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:

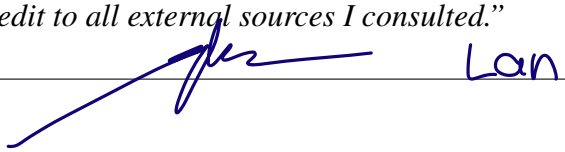
1. Submit your predictions for the test sets to Kaggle as early as possible. Include your Kaggle scores in your write-up. The Kaggle competition for this assignment can be found at
   - MNIST: https://www.kaggle.com/competitions/cs189-hw3-mnist-spring-2024/
   - SPAM: https://www.kaggle.com/competitions/cs189-hw3-spam-spring-2024/
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: 

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 $\mu_1 < b < \mu_2$.*

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.

Q2.1

• Posterior probability :

$$P(Y = C_i \mid X = x) = \frac{f_{X|Y=C_i}(x) \cdot P(Y = C_i)}{\sum_{j=1}^{2} f_{X|Y=C_j}(x) \, P(Y = C_j)}$$

To find the Bayes optimal decision boundary, find x such that the posterior probabilities are equal.

$\{x : P(Y = C_1 \mid X = x) = 0.5\}$

Because $P(Y = C_1) = P(Y = C_2) = 0.5$

We want $f_{X|Y=C_1}(x) = f_{X|Y=C_2}(x_2)$

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

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

Take log:

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

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

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

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

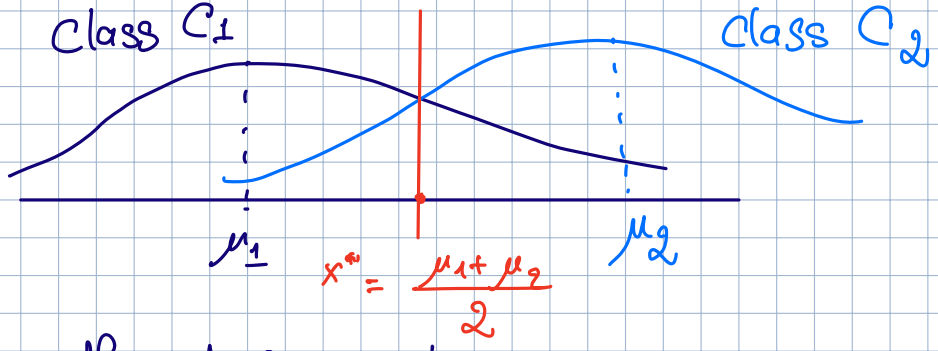
$$\Leftrightarrow \quad -2x(\mu_1 - \mu_2) = -\mu_1^2 + \mu_2^2$$

$$\Leftrightarrow \quad x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)}$$

$$\Rightarrow \quad \boxed{x = \frac{\mu_1 + \mu_2}{2}}$$

Q2.2

Because $\mu_1 < \mu_2$, we can visualize two distributions for two classes as follows:



Then, the decision rule is:

$$r(x) = \begin{cases} 1 & \text{if } x < \frac{\mu_1 + \mu_2}{2} \\ 2 & \text{otherwise} \end{cases}$$

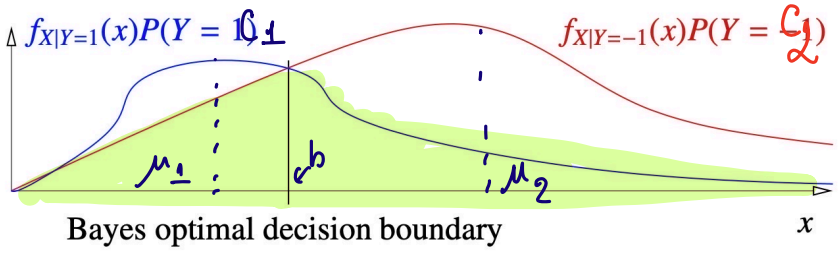
Q2 - 2

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

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



Bayes optimal decision boundary

Because the optimal decision boundary predict class with higher posterior probability, the error would be the area of the lower posterior probability when it misclassifies.

Because $\mu_2 > \mu_1$, the left side of b would be when it misclassifies C_2 as C_1

$$P_1 = P(C_2 \text{ misclassified as } C_1 | C_2) \cdot P(C_2) = f_{X|Y=C_2}(x) \cdot P(Y = C_2)$$

$$P_1 = \left(\frac{1}{2}\right) \cdot \int_{-\infty}^{b} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{\frac{-(x-\mu_2)^2}{2\sigma^2}\right\} dx$$

Similarly,

$$P_2 P(C_1 \text{ misclassified as } C_2 | C_1) \cdot P(C_1) = f_{X|Y=C_1}(x) \cdot P(Y = C_1)$$

$$P_2 = \left(\frac{1}{2}\right) \cdot \int_{b}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{\frac{-(x-\mu_1)^2}{2\sigma^2}\right\} dx$$

Hence,

$$P_e = P_1 + P_2$$

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

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* $\mu_1 < b < \mu_2$.

Q2.3

Because $P_e(b)$ is convex for $\mu_1 < b < \mu_2$, we can find b^* by taking the derivative and set it to 0.

By Fundamental Theorem of Calculus,

$$F(x) = \int_a^x f(t)\,dt$$

$$F'(x) = f(x)$$

$$\frac{dP_e(b)}{db} = \frac{1}{2\sqrt{2\pi}\sigma}\left[\exp\left\{\frac{-(b-\mu_2)^2}{2\sigma^2}\right\} - \exp\left\{-\frac{(b-\mu_1)^2}{2\sigma^2}\right\}\right] = 0$$

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

Taking log:

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

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

$$\Leftrightarrow \quad b^2 - 2b\mu_2 + \mu_2^2 = b^2 - 2b\mu_1 + \mu_1^2$$

$$\Leftrightarrow \quad -2b(\mu_2 - \mu_1) = -\mu_2^2 + \mu_1^2$$

$$\Leftrightarrow \quad b^* = \frac{\mu_2^2 - \mu_1^2}{2(\mu_2 - \mu_1)} = \frac{\mu_2 + \mu_1}{2}$$

This b^* is the same as the one derived 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.

Q3.1

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.

- Risk when we choose correct class (choose class i)

$$R(r(x)=i|x) = \lambda_s \sum_{j \neq i} P(Y=j|x) = \lambda_s \left(1 - P(Y=i|x)\right)$$

- Risk when we choose doubt ($i = c+1$):

$$R(r(x)=c+1|x) = \lambda_r \sum_{j=1}^{c} P(Y=j|x) = \lambda_r \cdot (1) = \lambda_r$$

For all j, if we choose class j, our risk would be:

$$R(r(x)=j|x) = \lambda_s \sum_{i \neq j} P(Y=i|x)$$

when $P(Y=i|x) \geq P(Y=j|x)$

$$R(r(x)=j|x) \geq R(r(x)=i|x)$$

Then, we would choose class i for all j to have smaller risk.

To favor class i over doubt, the risk choosing i should be lesser than choosing doubt:

$$R(r(x)=i|x) \leq \lambda_r$$

$$\lambda_s(1 - P(Y=i|x)) \leq \lambda_r$$

$$\lambda_s - \lambda_s P(Y=i|x) \leq \lambda_r$$

$$\lambda_s P(Y=i|x) \geq \lambda_s - \lambda_r$$

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

Q3.1

Lastly, when none of the two satisfied, we would favor doubt over its correct label because $\lambda_r \leq \lambda_s$ (Their risk would be λ_r or λ_s multiply by its probability of correct class)

In conclusion, we just show that

if $\lambda_r \leq \lambda_s$:

- Choose class i if $P(Y=i|x) \geq P(Y=j|x)$ and $P(Y=i|x) \geq 1 - \frac{\lambda_r}{\lambda_s}$
- 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.

Q32

- If $\lambda_r = 0$: Risk when choosing doubt becomes:

$$R(r(x) = C+1 \mid x) = \lambda_r = 0$$

This is the lowest risk over other choices. Thus, the decision would be to choose doubt all the time.

- If $\lambda_r > \lambda_s$, the condition for

$$P(Y = i \mid X) \geq 1 - \frac{\lambda_r}{\lambda_s}$$ would always be true because $\frac{\lambda_r}{\lambda_s} > 1 \Rightarrow 1 - \frac{\lambda_r}{\lambda_s} < 0$

Then the system would:

- Choose class i if $P(Y=i \mid X) \geq P(Y=j \mid X)$ (for the same reason as part 1)
- Choose class j otherwise

Intuitively,

- if $\lambda_r = 0$: There is no penalty for doubt, encouraging the system always choose doubt over classifying
- if $\lambda_r > \lambda_s$: The system is encouraged to make classification over choosing doubt because of higher penalty of choosing doubt.

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 $\hat{\mu}$?

4 Maximum Likelihood Estimation and Bias

Q4.1

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.

