) = \lambda_r$$

because f classified x as the doubt category

we can conclude that $R(r(x) = i \mid x) \leq R(f(x) = j \mid x)$

(in other words, the decision rule $r(x)$ obtains minimal risk) because

$$P(y = i \mid x) \geq 1 - \frac{\lambda_r}{\lambda_s}$$

$$\frac{\lambda_r}{\lambda_s} \geq 1 - P(y = i \mid x)$$

$$\lambda_s (1 - P(y = i \mid x)) \leq \lambda_r$$

$$R(r(x) = i \mid x) = \lambda_s (1 - P(y = i \mid x))$$

$$\leq \lambda_r = R(f(x) = j \mid x) \checkmark$$

(continued on next page)

Subcase 2: $f(x) = j$ where $j \neq i$ and $j \neq c+1$

$$R(f(x)=j \mid x) = \lambda_s (1 - P(y=i \mid x))$$

★ since we can't choose the doubt class

we can again conclude that $R(r(x)=i \mid x) \leq R(f(x)=j \mid x)$ because, if we assume that $f(x)$ is some arbitrary decision rule that randomly selects some class j for which $j \neq i, c+1$, then...

$$R(r(x)=i \mid x) = \lambda_s (1 - P(y=i \mid x)) \Rightarrow P(y=i \mid x) = 1 - \frac{R(r(x)=i \mid x)}{\lambda_s}$$

$$R(f(x)=j \mid x) = \lambda_s (1 - P(y=i \mid x)) \Rightarrow P(y=j \mid x) = 1 - \frac{R(f(x)=j \mid x)}{\lambda_s}$$

Since $P(y=i \mid x) \geq P(y=j \mid x)$: ★ Given in the definition of this case

$$1 - \frac{R(r(x)=i \mid x)}{\lambda_s} \geq 1 - \frac{R(f(x)=j \mid x)}{\lambda_s}$$

$$\cancel{\lambda_s} - R(r(x)=i \mid x) \geq \cancel{\lambda_s} - R(f(x)=j \mid x)$$

$$R(r(x)=i \mid x) \leq R(f(x)=j \mid x) \checkmark$$

Therefore, we have shown that for case 1, $r(x)$ is considered to be optimal because $R(r(x)=i \mid x) \leq R(f(x)=j \mid x)$ for some $f(x)$

(continued on next page)

case 2: Choose doubt otherwise $\Rightarrow$ $r(x) = i$ where

$i = c + 1$

$$R(r(x) = i \mid x) = \sum_{j=1}^{c} L(r(x) = i, y = j)\, P(Y = j \mid x)$$

$$= \lambda_r$$

* we're just choosing the loss associated with classifying x as doubt

In this case, $P(y = j \mid x) < 1 - \frac{\lambda_r}{\lambda_s}$

and $P(y = i \mid x) < P(y = j \mid x)$

we can again conclude that $R(r(x) = i \mid x) \leq R(f(x) = j \mid x)$

Subcase 1: $f(x) = j$ where $j \neq c + 1$

$$R(f(x) = j \mid x) = \lambda_s (1 - P(y = j \mid x)) \Rightarrow P(y = j \mid x) = 1 - \frac{R(f(x) = j \mid x)}{\lambda_s}$$

$$P(y = j \mid x) < 1 - \frac{\lambda_r}{\lambda_s}$$

$$\left(1 - \frac{R(f(x) = j \mid x)}{\lambda_s}\right) < 1 - \frac{\lambda_r}{\lambda_s}$$ * $R(r(x) = i \mid x) = \lambda_r$

$$1 - \frac{R(r(x) = i \mid x)}{\lambda_s} > 1 - \frac{R(f(x) = j \mid x)}{\lambda_s}$$

$$R(r(x) = i \mid x) < R(f(x) = j \mid x) \checkmark$$

(continued on next page

Subcase 2: $f(x) = j$ where $j = c + 1$

$$R(f(x) = j \mid x) = \lambda_r$$

Then, $R(r(x) = i \mid x) \leq R(f(x) = j \mid x)$ is arbitrarily true because

$$R(r(x) = i \mid x) = \lambda_r$$
$$= R(f(x) = j \mid x) \checkmark$$

Therefore, we have shown that for case 2, $r(x)$ is considered to be optimal because $R(r(x) = i \mid x) \leq R(f(x) = j \mid x)$ for some $f(x)$

Since $r(x)$ is optimal in all cases, we have shown that our predictor obtains minimum risk when $\lambda_r \leq \lambda_s$.

(2) If $\lambda_r = 0$, then there is no penalty for choosing doubt. Therefore, our classifier will classify a point x in the correct class i if it knows for certain that x belongs to i. Otherwise, it will classify x as belonging to the doubt class because the doubt class has no penalty whereas classifying x as some incorrect class j leads to a nonzero penalty.

If $\lambda_r > \lambda_s$, then there is a higher penalty for choosing doubt than there is for choosing the incorrect class. Intuitively, this means that our classifier will classify a point x in some class i which gives the highest probability of classifying x correctly rather than choosing the doubt class because that minimizes the penalty which we experience.

4 Maximum Likelihood Estimation and Bias

Let $X_1, \ldots, X_n \in \mathbb{R}$ be n sample points drawn independently from univariate normal distributions such that $X_i \sim \mathcal{N}(\mu, \sigma_i^2)$, where $\sigma_i = \sigma/\sqrt{i}$ for some parameter σ. (Every sample point comes from a distribution with a **different variance**.) Note the word "univariate"; we are working in dimension $d = 1$, and each "point" is just a real number.

1. Derive the maximum likelihood estimates, denoted $\hat{\mu}$ and $\hat{\sigma}$, for the mean μ and the parameter σ. (The formulae from class don't apply here, because every point has a different variance.) You may write an expression for $\hat{\sigma}^2$ rather than $\hat{\sigma}$ if you wish—it's probably simpler that way. Show all your work.

2. Given the true value of a statistic θ and an estimator $\hat{\theta}$ of that statistic, we define the *bias* of the estimator to be the the expected difference from the true value. That is,

$$\text{bias}(\hat{\theta}) = \mathbb{E}[\hat{\theta}] - \theta.$$

   We say that an estimator is *unbiased* if its bias is 0.

   Either prove or disprove the following statement: *The MLE sample estimator $\hat{\mu}$ is unbiased. Hint: Neither the true μ nor true σ^2 are known when estimating sample statistics, thus we need to plug in appropriate estimators.*

3. Either prove or disprove the following statement: *The MLE sample estimator $\hat{\sigma}^2$ is unbiased. Hint: Neither the true μ nor true σ^2 are known when estimating sample statistics, thus we need to plug in appropriate estimators.*

4. Suppose the Variance Fairy drops by to give us the true value of σ^2, so that we only have to estimate μ. Given the loss function $L(\hat{\mu}, \mu) = (\hat{\mu} - \mu)^2$, what is the risk of our MLE estimator $\hat{\mu}$?

(1) To find the maximum likelihood estimation $\mathcal{L}$:

$$\mathcal{L}(\mu, \sigma; x_1, x_2, \ldots x_n) = f(x_1) f(x_2) \cdots f(x_n)$$

$$= \prod_{i=1}^{n} f(x_i)$$

where $x_i \sim N(\mu, \sigma_i^2)$

$$f(x_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma_i^2}\right) \quad * \sigma_i = \frac{\sigma}{\sqrt{i}}$$

$$= \frac{\sqrt{i}}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i-\mu)^2 i}{2\sigma^2}\right)$$

(continued on next page)

$$\ell(\mu, \sigma; x_1, \ldots, x_n) = \ln f(x_1) + \ln f(x_2) + \cdots + \ln f(x_n)$$

$$= \sum_{i=1}^{n} \ln f(x_i)$$

$$= \sum_{i=1}^{n} \ln\left(\frac{\sqrt{i}}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x_i-\mu)^2 i}{2\sigma^2}\right)\right)$$

$$= \sum_{i=1}^{n} \ln\left(\frac{\sqrt{i}}{\sigma\sqrt{2\pi}}\right) + \ln\left(\exp\left(-\frac{(x_i-\mu)^2 i}{2\sigma^2}\right)\right)$$

$$= \sum_{i=1}^{n} \ln\left(\frac{\sqrt{i}}{\sigma\sqrt{2\pi}}\right) - \sum_{i=1}^{n} \frac{(x_i-\mu)^2 i}{2\sigma^2}$$

To derive $\hat{\mu}$: solve for $\frac{\partial \ell}{\partial \mu} = 0$

$$\frac{\partial \ell}{\partial \mu} = \frac{\partial}{\partial \mu}\left(\sum_{i=1}^{n} \underbrace{\ln\left(\frac{\sqrt{i}}{\sigma\sqrt{2\pi}}\right)}_{\frac{\partial}{\partial \mu} = 0} - \sum_{i=1}^{n} \underbrace{\frac{(x_i-\mu)^2 i}{2\sigma^2}}_{\frac{\partial}{\partial \mu} = \frac{-2(x_i-\mu)i}{2\sigma^2}}\right)$$

$$= 0 - \sum_{i=1}^{n} \frac{-\cancel{2}(x_i-\mu)i}{\cancel{2}\sigma^2}$$

$$= \sum_{i=1}^{n} \frac{(x_i-\mu)i}{\sigma^2} = 0$$

(continued on next page)

$$\sum_{i=1}^{n} (x_i - \mu)\, i = 0$$

$$\sum_{i=1}^{n} x_i \cdot i = \sum_{i=1}^{n} \mu \cdot i = \mu \sum_{i=1}^{n} i$$

$$\hat{\mu} = \frac{\sum_{i=1}^{n} x_i \cdot i}{\sum_{i=1}^{n} i}$$

To derive $\hat{\sigma}$: solve for $\frac{\partial \ell}{\partial \sigma} = 0$

$$\frac{\partial \ell}{\partial \sigma} = \frac{\partial}{\partial \sigma}\left(\sum_{i=1}^{n} \ln\left(\frac{\sqrt{i}}{\sigma\sqrt{2\pi}}\right) - \sum_{i=1}^{n} \frac{(x_i-\mu)^2 i}{2\sigma^2} \right)$$

$$= \sum_{i=1}^{n} \frac{\partial}{\partial \sigma}\left(\ln(\sqrt{i}) - \ln(\sigma) - \ln(\sqrt{2\pi})\right) - \sum_{i=1}^{n} \frac{\partial}{\partial \sigma}\left(\frac{(x_i-\mu)^2 i}{2\sigma^2}\right)$$

$\frac{\partial}{\partial \sigma} = 0$ $\quad \frac{\partial}{\partial \sigma} = \frac{1}{\sigma}$ $\quad \frac{\partial}{\partial \sigma} = 0$ $\quad \frac{\partial}{\partial \sigma} = \frac{(x_i-\mu)^2 i}{2}(-2\sigma^{-3})$

$$= \sum_{i=1}^{n} -\frac{1}{\sigma} + \sum_{i=1}^{n} \frac{(x_i-\mu)^2 i}{\sigma^3} = 0$$