Likelihood function:

$\sigma_i = \sigma/i$

$\sigma_i^2 = \sigma^2/i^2$

$$\text{Lik}(\mu,\sigma) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left\{-\frac{(X_i-\mu)^2}{2\sigma_i^2}\right\}$$

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

$$L(\mu,\sigma) = \log(\text{Lik}(\mu,\sigma)) = \sum_{i=1}^{n} \log\left(\frac{\sqrt{i}}{\sqrt{2\pi}\sigma}\right) - \sum_{i=1}^{n} \frac{i(X_i-\mu)^2}{2\sigma^2}$$

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

- $$\frac{\partial L(\mu,\sigma)}{\partial \mu} = \frac{2}{2\sigma^2}\sum_{i=1}^{n} i(X_i-\mu) = \sum_{i=1}^{n} \frac{i(X_i-\mu)}{\sigma^2} = 0$$

$$\Leftrightarrow \sum_{i=1}^{n} iX_i - \mu\sum_{i=1}^{n} i = 0$$

$$\hat{\mu} = \frac{\sum_{i=1}^{n} iX_i}{\sum_{i=1}^{n} i} = 2\frac{\sum_{i=1}^{n} iX_i}{n(n+1)}$$

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

$$\Leftrightarrow -n\sigma^2 + \sum_{i=1}^{n} (X_i-\mu)^2 = 0$$

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

Q4.2

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.*

$$\cdot E[\hat{\mu}] = E\left[\frac{2\sum_{i=1}^{n} i X_i}{n(n+1)}\right] = \frac{2}{n(n+1)} E\left[\sum_{i=1}^{n} i X_i\right]$$

$$= \frac{2}{n(n+1)} \sum_{i=1}^{n} i E[X_i] = \frac{2}{n(n+1)} \sum_{i=1}^{n} i \mu$$

$$= \frac{2}{n(n+1)} \frac{n(n+1)}{2} \cdot \mu = \mu$$

$$\text{bias}(\hat{\mu}) = E[\hat{\mu}] - \mu = \mu - \mu = 0$$

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

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.*

(243)

First, we observe:

$$\hat{\mu} = \frac{2}{n(n+1)} \sum_{i=1}^{n} i X_i$$

$$\Leftrightarrow \sum_{i=1}^{n} i X_i = \frac{n(n+1)}{2} \hat{\mu}$$

$$\text{Var}(\hat{\mu}) = E[\hat{\mu}^2] - E[\hat{\mu}]^2$$

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

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

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

$$= \left[\frac{2}{n(n+1)}\right]^2 \text{Var}\left(\sum_{i=1}^{n} i X_i\right)$$

$$= \left[\frac{2}{n(n+1)}\right]^2 \sum_{i=1}^{n} i^2 \text{Var}(X_i) \quad \text{(b/c } X_i\text{'s are independent)}$$

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

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

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

- Compute $E[\hat{\sigma}^2]$

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

$$= \frac{1}{n} E\left[\sum_{i=1}^{n} i X_i^2 - 2\hat{\mu} \sum_{i=1}^{n} i X_i + \hat{\mu}^2 \sum_{i=1}^{n} i\right]$$

$$= \frac{1}{n} E\left[\sum_{i=1}^{n} i X_i^2 - 2 \cdot \frac{n(n+1)}{2} \hat{\mu}^2 + \frac{n(n+1)}{2} \hat{\mu}^2\right]$$

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

$$= \frac{1}{n} \sum_{i=1}^{n} i E[X_i^2] - \frac{1}{n} \cdot \frac{n(n+1)}{2} E[\hat{\mu}^2]$$

$$= \frac{1}{n} \sum_{i=1}^{n} i \left(\text{Var}[X_i] + E[X_i]^2\right) - \frac{n+1}{2} \left(\frac{2}{n(n+1)} \sigma^2 + \mu^2\right)$$

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

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

$$\text{Bias}[\hat{\sigma}^2] = E[\hat{\sigma}^2] - \sigma^2 = \sigma^2 - \frac{1}{n} \sigma^2 - \sigma^2 = \frac{-1}{\sigma^2} \neq 0$$

Thus, $\hat{\sigma}^2$ is not unbiased.

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 $\hat{\mu}$?

Q4.4

$$R(\hat{\mu}) = E[L(\hat{\mu}, \mu)] = E[(\hat{\mu} - \mu)^2] = \text{Var}(\hat{\mu})$$

when σ is know, MLE $\hat{\mu}$ becomes:

$$\text{Lik}(\mu) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{\frac{-(X_i - \mu)^2}{2\sigma^2}\right\}$$

$$L(\mu) = \log(\text{Lik}(\mu)) = \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \sum_{i=1}^{n} \frac{(X_i - \mu)^2}{2\sigma^2}$$

$$\frac{\partial L(\mu)}{\partial \mu} = 2 \sum_{i=1}^{n} \frac{(X_i - \mu)}{2\sigma_i} = 0$$

$$\sum_{i=1}^{n} X_i - \sum_{i=1}^{n} \mu = 0$$

$$n\hat{\mu} = \sum_{i=1}^{n} X_i$$

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

$$R(\hat{\mu}) = \text{Var}(\hat{\mu}) = \text{Var}\left(\frac{1}{n} \sum_{i=1}^{n} X_i\right) = \frac{1}{n^2} \text{Var}\left(\sum_{i=1}^{n} X_i\right)$$

(b/c $X_1, X_2, \ldots X_n$ are independent) $= \frac{1}{n^2} \sum_{i=1}^{n} \text{Var}(X_i)$

$$= \frac{1}{n^2} \sum_{i=1}^{n} \sigma^2$$

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

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

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

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

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 $y^\top X$?

5 Covariance Matrices and Decompositions

Q5.1

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

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

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

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 $\mathrm{Var}(R) = \Sigma$.

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

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

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.

Geometrically, $\hat{\Sigma}$ is not invertible under circumstances where the sample points do not fully span the space due to:

- Lying on lower-dimension (line, plane, hyperplane)
- The number of samples being less than dimensionality of the space
- Perfect multicolinearity among two or more dimensions, showing redundancy in the data.

Q5.2

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.

Suggest: Regularization

$$\hat{\Sigma}_{new} = \hat{\Sigma} + \lambda I \quad ; \lambda > 0$$

How to choose λ:

1. Fixed small values: such as 10^{-4} or 10^{-6}
2. Cross-validation: optimize for model performance on a validation set
3. Eigenvalue Analysis: Set λ to a value slightly larger than the abosolute value of the smallest eigenvalue

Q5.3

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.

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

To maximize $f(x)$, we want to maximize $\exp\left\{ \frac{-1}{2} x^T \Sigma^{-1} x \right\}$, which is equivalent to minimize $x^T \Sigma^{-1} x$ (because of $\frac{-1}{2}$ term)
This can be achieved when x corresponds to the direction of eigenvector of Σ that has largest eigenvalue.

Similarly, to minimize $f(x)$, we would pick x corresponds to the direction of eigenvector of Σ that has smallest eigenvalue

Q5.4

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 $y^\top X$?

- $\mathrm{Var}(p) = \mathrm{Var}(y^T X) = y^T \mathrm{Var}(X)\, y = y^T \Sigma y$

We can write

$\Sigma = Q \Lambda Q^T$ where $Q = \begin{bmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{bmatrix}$; v_i : eigenvector

Then:

$\mathrm{Var}(p) = y^T Q \Lambda Q^T y$ $\quad \Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix}$; λ_i : eigenvalue

If we choose y to be aligned with eigenvector corresponding to λ_{max}, the $\mathrm{Var}(p)$ is maximized.

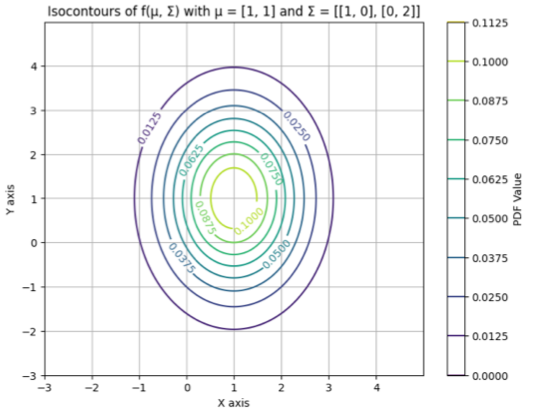
$\Rightarrow$ The λ_{max} of Σ represents the maximum variance that can be obtained by projecting X onto any unit vector.

6 Isocontours of Normal Distributions

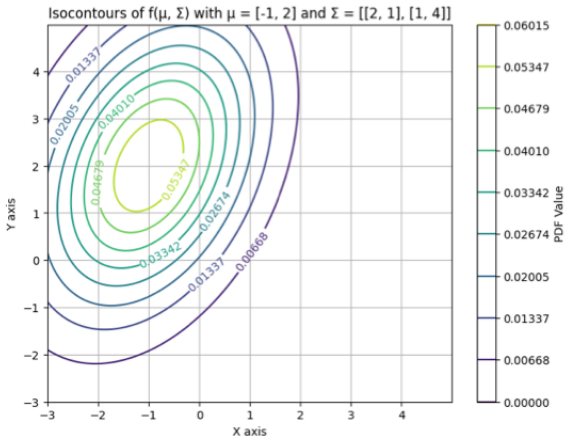
Let $f(\mu, \Sigma)$ be the probability density function of a normally distributed random variable in $\mathbb{R}^2$. Write code to plot the isocontours of the following functions, each on its own separate figure. Make sure it is clear which figure belongs to which part. You're free to use any plotting 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 $\Sigma_1 = \Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$.