$\sum_{i=1}^{n} -\frac{1}{\sigma} = -\frac{n}{\sigma}$

$$\frac{n}{\sigma} = \sum_{i=1}^{n} \frac{(x_i-\mu)^2 i}{\sigma^3}$$

(continued on next page)

$$n = \sigma \cdot \frac{1}{\sigma^3} \sum_{i=1}^{n} (x_i - \mu)^2 i$$

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{\mu})^2 i$$

(2) $\text{bias}(\hat{\mu}) = E(\hat{\mu}) - \mu$

$$= E\left(\frac{\sum_{i=1}^{n} x_i \cdot i}{\sum_{i=1}^{n} i} \right) - \mu$$

$$= \frac{1}{\sum_{i=1}^{n} i} E\left(\sum_{i=1}^{n} x_i \cdot i \right) - \mu$$

$$= \frac{1}{\sum_{i=1}^{n} i} \sum_{i=1}^{n} i \cdot \underbrace{E(x_i)}_{=\mu} - \mu$$

$$= \frac{\sum_{i=1}^{n} i}{\sum_{i=1}^{n} i} \mu - \mu$$

$$= \mu - \mu = 0 \Rightarrow$$

The MLE sample estimator $\hat{\mu}$ is unbiased

3 $\text{bias}(\hat{\sigma}^2) = E(\hat{\sigma}^2) - \sigma^2$

$$= E\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{\mu})^2 i\right) - \sigma^2$$

$$(x_i - \hat{\mu})^2 = x_i^2 - 2x_i\hat{\mu} + \hat{\mu}^2$$

$$= \frac{1}{n} E\left(\sum_{i=1}^{n} x_i^2 i - 2\sum_{i=1}^{n} x_i \hat{\mu} i + \sum_{i=1}^{n} \hat{\mu}^2 i\right) - \sigma^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} E(x_i^2) i - \frac{2}{n}\sum_{i=1}^{n} \underbrace{E(x_i \hat{\mu})}_{=\hat{\mu}} i + \frac{1}{n}\sum_{i=1}^{n} E(\hat{\mu}^2) i - \sigma^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} E(x_i^2) i - \frac{2}{n}\sum_{i=1}^{n} E(\hat{\mu}^2) i + \frac{1}{n}\sum_{i=1}^{n} E(\hat{\mu}^2) i - \sigma^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} E(x_i^2) i - \frac{1}{n} E(\hat{\mu}^2) \underbrace{\sum_{i=1}^{n} i}_{=\frac{n(n+1)}{2}} - \sigma^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} E(x_i^2) i - \frac{(n+1)}{2} E(\hat{\mu}^2) - \sigma^2$$

(continued on next page)

We know that $\text{var}(x) = E(x^2) - (E(x))^2$, so...

$$\text{var}(\hat{\mu}) = E(\hat{\mu}^2) - (E(\hat{\mu}))^2$$

$$\text{var}(\hat{\mu}) = \text{var}\left(\frac{\sum_{i=1}^{n} x_i \cdot i}{\sum_{i=1}^{n} i} \right) = \left(\frac{2}{n(n+1)} \right)^2 \text{var}\left(\sum_{i=1}^{n} x_i \cdot i \right)$$

$$= \left(\frac{2}{n(n+1)} \right)^2 \sum_{i=1}^{n} i^2 \, \underbrace{\text{var}(x_i)}_{= \sigma_i^2 = \frac{\sigma^2}{i}}$$

$$= \left(\frac{2}{n(n+1)} \right)^2 \sum_{i=1}^{n} i^2 \frac{\sigma^2}{i}$$

$$= \left(\frac{2}{n(n+1)} \right)^2 \sigma^2 \underbrace{\sum_{i=1}^{n} i}_{= \frac{n(n+1)}{2}}$$

$$= \frac{2}{n(n+1)} \sigma^2$$

$$\text{var}(\hat{\mu}) = E(\hat{\mu}^2) - (E(\hat{\mu}))^2$$

$$E(\hat{\mu}^2) = \frac{2}{n(n+1)} \sigma^2 + \mu^2$$

(continued on next page)

$$\underbrace{\text{Var}(x_i)}_{=\frac{\sigma^2}{i}} = E(x_i^2) - \underbrace{(E(x_i))^2}_{=\mu}$$

$$E(x_i^2) = \frac{\sigma^2}{i} + \mu^2$$

$$\text{bias}(\hat{\sigma}^2) = \frac{1}{n}\sum_{i=1}^{n} E(x_i^2)\,i - \frac{(n+1)}{2}E(\hat{\mu}^2) - \sigma^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{\sigma^2}{i}\,i + \mu^2 i - \frac{(n+1)}{2}\left(\frac{2}{n(n+1)}\sigma^2 + \mu^2\right) - \sigma^2$$

$$= \frac{1}{n}\underbrace{\sum_{i=1}^{n}\sigma^2}_{=n\sigma^2} + \mu^2\frac{1}{n}\sum_{i=1}^{n} i - \frac{(n+1)\,2}{2n(n+1)}\sigma^2 + \frac{(n+1)}{2}\mu^2 - \sigma^2$$

$$= \sigma^2 - \frac{(n+1)}{2}\mu^2 - \frac{\sigma^2}{n} + \frac{(n+1)}{2}\mu^2 - \sigma^2$$

$$= -\frac{\sigma^2}{n}$$

Since $\text{bias}(\hat{\sigma}^2) \neq 0$, our estimator is biased.

4

$$R(\hat{\mu}) = E(L(\hat{\mu}, \mu))$$

$$= E((\hat{\mu} - \mu)^2)$$

$$= E(\hat{\mu}^2 - 2\hat{\mu}\mu + \mu^2)$$

$$= E(\hat{\mu}^2) - 2E(\hat{\mu}\mu) + E(\mu^2)$$

$$= \frac{2}{n(n+1)}\sigma^2 + \mu^2 - 2\mu E(\hat{\mu}) + \mu^2$$

$$= \frac{2}{n(n+1)}\sigma^2 + \underbrace{\mu^2 - 2\mu\cdot\mu + \mu^2}_{= 0}$$

$$= \frac{2\sigma^2}{n(n+1)}$$

$$R(\hat{\mu}) = \frac{2}{n(n+1)}\sigma^2$$

5 Covariance Matrices and Decompositions

As described in lecture, the covariance matrix $\text{Var}(R) \in \mathbb{R}^{d \times d}$ for a random variable $R \in \mathbb{R}^d$ with mean $\mu \in \mathbb{R}^d$ is

$$\text{Var}(R) = \text{Cov}(R, R) = \mathbb{E}[(R - \mu)(R - \mu)^\top] = \begin{bmatrix} \text{Var}(R_1) & \text{Cov}(R_1, R_2) & \dots & \text{Cov}(R_1, R_d) \\ \text{Cov}(R_2, R_1) & \text{Var}(R_2) & & \text{Cov}(R_2, R_d) \\ \vdots & & \ddots & \vdots \\ \text{Cov}(R_d, R_1) & \text{Cov}(R_d, R_2) & \dots & \text{Var}(R_d) \end{bmatrix},$$

where $\text{Cov}(R_i, R_j) = \mathbb{E}[(R_i - \mu_i)(R_j - \mu_j)]$ and $\text{Var}(R_i) = \text{Cov}(R_i, R_i)$.

If the random variable R is sampled from the multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$ with the PDF

$$f(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-((x-\mu)^\top \Sigma^{-1}(x-\mu))/2},$$

then $\text{Var}(R) = \Sigma$.

Given n points $X_1, X_2, \dots, X_n$ sampled from $\mathcal{N}(\mu, \Sigma)$, we can estimate Σ with the maximum likelihood estimator

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} (X_i - \hat{\mu})(X_i - \hat{\mu})^\top,$$

which is also known as the *sample covariance matrix*.

1. The estimate $\hat{\Sigma}$ makes sense as an approximation of Σ only if $\hat{\Sigma}$ is invertible. Under what circumstances is $\hat{\Sigma}$ not invertible? *Express your answer in terms of the geometric arrangement of the sample points X_i.* We want a geometric characterization, not an algebraic one. Make sure your answer is complete; i.e., it includes all cases in which the covariance matrix of the sample is singular.

2. Suggest a way to fix a singular covariance matrix estimator $\hat{\Sigma}$ by replacing it with a similar but invertible matrix. Your suggestion may be a kludge, but it should not change the covariance matrix too much. Note that infinitesimal numbers do not exist; if your solution uses a very small number, explain how to calculate a number that is sufficiently small for your purposes.

3. Consider the normal distribution $\mathcal{N}(0, \Sigma)$ with mean $\mu = 0$. Consider all vectors of length 1; i.e., any vector x for which $\|x\| = 1$. Which vector(s) x of length 1 maximizes the PDF $f(x)$? Which vector(s) x of length 1 minimizes $f(x)$? Your answers should depend on the properties of Σ. Explain your answer.

4. Suppose we have $X \sim \mathcal{N}(0, \Sigma)$, $X \in \mathbb{R}^n$ and a unit vector $y \in \mathbb{R}^n$. We can compute the projection of the random vector X onto a unit direction vector y as $p = y^\top X$. First, compute the variance of p. Second, with this information, what does the largest eigenvalue $\lambda_{\max}$ of the covariance matrix tell us about the variances of expressions of the form $y^\top X$?

1. $\hat{\Sigma}$ is not invertible, in terms of a geometric characterization, if all of the sample points x_i lie on the same hyperplane in some given feature space instead of spanning the feature space because this could imply that there exists dependence within the data (i.e., the covariance matrix could be singular)

2. One way we can fix a singular covariance matrix to make it invertible is by redefining $\tilde{\hat{\Sigma}} = \hat{\Sigma} + cI$ where I is the identity matrix and c is some small positive constant. Doing so makes all eigenvalues of $\tilde{\hat{\Sigma}}$ (which are just the diagonal elements of the matrix) positive; therefore, $\tilde{\hat{\Sigma}}$ will be invertible because a symmetric matrix is invertible if it has no zero eigenvalue and $\tilde{\hat{\Sigma}}$ is symmetric. We can calculate a c small enough for our purposes by using hyperparameter tuning.

(3) If we have $\mu = 0$, then the PDF of $X \sim N(0, \Sigma)$ is given by:

$$f(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{((x-\mu)^T \Sigma^{-1} (x-\mu))}{2}\right)$$

$$= \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{x^T \Sigma^{-1} x}{2}\right)$$

Therefore, the x where $\|x\| = 1$ that maximizes $f(x)$ will be the x that maximizes $-\frac{x^T \Sigma^{-1} x}{2}$, which is the same x that minimizes $x^T \Sigma^{-1} x$

Therefore, x should be the normalized eigenvector corresponding to the maximum eigenvalue of Σ (aka the minimum eigenvalue of Σ^{-1}). As a result, $x^T \Sigma^{-1} x$ will be minimized, maximizing $\frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{x^T \Sigma^{-1} x}{2}\right)$.

On the other hand, the x that minimizes $f(x)$ maximizes $x^T \Sigma^{-1} x$, which will be the normalized eigenvector corresponding to the minimum eigenvalue of Σ (max eigenvalue of Σ^{-1}). As a result, $x^T \Sigma^{-1} x$ will be maximized, minimizing $\frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{x^T \Sigma^{-1} x}{2}\right)$.

4

$$\text{var}(p) = \text{var}(y^T x)$$

$$= y^T \text{var}(x)\, y$$

$$= y^T \Sigma y$$

* we're given that $\text{var}(x) = \Sigma$

Given that $\text{var}(p) = y^T \Sigma y$, the largest eigenvalue λ_{max} of the covariance matrix tells us when $\text{var}(p)$ is maximized. $\text{var}(y^T X)$ is maximized when y is the normalized (since $\|y\|_2 = 1$) eigenvector of the covariance matrix that corresponds to the eigenvalue λ_{max} because this value of y maximizes $y^T \Sigma y$ (as discussed in part 3).