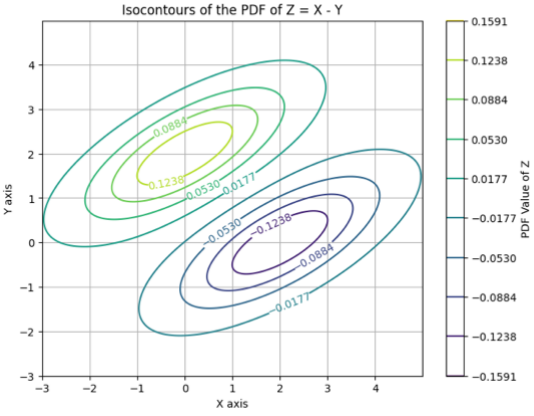
   (Yes, this is a difference between two PDFs. No, it is not itself a valid PDF. Just plot its isocontours anyway.)
4. $f(\mu_1, \Sigma_1) - f(\mu_2, \Sigma_2)$, where $\mu_1 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 2 & 1 \\ 1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


Isocontours of f(μ, Σ) with μ = [1, 1] and Σ = [[1, 0], [0, 2]]
X axis
Y axis
PDF Value

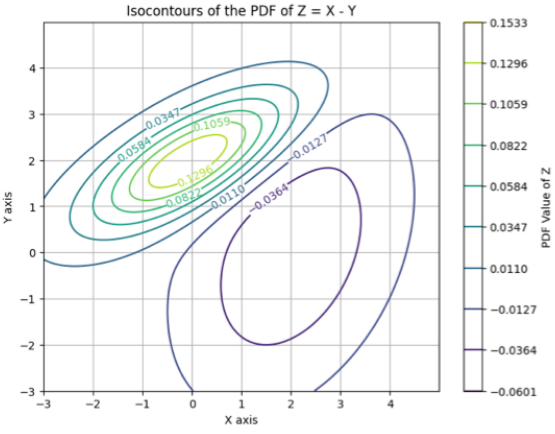
1


Isocontours of f(μ, Σ) with μ = [-1, 2] and Σ = [[2, 1], [1, 4]]
X axis
Y axis
PDF Value

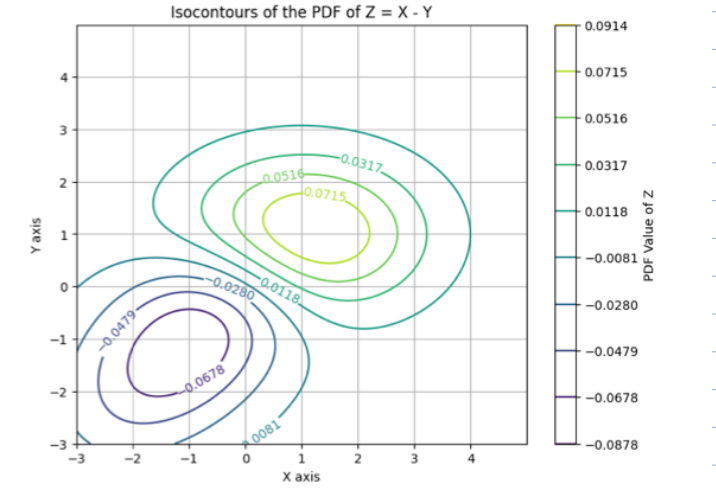
2


Isocontours of the PDF of Z = X - Y
X axis
Y axis
PDF Value of Z

3


Isocontours of the PDF of Z = X - Y
X axis
Y axis
PDF Value of Z

4


Isocontours of the PDF of Z = X - Y
X axis
Y axis
PDF Value of Z

5

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.

1. Compute the mean (in R 2) of the sample

```
np.random.seed(42)
X_1 = np.random.normal(loc=3, scale=3, size=100)
Y = np.random.normal(loc=4, scale=2, size=100)
X_2 = (1/2) * X_1 +Y
sample = np.array([(x_1, x_2) for (x_1, x_2) in zip(X_1, X_2)])


sample_mean = np.mean(sample, axis = 0)
sample_mean
```

```
array([2.68846045, 5.3888394 ])
```

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).

```
cov_hat = np.cov(sample.T)
cov_hat
```

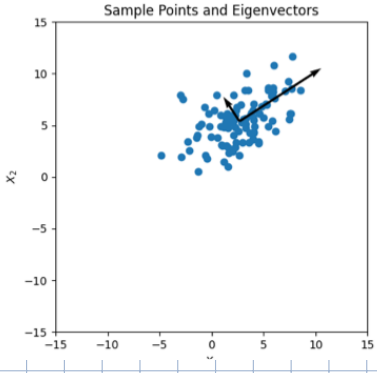
```
array([[7.42292904, 3.00253936],
       [3.00253936, 4.78474509]])
```

3. Compute the eigenvectors and eigenvalues of this covariance matrix

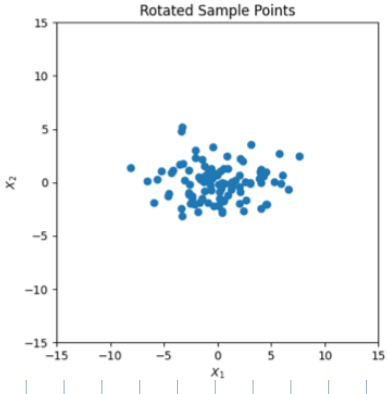
```
eigen_vals, eigen_vecs = np.linalg.eig(cov_hat)
eigen_vals, eigen_vecs
```

```
(array([9.38335628, 2.82431785]),
 array([[ 0.83732346, -0.54670781],
        [ 0.54670781,  0.83732346]]))
```

```
plt.figure(figsize=(5,5))
plt.xlim(-15,15)
plt.ylim(-15,15)
plt.xlabel("$X_1$")
plt.ylabel("$X_2$")
plt.title("Sample Points and Eigenvectors")
plt.scatter(sample[:, 0], sample[:, 1])
vec_X = [sample_mean[0], sample_mean[0]]
vec_Y = [sample_mean[1], sample_mean[1]]
vec_U = [eigen_vecs[0][0] * eigen_vals[0], eigen_vecs[0][1] * eigen_vals[1]]
vec_V = [eigen_vecs[1][0] * eigen_vals[0], eigen_vecs[1][1] * eigen_vals[1]]
plt.quiver(vec_X, vec_Y, vec_U, vec_V, angles="xy", scale_units="xy", scale=1)
plt.show()
```




```
plt.figure(figsize=(5,5))
plt.xlim(-15,15)
plt.ylim(-15,15)
plt.xlabel("$X_1$")
plt.ylabel("$X_2$")
plt.title("Rotated Sample Points")
plt.scatter(sample_rotated[:,0], sample_rotated[:, 1])
plt.show()
```



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.

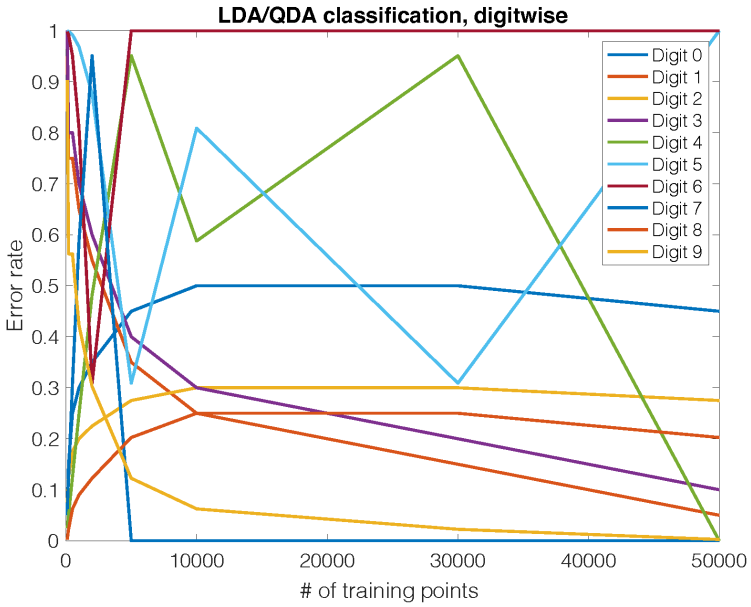


Figure 1: Sample graph with 10 plots

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.

8.1

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.

```
[17] def normalize_pixels(data):
         data = data.reshape(data.shape[0], -1)
         normalized = data/np.linalg.norm(data)
         return normalized
```

```
[18] ## Normalize data
     mnist_training_normalized = normalize_pixels(mnist_training_data)
     mnist_test_normalized = normalize_pixels(mnist_test)
```

```
[19] ## set random seed
     np.random.seed(42)
```

```
     ### Function to split data
     def split_train_val(data, labels, num_val, num_train):
         ## take arguments:
         ## training data, label data, number of data in val set, and number of data in train set
         ## return an array of val_x, train_x, val_y, train_y

         shuffled_indices = np.random.permutation(num_train+num_val)

         val_indices = shuffled_indices[:num_val]
         train_indices = shuffled_indices[num_val:]

         val_x = data[val_indices, :].reshape(num_val, -1)
         train_x = data[train_indices, :].reshape(num_train, -1)

         val_y = labels[val_indices]
         train_y = labels[train_indices]

         return [val_x, train_x, val_y, train_y]
```

```
[21] ## Spliting data
     ## Set aside 10,000 training mnist data as a validation set
     mnist_len = mnist_training_normalized.shape[0]
     mnist_num_val = 10000
     mnist_num_train = mnist_len - mnist_num_val
     mnist_split = split_train_val(mnist_training_normalized, mnist_labels, mnist_num_val, mnist_num_train)

     mnist_val_x = mnist_split[0]
     mnist_train_x = mnist_split[1]
     mnist_val_y = mnist_split[2]
     mnist_train_y = mnist_split[3]
```

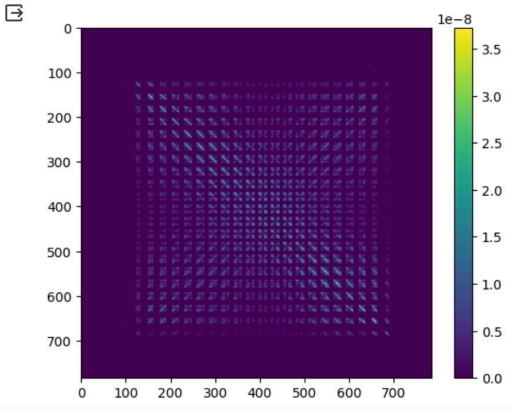
```
     ## Compute mean and variance for each class
     # Create a dictionary where key is class and value is an array of mean and variance
     means_and_variances = {}
     labels = np.unique(mnist_train_y)
     for l in labels:
         # Retrieve only data with label l
         data = mnist_train_x[mnist_train_y == l]
         data = data.reshape(data.shape[0], -1)
         # mean and variance
         mean = np.mean(data, axis = 0)
         variance = np.cov(data, rowvar = False)
         means_and_variances[l] = [mean, variance]
```

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?

8.2

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?

```
# Covariance class 0
cov_0 = means_and_variances[0][1]
cov_0 = np.abs(cov_0)
plt.imshow(cov_0)
plt.colorbar()
plt.show()
```



Digit : 0.

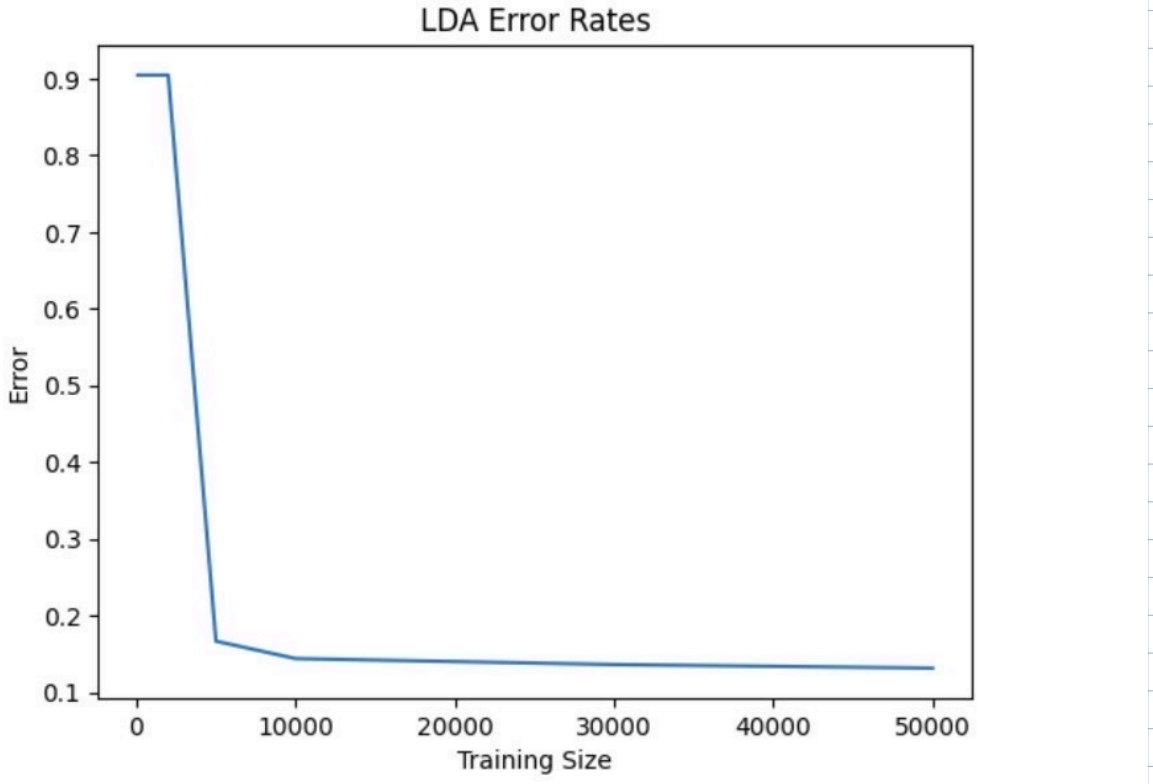
Diagonal terms are larger than off-diagonal terms. This can be interpreted the closer the pixels are to each other, the higher their correlation is, and vice versa.

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.

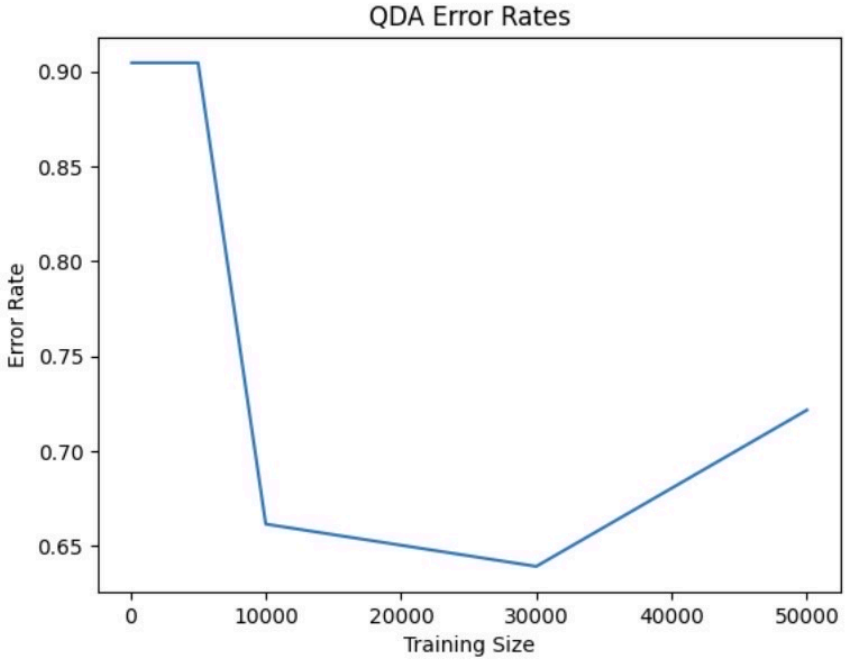
   In your implementation, you might run into issues of determinants overflowing or underflowing, or normal PDF probabilities underflowing. These problems might be solved by learning about `numpy.linalg.slogdet` and/or `scipy.stats.multivariate_normal.logpdf`.

9.3a



8.36

(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.



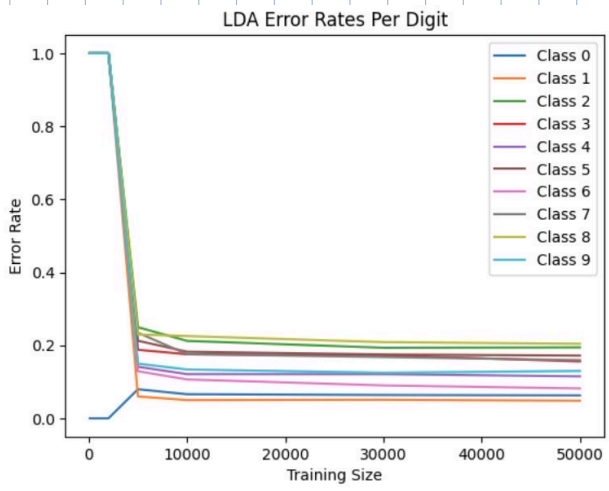
Q3.3c

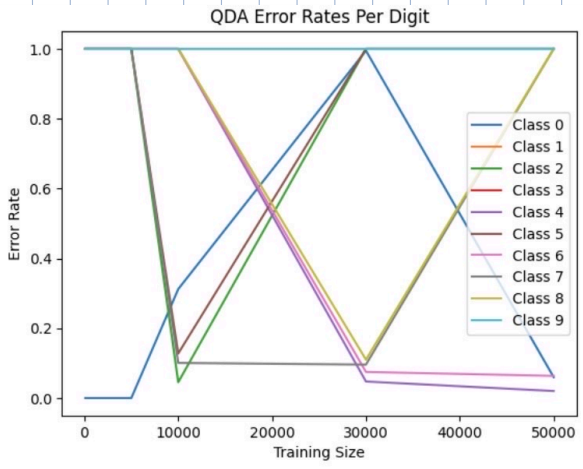
(c) (Written answer) Which of LDA and QDA performed better? Why?