6 Isocontours of Normal Distributions

Let $f(\mu, \Sigma)$ be the probability density function of a normally distributed random variable in $\mathbb{R}^2$. Write code to plot the isocontours of the following functions, each on its own separate figure. Make sure it is clear which figure belongs to which part. You're free to use any plotting libraries or stats utilities you like; for instance, in Python you can use Matplotlib and SciPy. Choose the boundaries of the domain you plot large enough to show the interesting characteristics of the isocontours (use your judgment). Make sure we can tell what isovalue each contour is associated with—you can do this with labels or a colorbar/legend.

1. $f(\mu, \Sigma)$, where $\mu = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$.
2. $f(\mu, \Sigma)$, where $\mu = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$.
3. $f(\mu_1, \Sigma_1) - f(\mu_2, \Sigma_2)$, where $\mu_1 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$ and $\Sigma_1 = \Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$.

   (Yes, this is a difference between two PDFs. No, it is not itself a valid PDF. Just plot its isocontours anyway.)
4. $f(\mu_1, \Sigma_1) - f(\mu_2, \Sigma_2)$, where $\mu_1 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$ and $\Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$.
5. $f(\mu_1, \Sigma_1) - f(\mu_2, \Sigma_2)$, where $\mu_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ and $\Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.

7 Eigenvectors of the Gaussian Covariance Matrix

Consider two one-dimensional random variables $X_1 \sim \mathcal{N}(3, 9)$ and $X_2 \sim \frac{1}{2}X_1 + \mathcal{N}(4, 4)$, where $\mathcal{N}(\mu, \sigma^2)$ is a Gaussian distribution with mean μ and variance σ^2. (This means that you have to draw X_1 first and use it to compute a random X_2.)

Write a program that draws $n = 100$ random two-dimensional sample points from (X_1, X_2). For each sample point, the value of X_2 is a function of the value of X_1 for that *same* sample point, but the sample points are independent of each other. In your code, make sure to choose and set a fixed random number seed for whatever random number generator you use, so your simulation is reproducible, and document your choice of random number seed and random number generator in your write-up. For each of the following parts, include the corresponding output of your program.

1. Compute the mean (in $\mathbb{R}^2$) of the sample.
2. Compute the 2×2 covariance matrix of the sample (based on the sample mean, not the true mean—which you would not know given real-world data).
3. Compute the eigenvectors and eigenvalues of this covariance matrix.
4. On a two-dimensional grid with a horizonal axis for X_1 with range $[-15, 15]$ and a vertical axis for X_2 with range $[-15, 15]$, plot

   (i) all $n = 100$ data points, and

   (ii) arrows representing both covariance eigenvectors. The eigenvector arrows should originate at the mean and have magnitudes equal to their corresponding eigenvalues.

   Hint: make *sure* your plotting software is set so the figure is square (i.e., the horizontal and vertical scales are the same). Not doing that may lead to hours of frustration!
5. Let $U = [v_1 \quad v_2]$ be a 2×2 matrix whose columns are the **unit** eigenvectors of the covariance matrix, where v_1 is the eigenvector with the larger eigenvalue. We use $U^\top$ as a rotation matrix to rotate each sample point from the (X_1, X_2) coordinate system to a coordinate system aligned with the eigenvectors. (As $U^\top = U^{-1}$, the matrix U reverses this rotation, moving back from the eigenvector coordinate system to the original coordinate system). *Center* your sample points by subtracting the mean μ from each point; then rotate each point by $U^\top$, giving $x_{\text{rotated}} = U^\top(x - \mu)$. Plot these rotated points on a new two dimensional-grid, again with both axes having range $[-15, 15]$. (You are not required to plot the eigenvectors, which would be horizontal and vertical.)

In your plots, **clearly label the axes and include a title**. Moreover, **make sure the horizontal and vertical axis have the same scale!** The aspect ratio should be one.

8 Gaussian Classifiers for Digits and Spam

In this problem, you will build classifiers based on Gaussian discriminant analysis. Unlike Homework 1, you are NOT allowed to use any libraries for out-of-the-box classification (e.g. `sklearn`). You may use anything in `numpy` and `scipy`.

The training and test data can be found with this homework. **Do NOT use the training/test data from Homework 1, as they have changed for this homework.** The starter code is similar to HW1's; we provide `check.py` and `save_csv.py` files for you to produce your Kaggle submission files. Submit your predicted class labels for the test data on the Kaggle competition website and be sure to include your Kaggle display name and scores in your writeup. Also be sure to include an appendix of your code at the end of your writeup.

Reminder: please also select relevant code from the appendix on Gradescope for your answer to each question.

1. Taking pixel values as features (no new features yet, please), fit a Gaussian distribution to each digit class using maximum likelihood estimation. This involves computing a mean and a covariance matrix for each digit class, as discussed in Lecture 9 and Section 4.4 of *An Introduction to Statistical Learning*. Attach the relevant code as your answer to this part.

   Hint: You may, and probably should, contrast-normalize the images before using their pixel values. One way to normalize is to divide the pixel values of an image by the l_2-norm of its pixel values.

2. (Written answer + graph) Visualize the covariance matrix for a particular class (digit). Tell us which digit and include your visualization in your write-up. How do the diagonal terms compare with the off-diagonal terms? What do you conclude from this?

3. Classify the digits in the test set on the basis of posterior probabilities with two different approaches.

   (a) (Graphs) Linear discriminant analysis (LDA). Model the class conditional probabilities as Gaussians $\mathcal{N}(\mu_C, \Sigma)$ with different means μ_C (for class C) and the same pooled within-class covariance matrix Σ, which you compute from a weighted average of the 10 covariance matrices from the 10 classes, as described in Lecture 9.

   In your implementation, you might run into issues of determinants overflowing or underflowing, or normal PDF probabilities underflowing. These problems might be solved by learning about `numpy.linalg.slogdet` and/or `scipy.stats.multivariate_normal.logpdf`.

   To implement LDA, you will sometimes need to compute a matrix-vector product of the form $\Sigma^{-1}x$ for some vector x. You should **not** compute the inverse of Σ (nor the determinant of Σ) as it is not guaranteed to be invertible. Instead, you should find a way to solve the implied linear system without computing the inverse.

   Hold out 10,000 randomly chosen training points for a validation set. (You may reuse your Homework 1 solution or an out-of-the-box library for dataset splitting *only*.)

Classify each image in the validation set into one of the 10 classes. Compute the error rate ($1 - \frac{\text{\# points correctly classified}}{\text{\# total points}}$) on the validation set and plot it over the following numbers of randomly chosen training points: 100, 200, 500, 1,000, 2,000, 5,000, 10,000, 30,000, 50,000. (Expect unpredictability in your error rate when few training points are used.)

(b) (Graphs) Quadratic discriminant analysis (QDA). Model the class conditional probabilities as Gaussians $\mathcal{N}(\mu_C, \Sigma_C)$, where Σ_C is the estimated covariance matrix for class C. (If any of these covariance matrices turn out singular, implement the trick you described in Q7(b). You are welcome to use validation to choose the right constant(s) for that trick.) Repeat the same tests and error rate calculations you did for LDA.

(c) (Written answer) Which of LDA and QDA performed better? Why?

(d) (Written answer + graph) Include a plot of validation error versus the number of training points for each digit. Plot all the 10 curves on the same graph as shown in Figure 1. Which digit is easiest to classify? Write down your answer and suggest why you think it's the easiest digit.

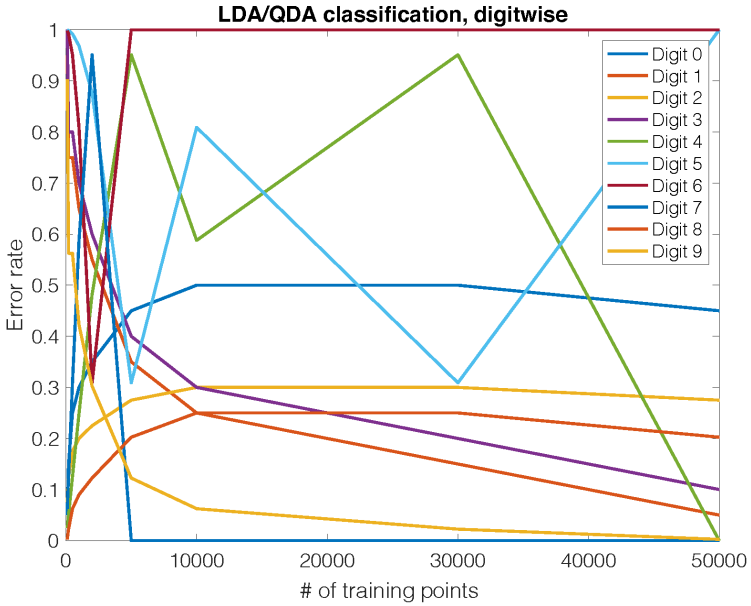


Figure 1: Sample graph with 10 plots

4. (Written answer) With `mnist-data-hw3.npz`, train your best classifier for the `training_data` and classify the images in the `test_data`. Submit your labels to the online Kaggle competition. Record your optimum prediction rate in your write-up and include your Kaggle username. Don't forget to use the "submissions" tab or link on Kaggle to select your best submission!

   You are welcome to compute extra features for the Kaggle competition, as long as they do not use an exterior learned model for their computation (no transfer learning!). If you do so, please describe your implementation in your assignment. Please use extra features **only** for the Kaggle portion of the assignment.

5. (Written answer) Next, apply LDA or QDA (your choice) to spam (`spam-data-hw3.npz`). Submit your test results to the online Kaggle competition. Record your optimum prediction rate in your submission. If you use additional features (or omit features), please describe them. We include a `featurize.py` file (similar to HW1's) that you may modify to create new features.

   Optional: If you use the defaults, expect relatively low classification rates. We suggest using a Bag-Of-Words model. You are encouraged to explore alternative hand-crafted features, and are welcome to use any third-party library to implement them, as long as they do not use a separate model for their computation (no large language models, BERT, or word2vec!).

Submission Checklist

Please ensure you have completed the following before your final submission.

At the beginning of your writeup...

1. Have you copied and hand-signed the honor code specified in Question 1?
2. Have you listed all students (Names and ID numbers) that you collaborated with?

In your writeup for Question 8...

1. Have you included your **Kaggle Score** and **Kaggle Username** for **both** questions 8.4 and 8.5?

At the end of the writeup...

1. Have you provided a code appendix including all code you wrote in solving the homework?
2. Have you included featurize.py in your code appendix if you modified it?

Executable Code Submission

1. Have you created an archive containing all ".py" files that you wrote or modified to generate your homework solutions (including featurize.py if you modified it)?
2. Have you removed all data and extraneous files from the archive?
3. Have you included a README file in your archive briefly describing how to run your code on the test data and reproduce your Kaggle results?