In my case, LDA performed better with significant improvement on error rates. Generally, I expected QDA would perform better but it is not in this data. This can be of any reasons, perhaps QDA tend to overfit or the underlying boundary is close to linear.

8.3 d

(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.





The easiest digit to classify will have the lowest error rate among all digits.
For this reason, LDA has digit 1 and QDA has digit 4.

Q8.4

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.

Kaggle user name : lanadore

Score : 0.852

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!).

Kaggle username: lanadore

Score: 0.786

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.

Code Appendix

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!unzip -u "/content/drive/MyDrive/CS 189/hw3.zip" -d /content
```

```
## Locate this notebook inside data
%cd /content/hw3/data
```

```
/content/hw3/data
```

CS 189

Homework 2

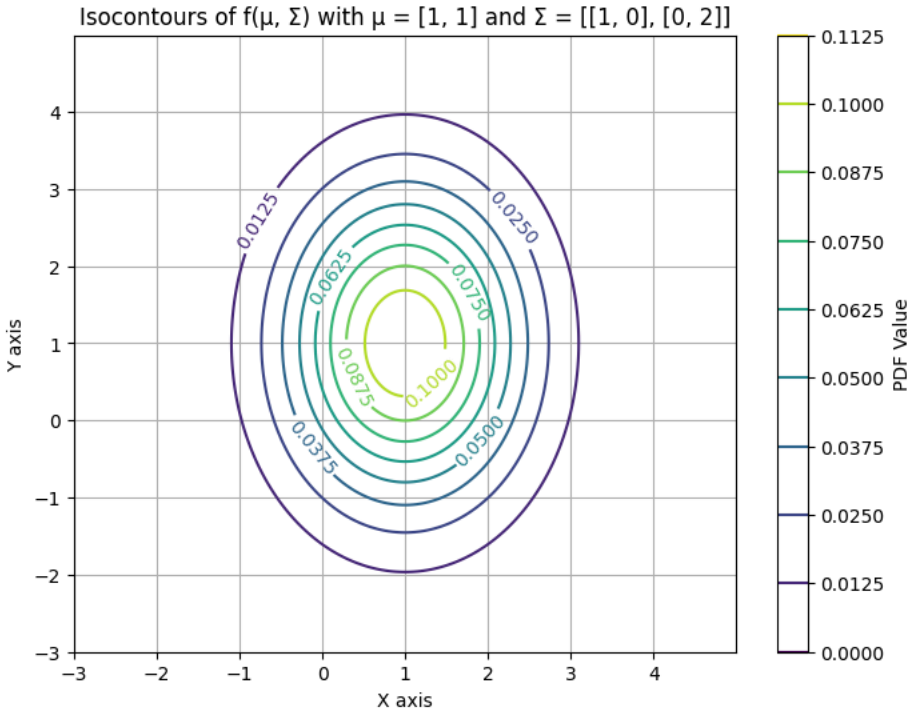
Submitted by Lan Dinh

```
# import libraries
import matplotlib.pyplot as plt
import numpy as np
import scipy
```

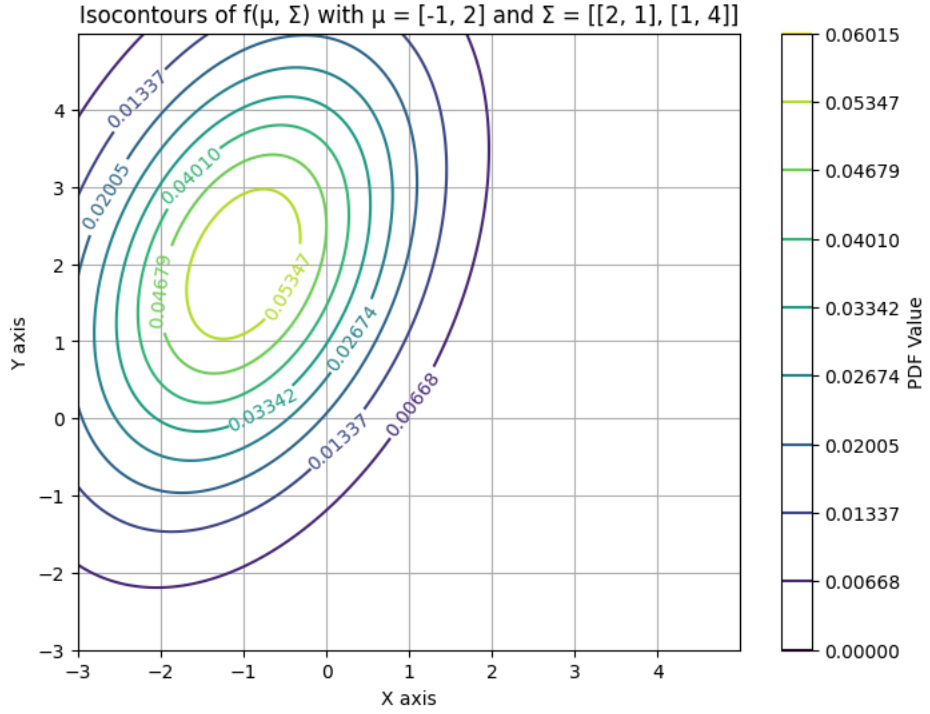
6. Isocontours of Normal Distribution

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

## 6.1

# Parameters for the distribution
mu_1 = np.array([1, 1])
sigma_1 = np.array([[1, 0], [0, 2]])

# Create a grid of points
x, y = np.mgrid[-3:5:.01, -3:5:.01]
pos = np.dstack((x, y))

# Create the multivariate normal distribution and compute its PDF over the grid
rv_1 = multivariate_normal(mu_1, sigma_1)
pdf_1 = rv_1.pdf(pos)

# Plot the isocontours for the distribution
plt.figure(figsize=(8, 6))
contour_1 = plt.contour(x, y, pdf_1, levels=np.linspace(np.min(pdf_1), np.max(pdf_1), 10))
plt.clabel(contour_1, inline=1, fontsize=10)
plt.title('Isocontours of f(μ, Σ) with μ = [1, 1] and Σ = [[1, 0], [0, 2]]')
plt.xlabel('X axis')
```

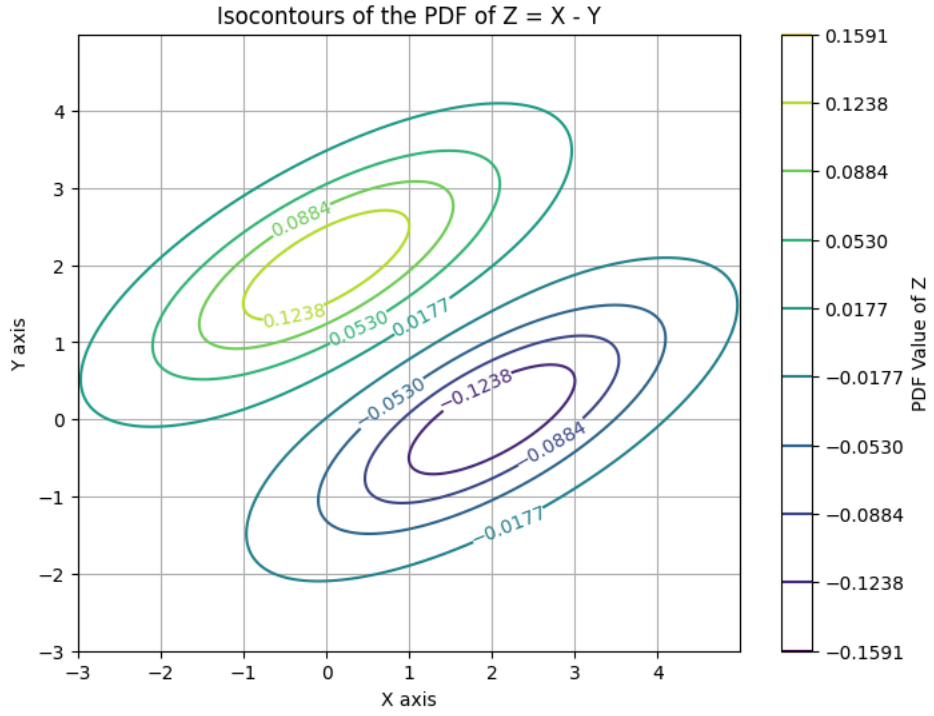
```
    plt.ylabel('Y axis')
    plt.colorbar(contour_1, label='PDF Value')
    plt.grid(True)
    plt.show()
```