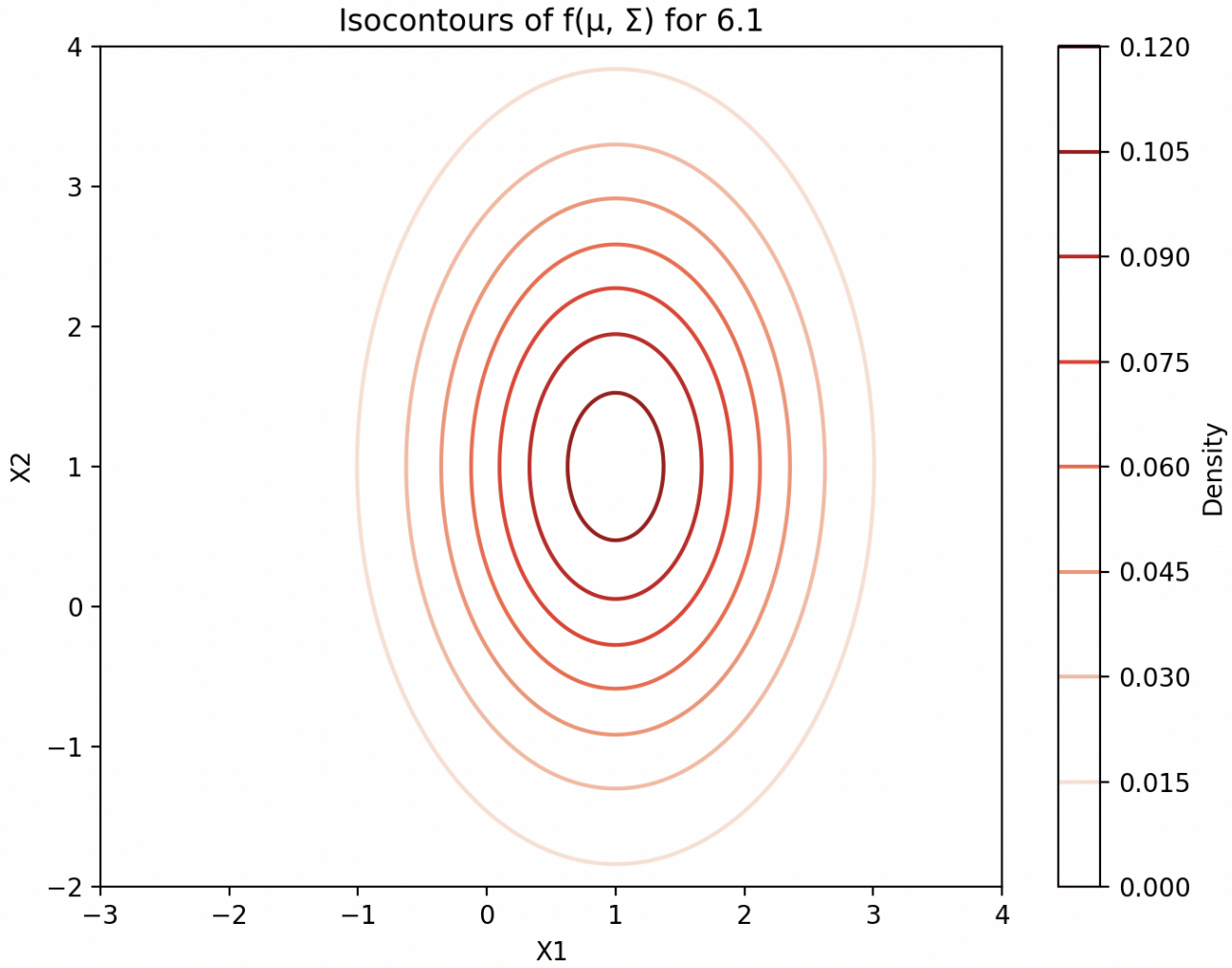
Submissions

1. Have you submitted your test set predictions for both **MNIST** and **SPAM** to the appropriate Kaggle challenges?
2. Have you submitted your written solutions to the Gradescope assignment titled **HW3 Write-Up** and selected pages appropriately?
3. Have you submitted your executable code archive to the Gradescope assignment titled **HW3 Code**?

Congratulations! You have completed Homework 3.

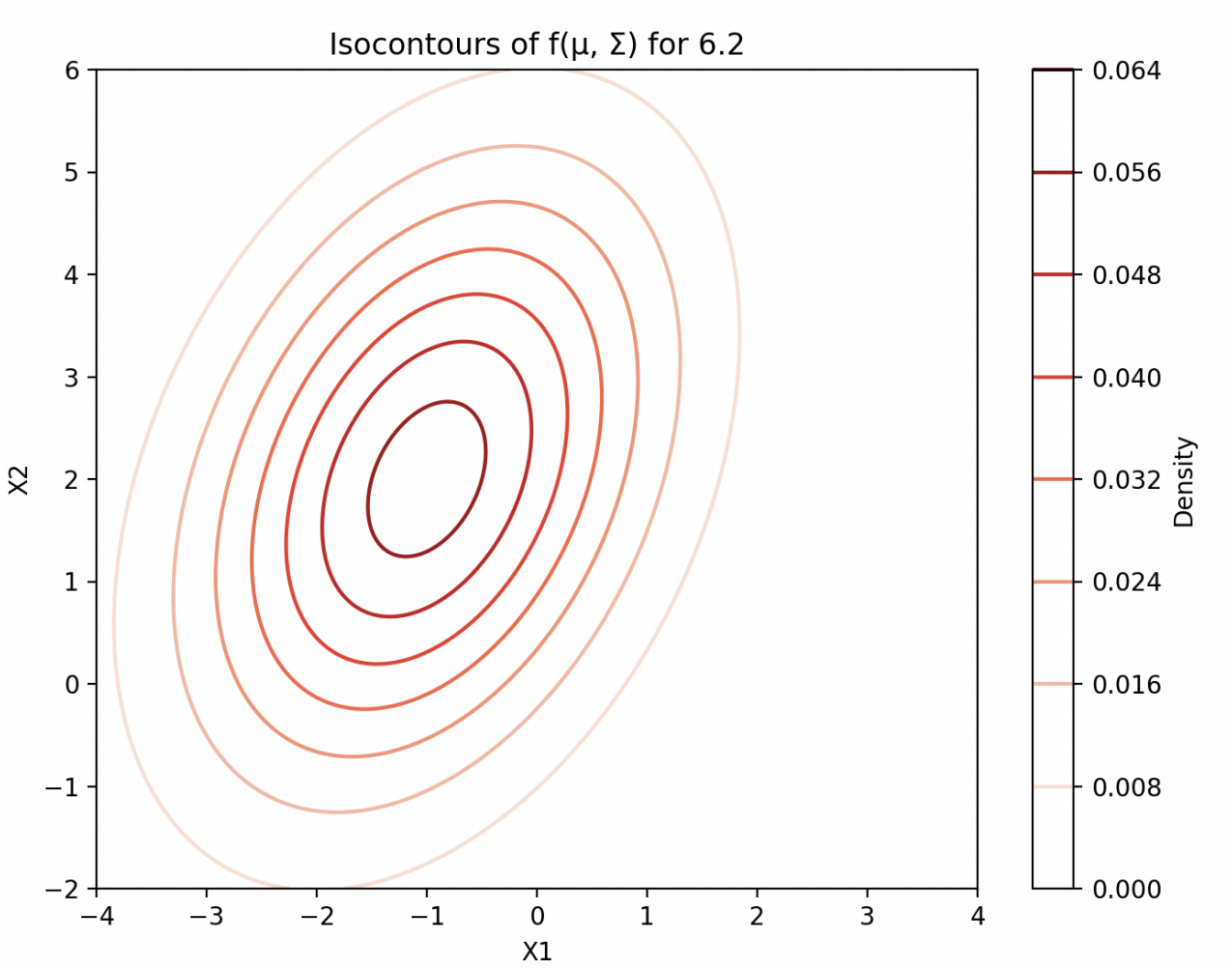
Problem 6 Deliverables:

1.



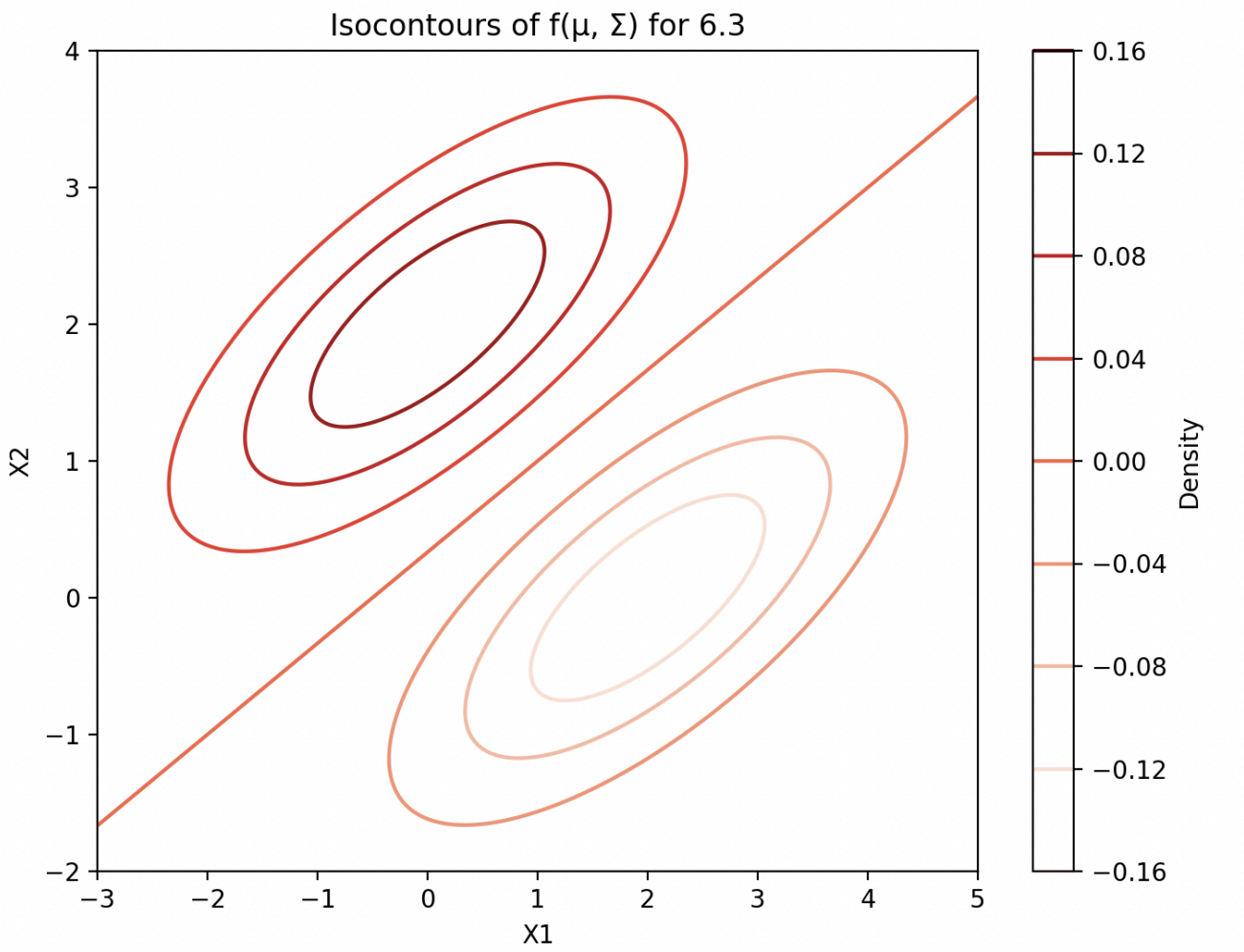
The code for this problem is located in the code appendix.