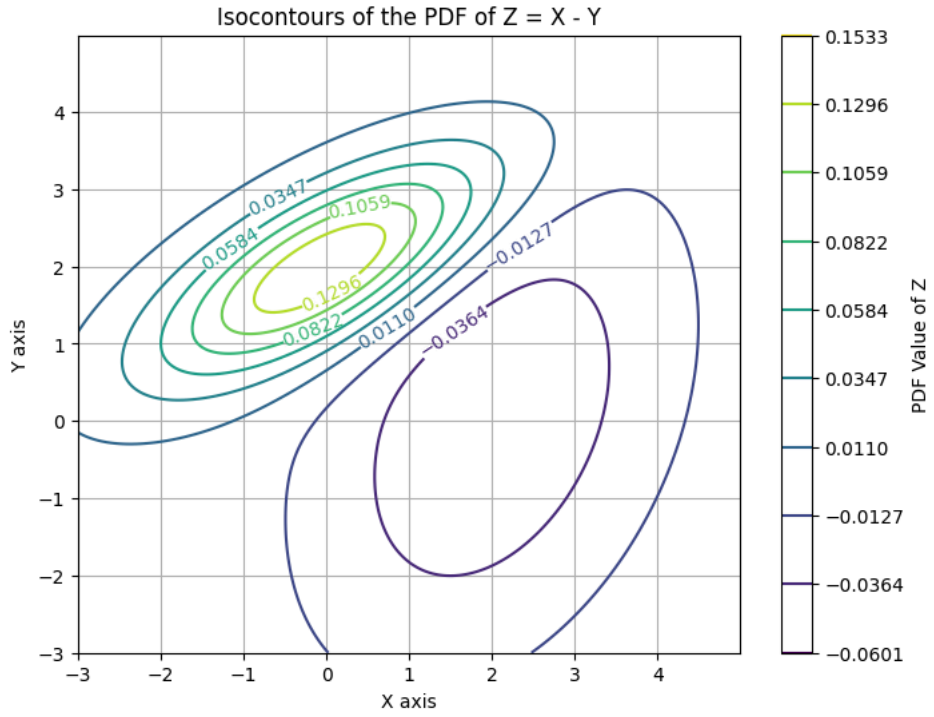
```
## 6.2
# Parameters for the distribution
mu_2 = np.array([-1, 2])
sigma_2 = np.array([[2, 1], [1, 4]])

# Create a grid of points
x, y = np.mgrid[-3:5:.01, -3:5:.01]
pos = np.dstack((x, y))

# Create the multivariate normal distribution and compute its PDF over the grid
rv_2 = multivariate_normal(mu_2, sigma_2)
pdf_2 = rv_2.pdf(pos)

# Plot the isocontours for the distribution
plt.figure(figsize=(8, 6))
contour_2 = plt.contour(x, y, pdf_2, levels=np.linspace(np.min(pdf_2), np.max(pdf_2), 10))
plt.clabel(contour_2, inline=1, fontsize=10)
plt.title('Isocontours of f(μ, Σ) with μ = [-1, 2] and Σ = [[2, 1], [1, 4]]')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.colorbar(contour_2, label='PDF Value')
plt.grid(True)
plt.show()
```



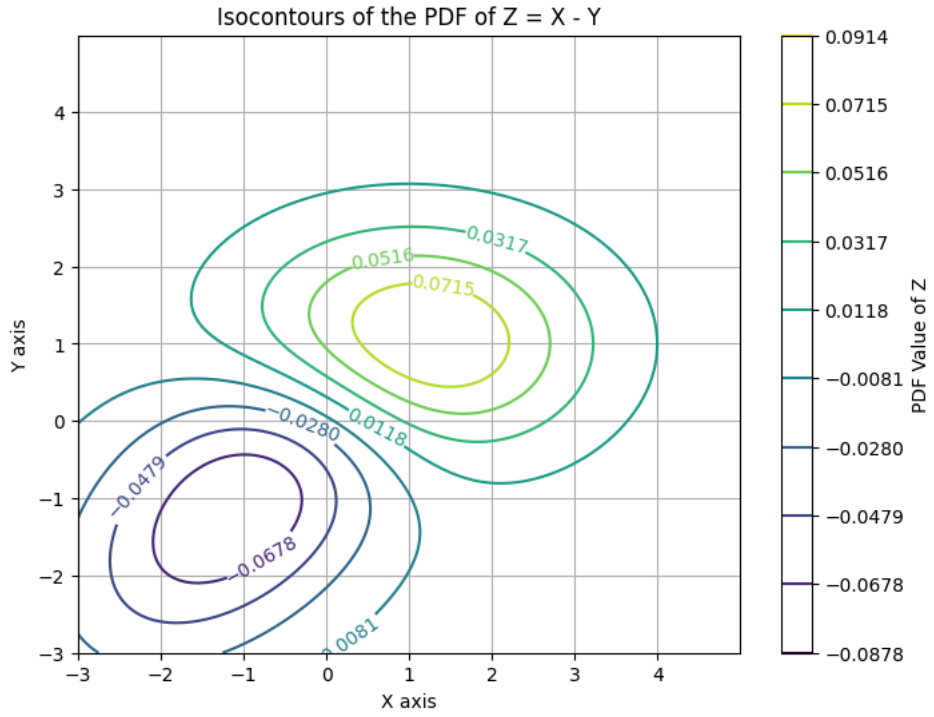
```
## 6.3

# Parameters for the distributions X and Y
mu_3a = np.array([0, 2])
mu_3b = np.array([2, 0])
sigma_3a = sigma_3b = np.array([[2, 1], [1, 1]])

# Create a grid of points
x, y = np.mgrid[-3:5:.01, -3:5:.01]
pos = np.dstack((x, y))

# Create the multivariate normal distribution for Z and compute its PDF over the grid
rv_3a = multivariate_normal(mu_3a, sigma_3a)
rv_3b = multivariate_normal(mu_3b, sigma_3b)
pdf_3 = rv_3a.pdf(pos) - rv_3b.pdf(pos)

# Plot the isocontours for the PDF of Z
plt.figure(figsize=(8, 6))
contour_3 = plt.contour(x, y, pdf_3, levels=np.linspace(np.min(pdf_3), np.max(pdf_3), 10))
plt.clabel(contour_3, inline=1, fontsize=10)
plt.title('Isocontours of the PDF of Z = X - Y')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.colorbar(contour_3, label='PDF Value of Z')
plt.grid(True)
plt.show()
```



```
## 6.4

# Parameters for the distributions X and Y
mu_4a = np.array([0, 2])
mu_4b = np.array([2, 0])
sigma_4a = np.array([[2, 1], [1, 1]])
sigma_4b = np.array([[2, 1], [1, 4]])

# Create a grid of points
x, y = np.mgrid[-3:5:.01, -3:5:.01]
pos = np.dstack((x, y))

# Create the multivariate normal distribution for Z and compute its PDF over the grid
rv_4a = multivariate_normal(mu_4a, sigma_4a)
rv_4b = multivariate_normal(mu_4b, sigma_4b)
pdf_4 = rv_4a.pdf(pos) - rv_4b.pdf(pos)

# Plot the isocontours for the PDF of Z
plt.figure(figsize=(8, 6))
contour_4 = plt.contour(x, y, pdf_4, levels=np.linspace(np.min(pdf_4), np.max(pdf_4), 10))
plt.clabel(contour_4, inline=1, fontsize=10)
plt.title('Isocontours of the PDF of Z = X - Y')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.colorbar(contour_4, label='PDF Value of Z')
plt.grid(True)
plt.show()
```



```
## 6.5

# Parameters for the distributions X and Y
mu_5a = np.array([1, 1])
mu_5b = np.array([-1, -1])
sigma_5a = np.array([[2, 0], [0, 1]])
sigma_5b = np.array([[2, 1], [1, 2]])

# Create a grid of points
x, y = np.mgrid[-3:5:.01, -3:5:.01]
pos = np.dstack((x, y))

# Create the multivariate normal distribution for Z and compute its PDF over the grid
rv_5a = multivariate_normal(mu_5a, sigma_5a)
rv_5b = multivariate_normal(mu_5b, sigma_5b)
pdf_5 = rv_5a.pdf(pos) - rv_5b.pdf(pos)

# Plot the isocontours for the PDF of Z
plt.figure(figsize=(8, 6))
contour_5 = plt.contour(x, y, pdf_5, levels=np.linspace(np.min(pdf_5), np.max(pdf_5), 10))
plt.clabel(contour_5, inline=1, fontsize=10)
plt.title('Isocontours of the PDF of Z = X - Y')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.colorbar(contour_5, label='PDF Value of Z')
plt.grid(True)
plt.show()
```



7. Eigenvectors of the Gaussian Covariance Matrix

Consider two one-dimensional random variables X1 ~ N(3, 9) and X2 ~ 1 2 X1 + N(4, 4), where N(μ, σ2) is a Gaussian distribution with mean μ and variance σ 2 . (This means that you have to draw X1 first and use it to compute a random X2.) Write a program that draws n = 100 random two-dimensional sample points from (X1, X2). For each sample point, the value of X2 is a function of the value of X1 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 R 2) of the sample

```
np.random.seed(42)
X_1 = np.random.normal(loc=3, scale=3, size=100)
Y = np.random.normal(loc=4, scale=2, size=100)
X_2 = (1/2) * X_1 +Y
sample = np.array([(x_1, x_2) for (x_1, x_2) in zip(X_1, X_2)])
```

```
sample_mean = np.mean(sample, axis = 0)
sample_mean
```

```
array([2.68846045, 5.3888394 ])
```