2.



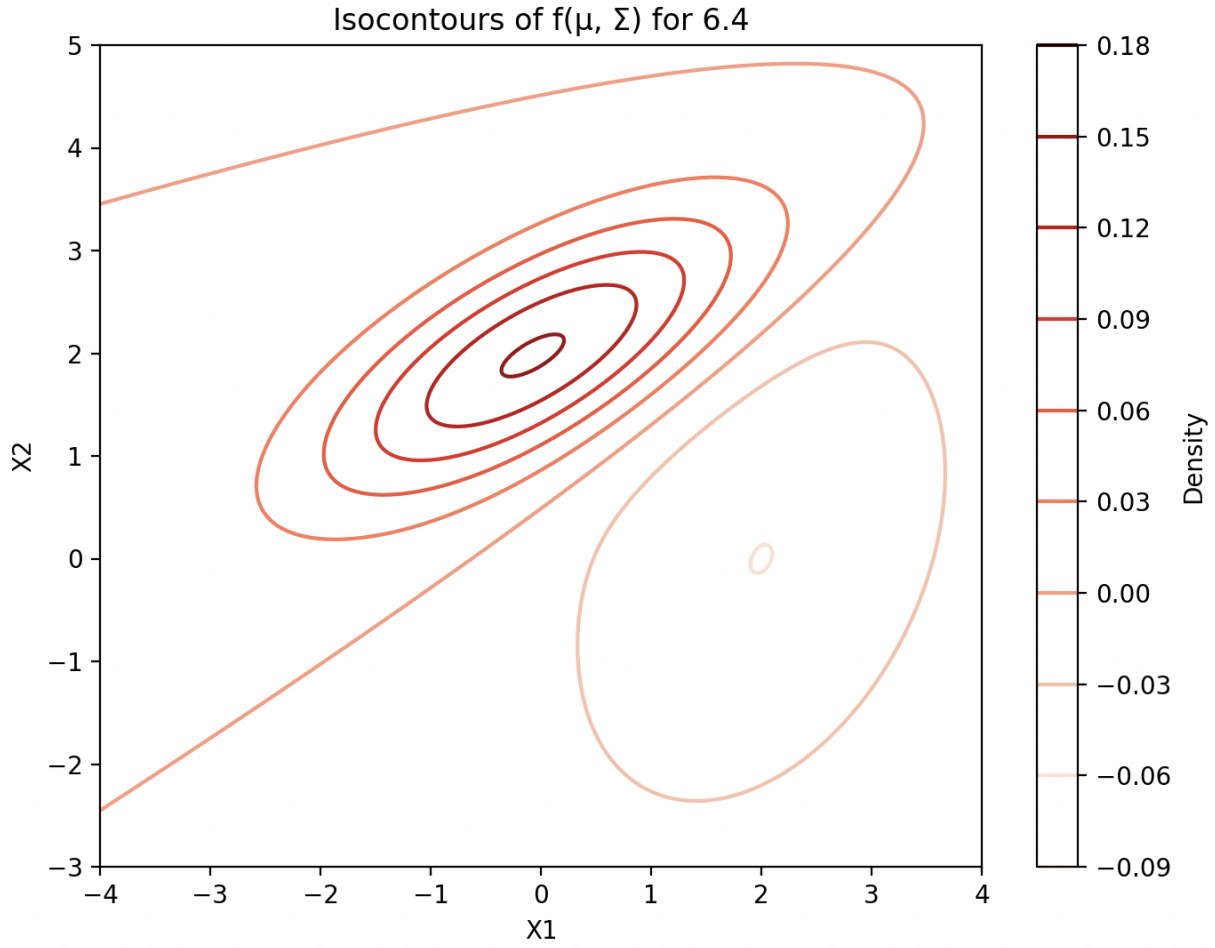
The code for this problem is located in the code appendix.

3.



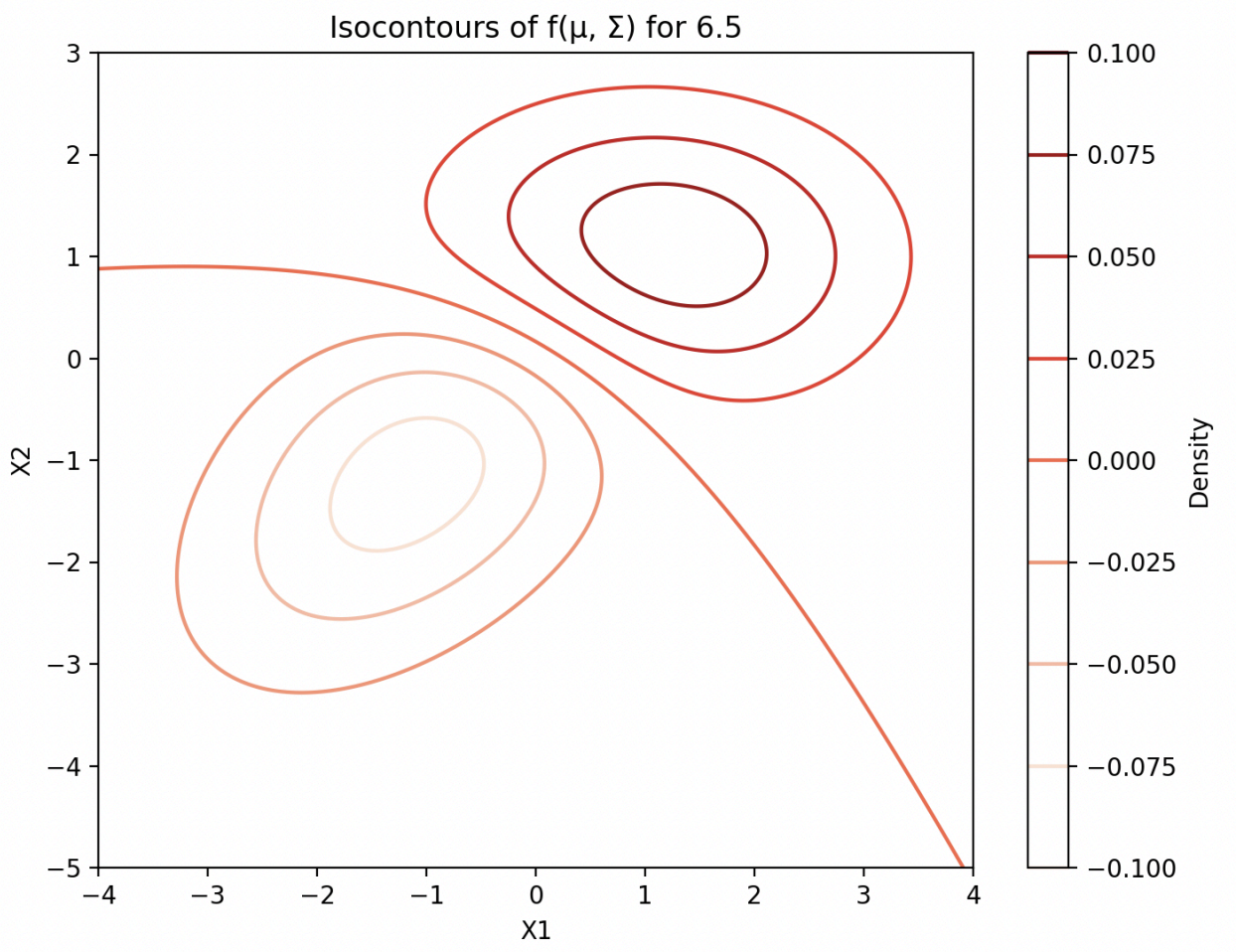
The code for this problem is located in the code appendix.

4.



The code for this problem is located in the code appendix.

5.



The code for this problem is located in the code appendix.

Problem 6 Code Appendix:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import multivariate_normal

##############
# Question 6.1
mu1 = np.array([1, 1])
sigma1 = np.array([[1, 0], [0, 2]])

# Grid of x, y points
x = np.linspace(-3, 4, 500)
y = np.linspace(-2, 4, 500)
X, Y = np.meshgrid(x, y)
```

```
pos = np.dstack((X, Y))

rv1 = multivariate_normal(mu1, sigma1)

# Plot
plt.figure(figsize=(8, 6))
plt.contour(x, y, rv1.pdf(pos), cmap='Reds')
plt.colorbar(label='Density')
plt.title('Isocontours of f(μ, Σ) for 6.1')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()

##############
# Question 6.2
mu2 = np.array([-1, 2])
sigma2 = np.array([[2, 1], [1, 4]])

# Grid of x, y points
x = np.linspace(-4, 4, 500)
y = np.linspace(-2, 6, 500)
X, Y = np.meshgrid(x, y)
pos = np.dstack((X, Y))

rv2 = multivariate_normal(mu2, sigma2)

# Plot
plt.figure(figsize=(8, 6))
plt.contour(x, y, rv2.pdf(pos), cmap='Reds')
plt.colorbar(label='Density')
plt.title('Isocontours of f(μ, Σ) for 6.2')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()

##############
# Question 6.3
mu1 = np.array([0, 2])
mu2 = np.array([2, 0])
sigma1 = np.array([[2, 1], [1, 1]])

# Grid of x, y points
```

```
x = np.linspace(-3, 5, 500)
y = np.linspace(-2, 4, 500)
X, Y = np.meshgrid(x, y)
pos = np.dstack((X, Y))

rv1 = multivariate_normal(mu1, sigma1)
rv2 = multivariate_normal(mu2, sigma1)
z = rv1.pdf(pos) - rv2.pdf(pos)

# Plot
plt.figure(figsize=(8, 6))
plt.contour(x, y, z, cmap='Reds')
plt.colorbar(label='Density')
plt.title('Isocontours of f(μ, Σ) for 6.3')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()

##############
# Question 6.4
mu1 = np.array([0, 2])
mu2 = np.array([2, 0])
sigma1 = np.array([[2, 1], [1, 1]])
sigma2 = np.array([[2, 1], [1, 4]])

# Grid of x, y points
x = np.linspace(-4, 4, 500)
y = np.linspace(-3, 5, 500)
X, Y = np.meshgrid(x, y)
pos = np.dstack((X, Y))

rv1 = multivariate_normal(mu1, sigma1)
rv2 = multivariate_normal(mu2, sigma2)
z = rv1.pdf(pos) - rv2.pdf(pos)

# Plot
plt.figure(figsize=(8, 6))
plt.contour(x, y, z, cmap='Reds')
plt.colorbar(label='Density')
plt.title('Isocontours of f(μ, Σ) for 6.4')
plt.xlabel('X1')
plt.ylabel('X2')
```

```
plt.show()

##############
# Question 6.5
mu1 = np.array([1, 1])
mu2 = np.array([-1, -1])
sigma1 = np.array([[2, 0], [0, 1]])
sigma2 = np.array([[2, 1], [1, 2]])

# Grid of x, y points
x = np.linspace(-4, 4, 500)
y = np.linspace(-5, 3, 500)
X, Y = np.meshgrid(x, y)
pos = np.dstack((X, Y))

rv1 = multivariate_normal(mu1, sigma1)
rv2 = multivariate_normal(mu2, sigma2)
z = rv1.pdf(pos) - rv2.pdf(pos)