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).

```
cov_hat = np.cov(sample.T)
cov_hat
```

```
array([[7.42292904, 3.00253936],
       [3.00253936, 4.78474509]])
```

3. Compute the eigenvectors and eigenvalues of this covariance matrix

```
eigen_vals, eigen_vecs = np.linalg.eig(cov_hat)
eigen_vals, eigen_vecs
```

```
(array([9.38335628, 2.82431785]),
 array([[ 0.83732346, -0.54670781],
        [ 0.54670781,  0.83732346]]))
```

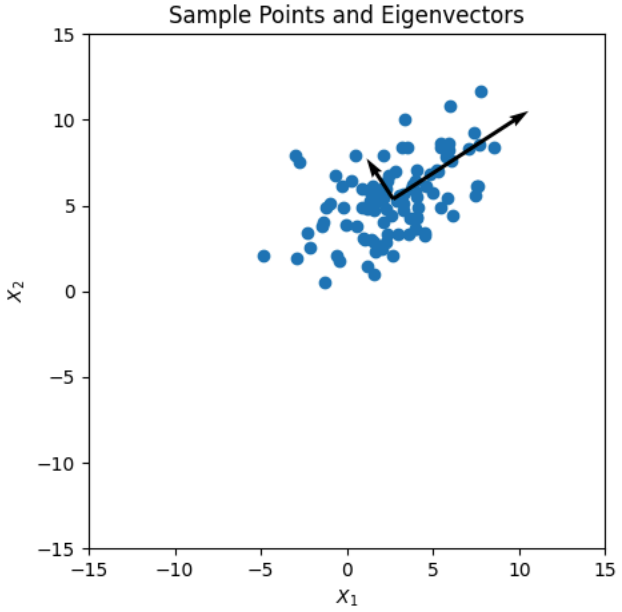
4. On a two-dimensional grid with a horizonal axis for X1 with range [−15, 15] and a vertical axis for X2 with range [−15, 15], plot

(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.

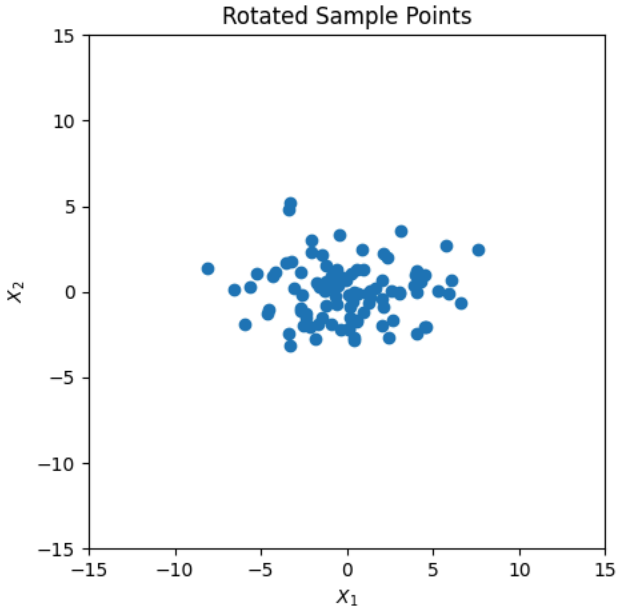
Hint: make sure your plotting software is set so the figure is square (i.e., the horizontal and vertical scales are the same). Not doing that may lead to hours of frustration!

```
plt.figure(figsize=(5,5))
plt.xlim(-15,15)
plt.ylim(-15,15)
plt.xlabel("$X_1$")
plt.ylabel("$X_2$")
plt.title("Sample Points and Eigenvectors")
plt.scatter(sample[:, 0], sample[:, 1])
vec_X = [sample_mean[0], sample_mean[0]]
vec_Y = [sample_mean[1], sample_mean[1]]
vec_U = [eigen_vecs[0][0] * eigen_vals[0], eigen_vecs[0][1] * eigen_vals[1]]
vec_V = [eigen_vecs[1][0] * eigen_vals[0], eigen_vecs[1][1] * eigen_vals[1]]
plt.quiver(vec_X, vec_Y, vec_U, vec_V, angles="xy", scale_units="xy", scale=1)
plt.show()
```



5. Let U = [v1 v2] be a 2×2 matrix whose columns are the unit eigenvectors of the covariance matrix, where v1 is the eigenvector with the larger eigenvalue. We use U ⊤ as a rotation matrix to rotate each sample point from the (X1, X2) coordinate system to a coordinate system aligned with the eigenvectors. (As U ⊤ = 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 ⊤ , giving xrotated = U ⊤ (x − μ). 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.)

```
sample_rotated = np.dot(eigen_vecs.T, (sample - sample_mean).T).T

plt.figure(figsize=(5,5))
plt.xlim(-15,15)
plt.ylim(-15,15)
plt.xlabel("$X_1$")
plt.ylabel("$X_2$")
plt.title("Rotated Sample Points")
plt.scatter(sample_rotated[:,0], sample_rotated[:, 1])
plt.show()
```



8. Gaussian Classifiers for Digits and Spam

In this problem, you will build classifiers based on Gaussian discriminant analysis. Unlike Home- work 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.

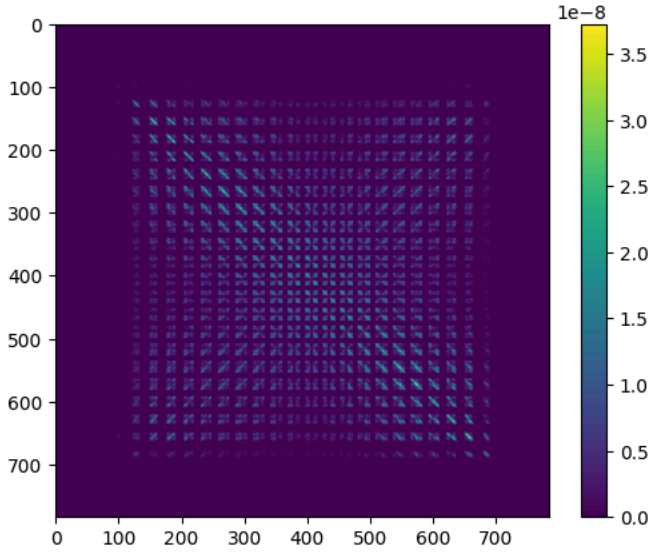
Hint: You may, and probably should, contrast-normalize the images before using their pixel values. One way to normalize is to divide the pixel values of an image by the l2-norm of its pixel values.

```
## Load mnist dataset
mnist_data = np.load("../data/mnist-data-hw3.npz")
mnist_training_data = mnist_data["training_data"]
mnist_labels = mnist_data["training_labels"]
mnist_test = mnist_data['test_data']


def normalize_pixels(data):
    data = data.reshape(data.shape[0], -1)
    normalized = data/np.linalg.norm(data)
    return normalized


## Normalize data
mnist_training_normalized = normalize_pixels(mnist_training_data)
mnist_test_normalized = normalize_pixels(mnist_test)


## set random seed
np.random.seed(42)


### Function to split data
def split_train_val(data, labels, num_val, num_train):
    ## take arguments:
    ## training data, label data, number of data in val set, and number of data in train set
    ## return an array of val_x, train_x, val_y, train_y

    shuffled_indices = np.random.permutation(num_train+num_val)

    val_indices = shuffled_indices[:num_val]
    train_indices = shuffled_indices[num_val:]

    val_x = data[val_indices, :].reshape(num_val, -1)
    train_x = data[train_indices, :].reshape(num_train, -1)

    val_y = labels[val_indices]
    train_y = labels[train_indices]

    return [val_x, train_x, val_y, train_y]
```

```
## Spliting data
## Set aside 10,000 training mnist data as a validation set
mnist_len = mnist_training_normalized.shape[0]
mnist_num_val = 10000
mnist_num_train = mnist_len - mnist_num_val
mnist_split = split_train_val(mnist_training_normalized, mnist_labels, mnist_num_val, mnist_num_train)

mnist_val_x = mnist_split[0]
mnist_train_x = mnist_split[1]
mnist_val_y = mnist_split[2]
mnist_train_y = mnist_split[3]



## Compute mean and variance for each class
# Create a dictionary where key is class and value is an array of mean and variance
means_and_variances = {}
labels = np.unique(mnist_train_y)
for l in labels:
    # Retrieve only data with label l
    data = mnist_train_x[mnist_train_y == l]
    data = data.reshape(data.shape[0], -1)
    # mean and variance
    mean = np.mean(data, axis = 0)
    variance = np.cov(data, rowvar = False)
    means_and_variances[l] = [mean, variance]
```

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?

```
# Covariance class 0
cov_0 = means_and_variances[0][1]
cov_0 = np.abs(cov_0)
plt.imshow(cov_0)
plt.colorbar()
plt.show()
```



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 N(μC, Σ) with different means μC (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 Σ −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.)