# Plot
plt.figure(figsize=(8, 6))
plt.contour(x, y, z, cmap='Reds')
plt.colorbar(label='Density')
plt.title('Isocontours of f(μ, Σ) for 6.5')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```

Problem 7 Deliverables:

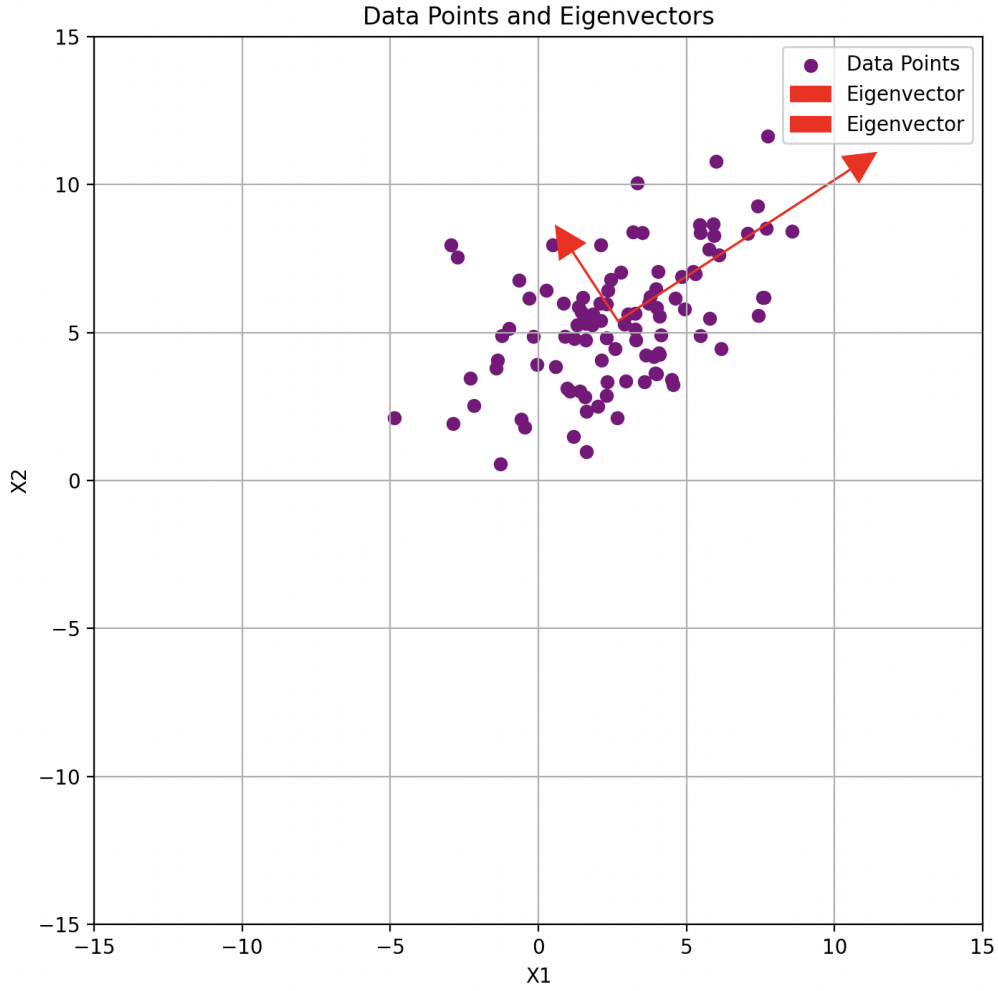
The random seed I chose for this problem is **np.random.seed(42)**.

1. Mean of the sample: [2.68846045 5.3888394]
The code for this problem is located in the code appendix.

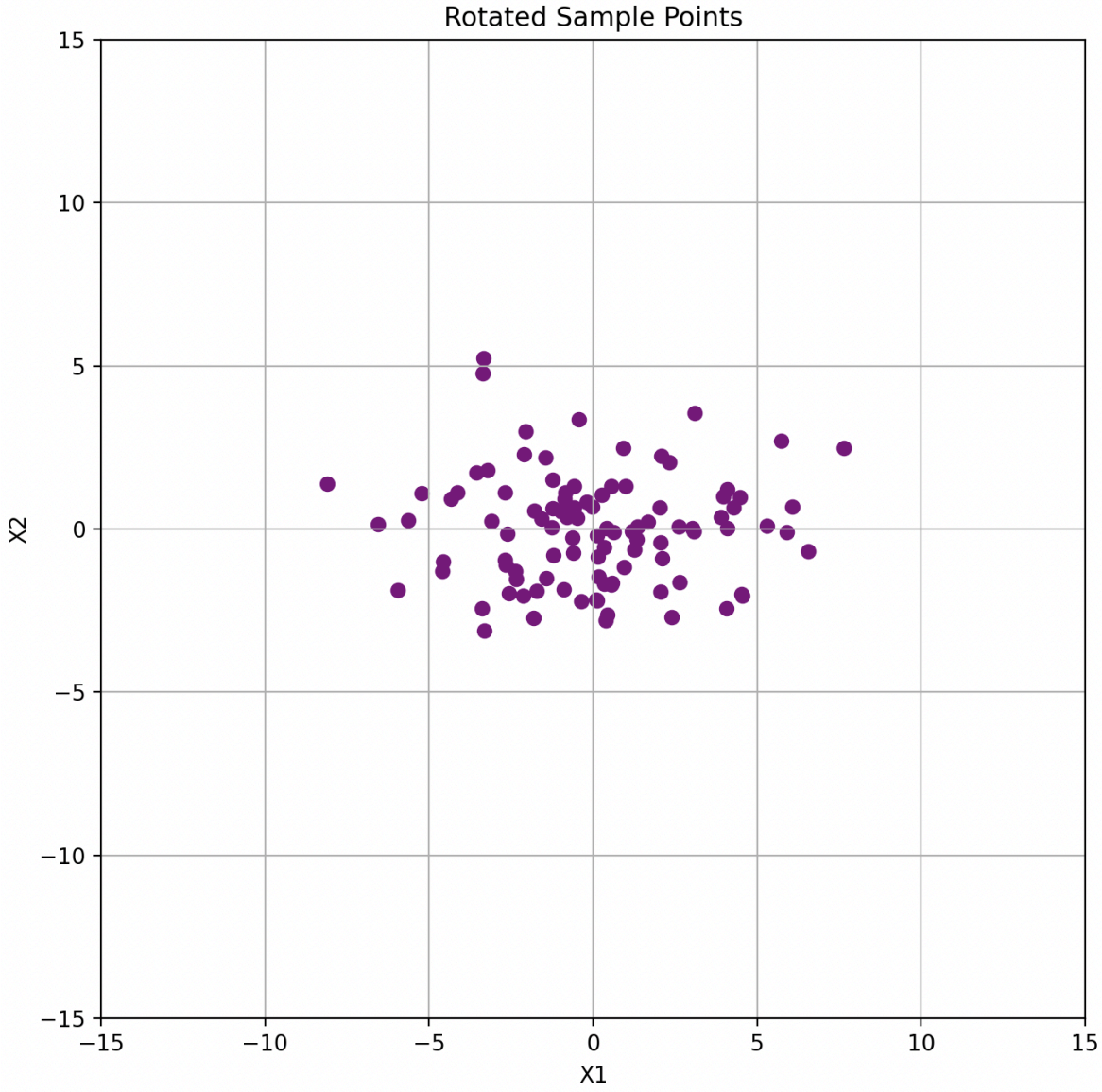
2. Covariance matrix of the sample:
[[7.42292904 3.00253936]
[3.00253936 4.78474509]]
The code for this problem is located in the code appendix.

3. Eigenvalues of the sample: [9.38335628 2.82431785]
Eigenvectors of the sample:
[[0.83732346 -0.54670781]
[0.54670781 0.83732346]]
The code for this problem is located in the code appendix.

4. The code for this problem is located in the code appendix.



5. The code for this problem is located in the code appendix.



Problem 7 Code Appendix:

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(42)
```

```
##############
# Question 7.1

X1 = np.random.normal(loc=3, scale=3, size=100) # X1 ~ N(3, 9)
X2 = 0.5*X1 + np.random.normal(loc=4, scale=2, size=100) # X2 ~ 0.5*X1 + N(4, 4)
sample_points = np.column_stack((X1, X2))

# Compute the mean
sample_mean = np.mean(sample_points, axis=0)
print("Mean of the sample:")
print(sample_mean)
# Results: Mean of the sample = [2.68846045, 5.3888394]

##############
# Question 7.2

sample_covariance_matrix = np.cov(sample_points, rowvar=False)
print("\nCovariance matrix of the sample:")
print(sample_covariance_matrix)
# Results: Covariance matrix of the sample = [[7.42292904 3.00253936]
#                                             [3.00253936 4.78474509]]

##############
# Question 7.3
eigenvalues, eigenvectors = np.linalg.eig(sample_covariance_matrix)
print("\nEigenvalues of the sample:")
print(eigenvalues)
print("\nEigenvectors of the sample:")
print(eigenvectors)
# Results: Eigenvalues of the sample = [9.38335628 2.82431785]
#          Eigenvectors of the sample = [[ 0.83732346 -0.54670781]
#                                        [ 0.54670781  0.83732346]]

##############
# Question 7.4

# Part (i): plot all n=100 data points
plt.figure(figsize=(8, 8))  # Set figure size to make it square
plt.scatter(sample_points[:, 0], sample_points[:, 1], color='purple', label='Data
Points')

# Part (ii): plot the eigenvectors as arrows
```

```
for i in range(len(eigenvalues)):
   eigenvector = eigenvectors[:, i] # access the ith column (or eigenvector) of the
eigenvectors array
   eigenvalue = eigenvalues[i]
   scaled_eigenvector = eigenvector * eigenvalue
   plt.arrow(sample_mean[0], sample_mean[1], scaled_eigenvector[0],
scaled_eigenvector[1],
             head_width=1, head_length=1, fc='red', ec='red', label=f'Eigenvector')

# Plot
plt.xlim(-15, 15)
plt.ylim(-15, 15)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Data Points and Eigenvectors')
plt.legend()
plt.grid(True)
plt.show()

##############
# Question 7.5

# Sort eigenvectors by eigenvalues (descending order)
sort_indices = np.argsort(eigenvalues)[::-1]
eigenvectors = eigenvectors[:, sort_indices]

rotation_matrix = eigenvectors  # rotation matrix U^T
centered_points = sample_points - sample_mean  # center the sample points
rotated_points = np.dot(centered_points, rotation_matrix)  # rotate each point by U^T

# Plot
plt.figure(figsize=(8, 8))
plt.scatter(rotated_points[:, 0], rotated_points[:, 1], color='purple', label='Rotated
Points')
plt.xlim(-15, 15)
plt.ylim(-15, 15)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Rotated Sample Points')
plt.grid(True)
plt.show()
```

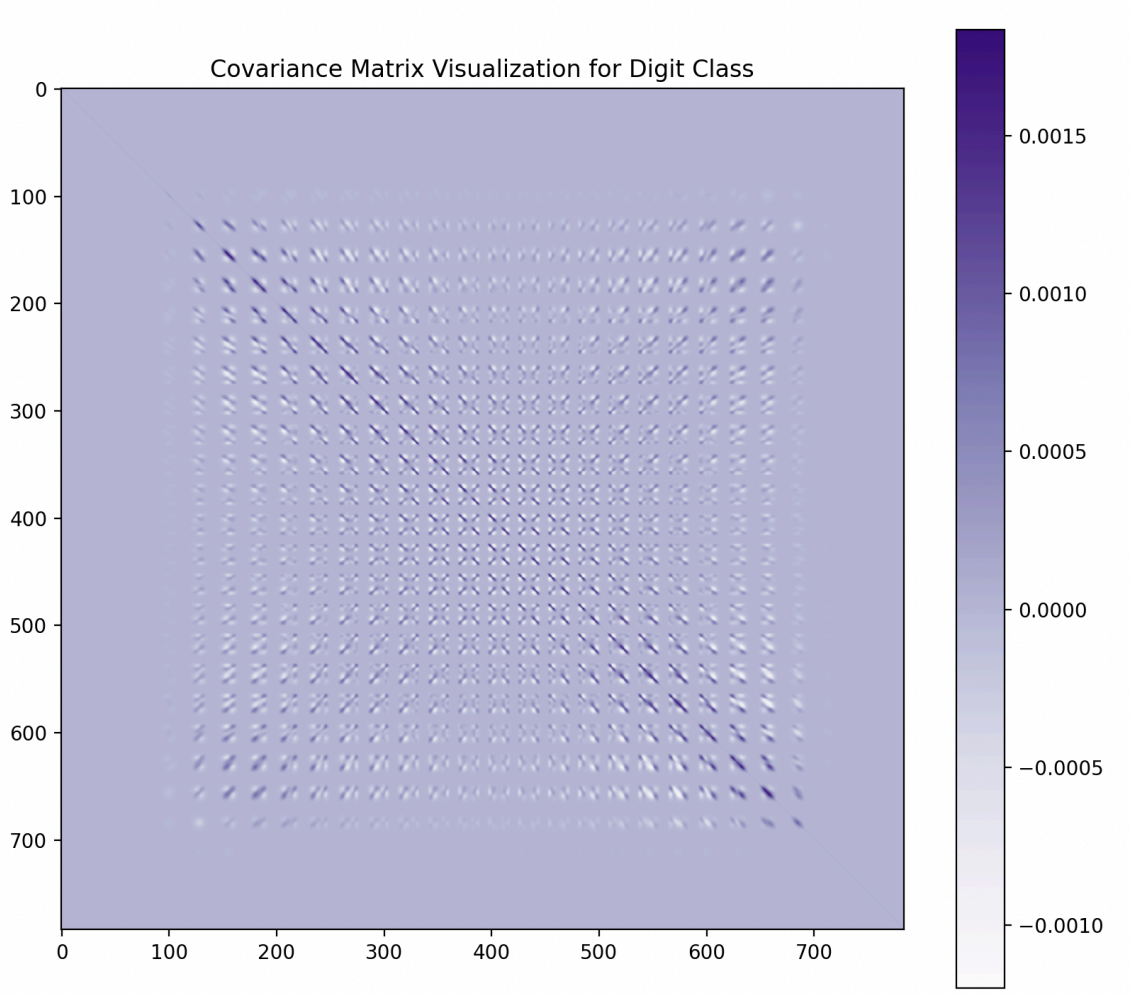
Problem 8 Deliverables:

1.

The code for this problem is located in the code appendix.

2.

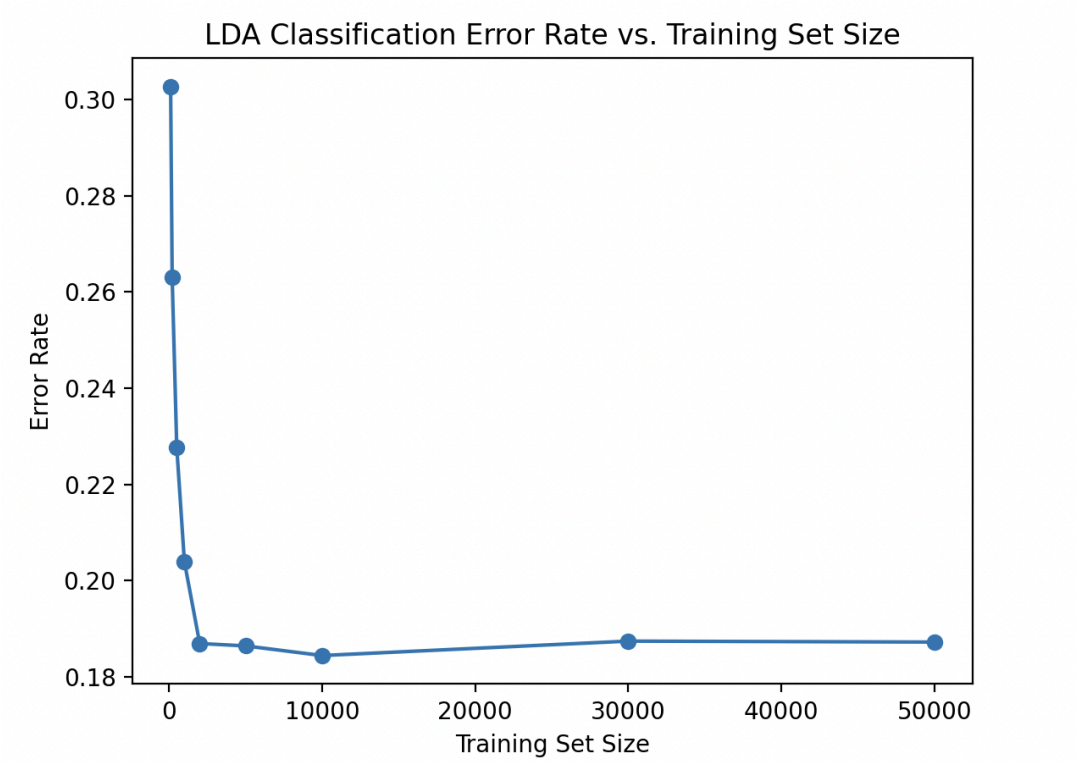
This is the covariance matrix for the digit class “0.” We can observe that the diagonal terms are larger than the non-diagonal terms. As we move further and further away from the diagonal, the entries become smaller and smaller. The diagonal terms of our covariance matrix represent the variance of each feature within the digit class, indicating how much each feature varies individually across the samples in the class. On the other hand, the off-diagonal terms represent the covariance between pairs of features within the digit class, indicating how much two features vary together across the samples in the class and capturing the relationship between different features. Knowing this, we can conclude that the correlation between features that are closer to each other is higher than the correlation between features that are further apart.



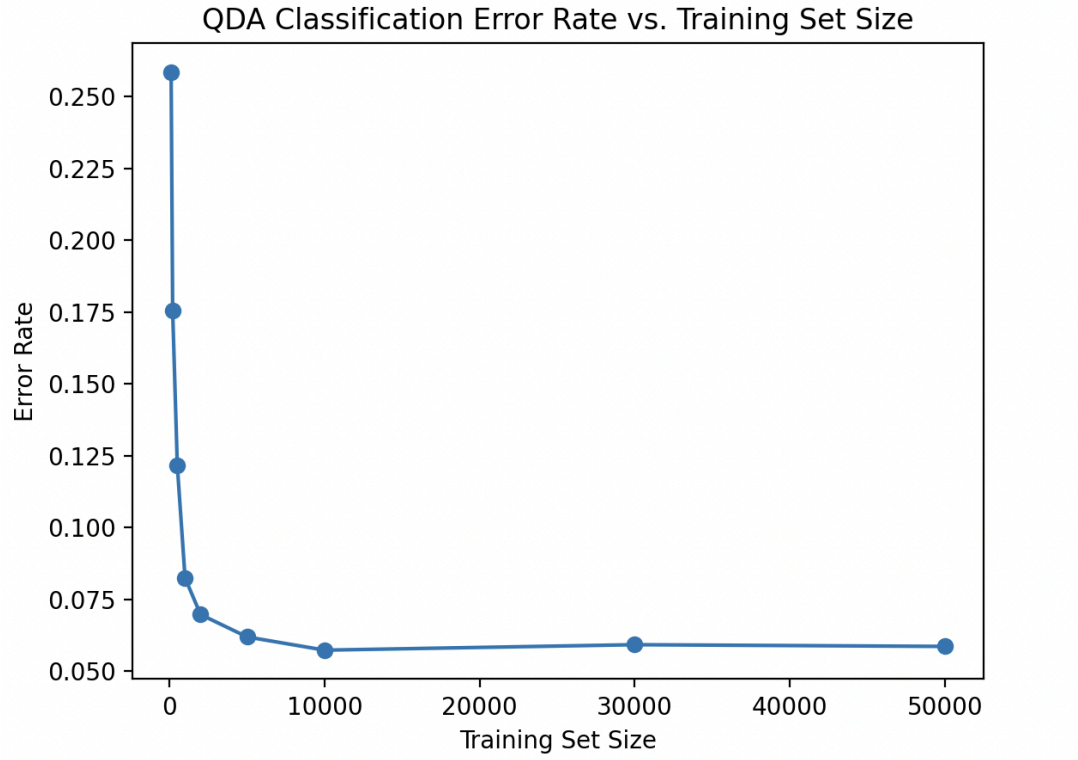
The code for this problem is located in the code appendix.

3.

a) The code for this problem is located in the code appendix.

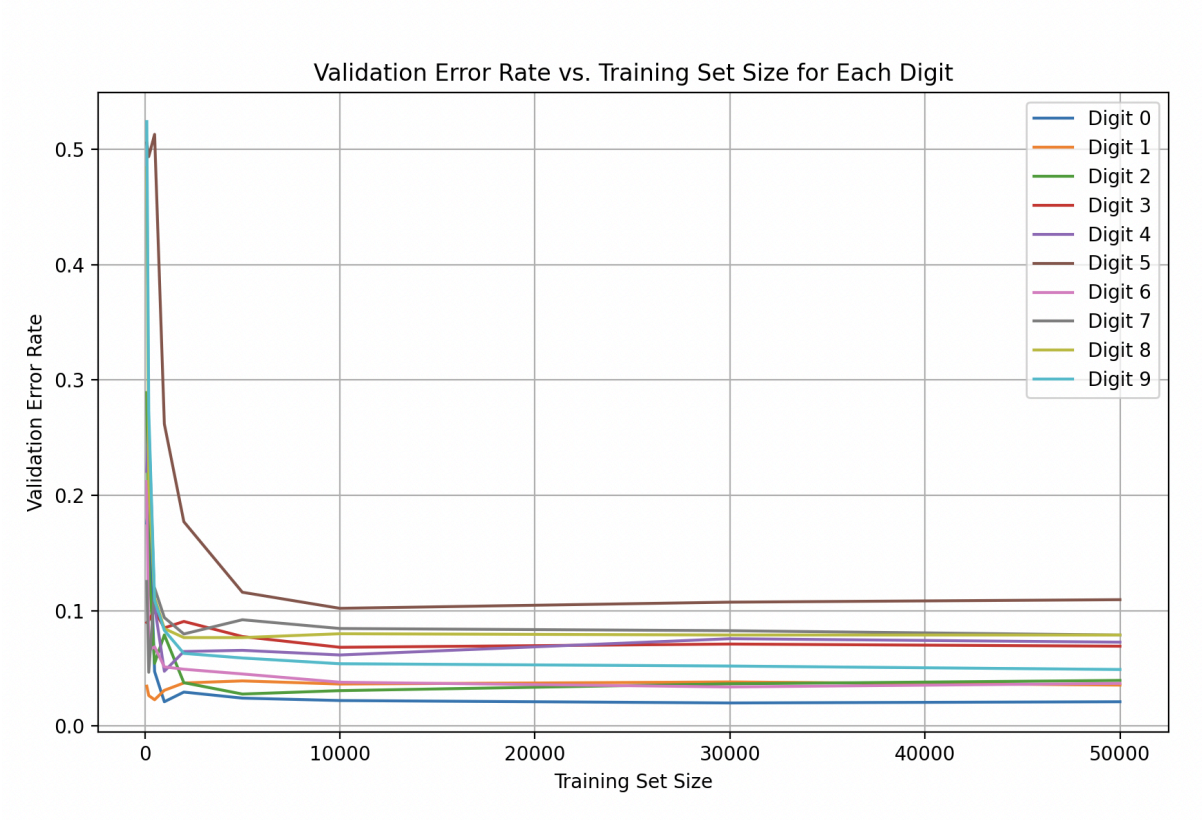


b) The code for this problem is located in the code appendix.



c) We can see that QDA performed better than LDA, as QDA converges to a lower error rate. This is because QDA offers more model flexibility by allowing each class to have its own covariance matrix, allowing it to capture more complex relationships between features within each class. QDA can also model non-linear decision boundaries, which may be better in cases where a given dataset has complex or non-linear relationships between features (in other words, LDA can lead to underfitting, which reduces the model's accuracy; however, QDA can lead to overfitting, which can also increase the error rate).

d) Pictured below is a plot of validation error versus the number of training points for each digit using QDA. The digit which is easiest to classify is digit 0. This is most likely the case because you can distinguish between digits like 0 and 1 by just examining a few pixels at the center of the image (typically white for digit 0 but black for digit 1) or at the left and right edges of the image (typically black for digit 0 but white for digit 1). Pairs of digits like 4 and 9, 3 and 8, and 1 and 7 look more similar and have more overlap, so they will be more challenging to classify.
External source used: http://varianceexplained.org/r/digit-eda/



The code for this problem is located in the code appendix.

4.

My Kaggle username: "Sajal Ravish"
MNIST dataset Kaggle score: 0.933

5.

My Kaggle username: “Sajal Ravish”
Spam dataset Kaggle score: 0.793

Problem 8 Code Appendix:

```
##############################################
# Question 8.1: Fitting a Gaussian distribution
import numpy as np
np.random.seed(42)

# data_name can be "mnist" or "spam"
def load_training_data(data_name):
    data = np.load(f"../data/{data_name}-data-hw3.npz")
    print("Loaded %s data!" % data_name)
    # fields = "test_data", "training_data", "training_labels"
    return data["training_data"], data["training_labels"]

# Function that normalizes images
def normalize_data(data):
    return np.array([(image / np.linalg.norm(image)).flatten() for image in data])

# Function that fits Gaussian distributions to each digit class
def fit_gaussian(data, labels, classes):
    fitted = {}
    for cls in classes:
        cls_data = data[(labels == cls).flatten()]
        mean = np.mean(cls_data, axis=0)
        covariance = np.cov(cls_data, rowvar=False)
        # If our covariance matrix is singular:
        if np.linalg.det(covariance) == 0:
            covariance += 1e-4 * np.eye(len(covariance))
        prior = len(cls_data) / len(data)
        fitted[cls] = (mean, covariance, prior)
    return fitted

mnist_data, mnist_labels = load_training_data("mnist")
mnist_classes = np.unique(mnist_labels)
mnist_data_normalized = normalize_data(mnist_data)
mnist_fitted = fit_gaussian(mnist_data_normalized, mnist_labels, mnist_classes)


##############################################
```

```
# Question 8.2: Visualizing the covariance matrix
import matplotlib.pyplot as plt

class_0_mean, class_0_covariance, class_0_prior = mnist_fitted[0]
plt.figure(figsize=(10, 10))
plt.imshow(class_0_covariance, cmap='Purples')
plt.title('Covariance Matrix Visualization for Digit Class')
plt.colorbar()
plt.show()


######################################################
# Question 8.3, part (a): Linear discriminant analysis
from scipy.stats import multivariate_normal

def partition(data, labels, split_ratio):
   # The split_ratio is what percent/number of training points we want to set aside
for the validation set

   # reshape and stack data and labels
   labels = labels.reshape((-1, 1))
   if data.ndim > 2:
       data = data.reshape((data.shape[0], -1))
   stacked_data = np.hstack((data, labels))

   # determine split_size
   if split_ratio <= 1:
       split_size = int(len(stacked_data) * split_ratio)
   else:
       split_size = split_ratio

   # randomly shuffle the data
   np.random.shuffle(stacked_data)

   # partition data
   training_data = stacked_data[split_size:]
   validation_data = stacked_data[:split_size]

   # separate features and labels for training and validation sets
   training_features = training_data[:, :-1]
   training_labels = training_data[:, -1]
```

```
    validation_features = validation_data[:, :-1]
    validation_labels = validation_data[:, -1]

    return training_features, training_labels, validation_features, validation_labels

# Compute means and pooled covariance for LDA
def fit_lda(data, labels, classes):
    means = {}
    pooled_covariance = 0
    n = len(data)
    for cls in classes:
        cls_data = data[(labels == cls).flatten()]
        mean = np.mean(cls_data, axis=0)
        means[cls] = (mean)
        pooled_covariance += np.sum(np.linalg.norm(cls_data - mean, axis=1)**2)
    pooled_covariance /= n
    return means, pooled_covariance

def predict_lda(data, means, pooled_covariance):
    # Calculate the log likelihood of each class and return the class with the maximum
likelihood
    log_likelihoods = []
    for cls in means:
        log_likelihood = multivariate_normal.logpdf(data, mean=means[cls],
cov=pooled_covariance, allow_singular=True)
        log_likelihoods.append(log_likelihood)

    log_likelihoods = np.array(log_likelihoods)
    predictions = np.argmax(log_likelihoods, axis=0)
    return predictions

mnist_training_features, mnist_training_labels, mnist_validation_features,
mnist_validation_labels = partition(mnist_data, mnist_labels, 10000)
training_sizes = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
error_rates = []

for size in training_sizes:
    # Sample the training data
    mnist_training_features_samples = normalize_data(mnist_training_features)[:size]
    mnist_training_labels_samples = mnist_training_labels[:size]
    mnist_validation_features_normalized = normalize_data(mnist_validation_features)
```

```
    means, pooled_covariance = fit_lda(mnist_training_features_samples,
mnist_training_labels_samples, mnist_classes)

    predictions = predict_lda(mnist_validation_features_normalized, means,
pooled_covariance)

    # Classify and compute error rate
    correct_predictions = np.sum(predictions == mnist_validation_labels)
    error_rate = 1.0 - (correct_predictions / len(mnist_validation_labels))
    error_rates.append(error_rate)

# Plot the Error Rates
plt.plot(training_sizes, error_rates, marker='o')
plt.xlabel('Training Set Size')
plt.ylabel('Error Rate')
plt.title('LDA Classification Error Rate vs. Training Set Size')
plt.show()

############################################################
# Question 8.3, part (b): Quadratic discriminant analysis

# Reset error_rates list before computing error rates for QDA
error_rates = []
error_rates_per_digit = {digit: [] for digit in mnist_classes}

def predict_qda(data, fitted):
    # Calculate the log likelihood of each class and return the class with the maximum
likelihood
    log_likelihoods = []
    for i in range(len(fitted)):
        mean_val, cov_matrix, prior = fitted[i]
        log_likelihood = multivariate_normal.logpdf(data, mean=mean_val,
cov=cov_matrix, allow_singular=True) + np.log(prior)
        log_likelihoods.append(log_likelihood)

    log_likelihoods = np.array(log_likelihoods)
    predictions = np.argmax(log_likelihoods, axis=0)
    return predictions

for size in training_sizes:
    # Sample the training data
    mnist_training_features_samples = normalize_data(mnist_training_features)[:size]
```

```
    mnist_training_labels_samples = mnist_training_labels[:size]
    mnist_validation_features_normalized = normalize_data(mnist_validation_features)

    # Compute means and covariances
    mnist_fitted = fit_gaussian(mnist_training_features_samples,
mnist_training_labels_samples, mnist_classes)
    predictions = predict_qda(mnist_validation_features_normalized, mnist_fitted)

    # Classify and compute error rate
    correct_predictions = np.sum(predictions == mnist_validation_labels)
    error_rate = 1.0 - (correct_predictions / len(mnist_validation_labels))
    error_rates.append(error_rate)

    # Update error rates for each digit for LDA
    for digit in mnist_classes:
        digit_indices = np.where(mnist_validation_labels == digit)[0]
        digit_error = 1.0 - (np.sum(predictions[digit_indices] == digit) /
len(digit_indices))
        error_rates_per_digit[digit].append(digit_error)

# Plot the Error Rates
plt.plot(training_sizes, error_rates, marker='o')
plt.xlabel('Training Set Size')
plt.ylabel('Error Rate')
plt.title('QDA Classification Error Rate vs. Training Set Size')
plt.show()

################################################################################
#############
# Question 8.3, part (c): Plot validation error versus the number of training points
for each digit

# Plot
plt.figure(figsize=(10, 6))
for digit, rates in error_rates_per_digit.items():
    plt.plot(training_sizes, rates, label=f'Digit {digit}')
plt.xlabel('Training Set Size')
plt.ylabel('Validation Error Rate')
plt.title('Validation Error Rate vs. Training Set Size for Each Digit')
plt.legend()
plt.grid(True)
plt.show()
```

```
#####################################
# Question 8.4: MNIST Kaggle submission

import pandas as pd

# A code snippet to help you save your results into a kaggle accepted csv
# Usage: results_to_csv(clf.predict(X_test))
def results_to_csv(y_test, file_name):
   y_test = y_test.astype(int)
   df = pd.DataFrame({'Category': y_test})
   df.index += 1 # Ensures that the index starts at 1
   df.to_csv(file_name, index_label='Id')

# save predictions to csv file
def load_test_data(data_name):
   data = np.load(f"../data/{data_name}-data-hw3.npz")
   print("Loaded %s data!" % data_name)
   # fields = "test_data", "training_data", "training_labels"
   return data["test_data"]

# Sample the training data
mnist_training_features_samples = normalize_data(mnist_training_features)[:15000]
mnist_training_labels_samples = mnist_training_labels[:15000]

mnist_test_data = load_test_data("mnist")
mnist_test_data_normalized = normalize_data(mnist_test_data)
# Compute means and covariances
mnist_fitted = fit_gaussian(mnist_training_features_samples,
mnist_training_labels_samples, mnist_classes)

# Classify
mnist_test_predictions = predict_qda(mnist_test_data_normalized, mnist_fitted)

results_to_csv(mnist_test_predictions, "MNIST-submission.csv")
print("MNIST data saved to submission.csv")

#####################################
# Question 8.5: Spam Kaggle submission

# Sample the training data
spam_data, spam_labels = load_training_data("spam")
```

```
spam_classes = np.unique(spam_labels)
spam_fitted = fit_gaussian(spam_data, spam_labels, spam_classes)

# Classify
spam_test_data = load_test_data("spam")
spam_test_predictions = predict_qda(spam_test_data, spam_fitted)

results_to_csv(spam_test_predictions, "spam-submission.csv")
print("spam data saved to submission.csv")
```