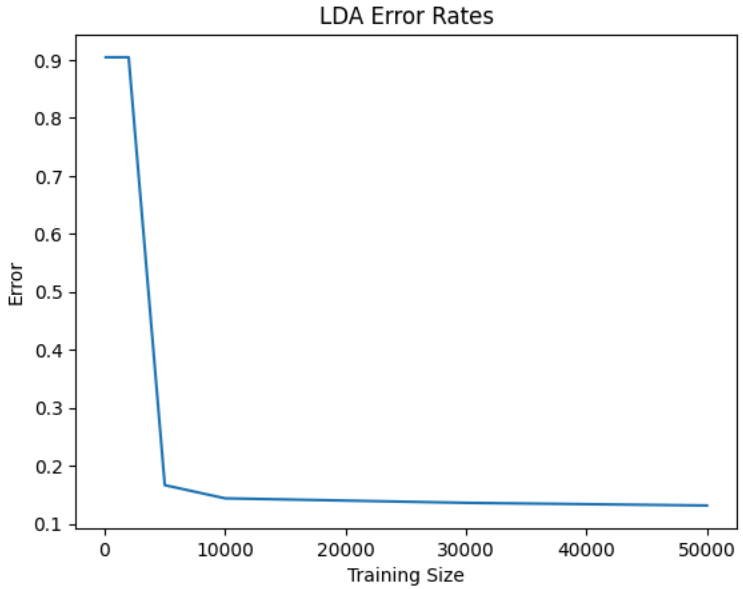
Classify each image in the validation set into one of the 10 classes. Compute the error rate (1− # points correctly classified / # total points) on the validation set and plot it over the following numbers of randomly chosen training points: 100, 200, 500, 1,000, 2,000, 5,000, 10,000, 30,000, 50,000.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

```
import math
class LDAModel():
  def __init__(self):
    super().__init__()

  def fit(self, X, y):
    # unique labels
    self.labels = np.unique(y)
    num_observe = X.shape[0]
    num_features = X.shape[1]
    # paramters include mean, weighted covariance, and prior probability for each class
    self.pars =[]

    # pooled covariance
    self.pooled_cov = np.zeros((num_features, num_features))

    for label in self.labels:
      # Retrieve data with label only
      data = X[y == label]
      # Compute parameters
      mean = np.mean(data, axis=0)
      cov = np.dot((data-mean).T, data-mean)
      self.pooled_cov = np.add(self.pooled_cov, cov)
      weighted_cov = cov/data.shape[0]
      prior_prob = data.shape[0] / num_observe
      self.pars.append((label, mean, weighted_cov, prior_prob))
    self.pooled_cov = self.pooled_cov/num_observe

  def predict(self, X):
    predicted = np.array([
        (scipy.stats.multivariate_normal.logpdf(X, allow_singular=True, cov=self.pooled_cov, mean=mean)
        + math.log(prob_prior))
        for (c, mean, cov, prob_prior) in self.pars
        ])
    predicted = self.labels[np.argmax(predicted, axis=0)].reshape(-1,)
    return predicted

  def evaluate(self, X, y):
    predicted_y = self.predict(X)
    return sum(predicted_y == y)/y.shape[0]


training_sizes = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]


LDA_errors =[]
for s in training_sizes:
  sample_train_x = mnist_train_x[:s]
  sample_train_y = mnist_train_y[:s]
  sample_LDA = LDAModel()
  sample_LDA.fit(sample_train_x, sample_train_y)

  accuracy = sample_LDA.evaluate(mnist_val_x, mnist_val_y)
  LDA_errors.append(1-accuracy)


LDA_errors
```

```
[0.9045,
 0.9045,
 0.9045,
 0.9045,
 0.9045,
 0.1664,
 0.14370000000000005,
 0.13590000000000002,
 0.13129999999999997]
```

```
plt.title("LDA Error Rates")
plt.xlabel("Training Size")
plt.ylabel("Error")
plt.plot(training_sizes, LDA_errors)
```

```
[<matplotlib.lines.Line2D at 0x7ef9c0885600>]
```



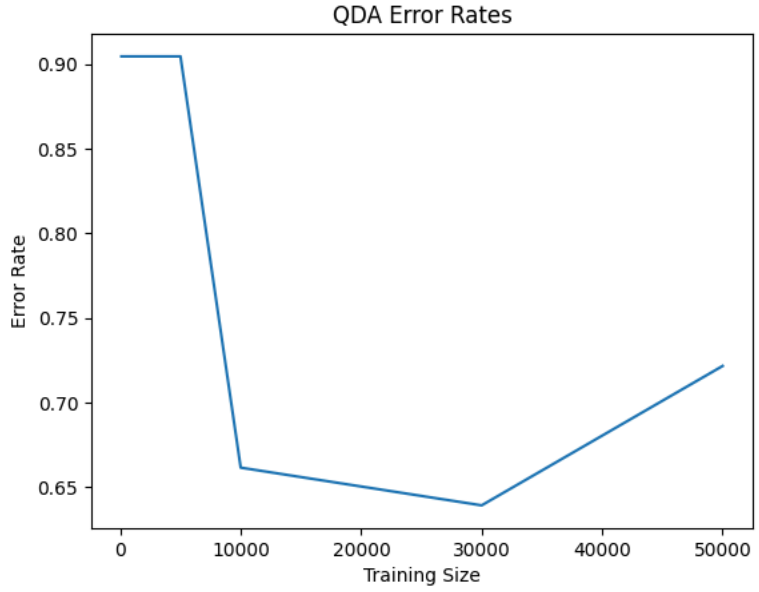
(b) (Graphs) Quadratic discriminant analysis (QDA). Model the class conditional probabilities as Gaussians N(μC, ΣC), where ΣC is the estimated covariance matrix for class C. (If any of these covariance matrices turn out singular, implement the trick you described in Q7(b). You are welcome to use validation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

```
class QDAModel():
  def __init__(self):
    super().__init__()

  def fit(self, X, y):
    # unique labels
    self.labels = np.unique(y)
    num_observe = X.shape[0]
    num_features = X.shape[1]
    # paramters include mean, weighted covariance, and prior probability for each class
    self.pars =[]

    # pooled covariance
    self.pooled_cov = np.zeros((num_features, num_features))

    for label in self.labels:
      # Retrieve data with label only
      data = X[y == label]
      # Compute parameters
      mean = np.mean(data, axis=0)
      cov = np.dot((data-mean).T, data-mean)
      self.pooled_cov = np.add(self.pooled_cov, cov)
      weighted_cov = cov/data.shape[0]
      prior_prob = data.shape[0] / num_observe
      self.pars.append((label, mean, weighted_cov, prior_prob))
    self.pooled_cov = self.pooled_cov/num_observe

  def predict(self, X):
    predicted = np.array([
        (scipy.stats.multivariate_normal.logpdf(X, allow_singular=True, cov=cov, mean=mean)
        + math.log(prob_prior))
        for (c, mean, cov, prob_prior) in self.pars
        ])
    predicted = self.labels[np.argmax(predicted, axis=0)].reshape(-1,)
    return predicted

  def evaluate(self, X, y):
    predicted_y = self.predict(X)
    return sum(predicted_y == y)/y.shape[0]


QDA_errors =[]
for s in training_sizes:
  sample_train_x = mnist_train_x[:s]
  sample_train_y = mnist_train_y[:s]
  sample_QDA = QDAModel()
  sample_QDA.fit(sample_train_x, sample_train_y)
  accuracy = sample_QDA.evaluate(mnist_val_x, mnist_val_y)
  QDA_errors.append(1-accuracy)


QDA_errors
```

```
[0.9045, 0.9045, 0.9045, 0.9045, 0.9045, 0.9045, 0.6614, 0.6391, 0.7215]
```

```
plt.title("QDA Error Rates")
plt.xlabel("Training Size")
plt.ylabel("Error Rate")
plt.plot(training_sizes, QDA_errors)
```

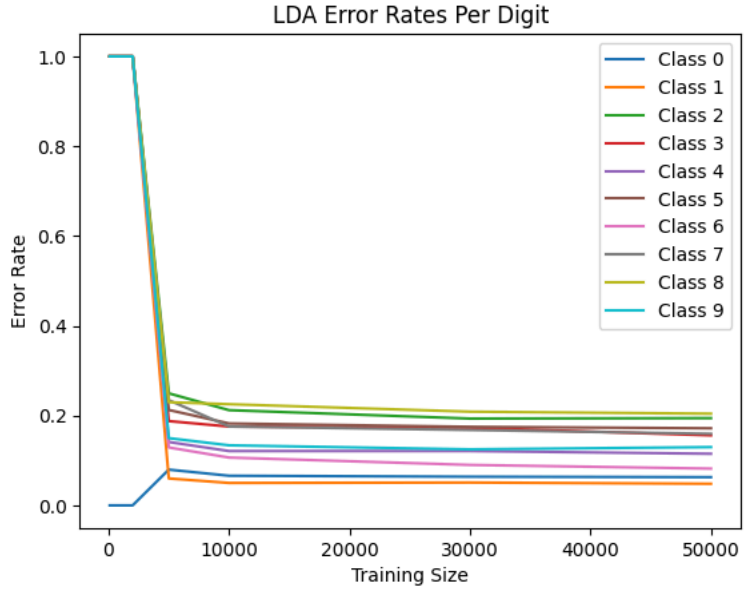
```
[<matplotlib.lines.Line2D at 0x7ef9c0c71d20>]
```



(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.

```
LDA_errors_per_digits = dict({0:[], 1:[], 2:[], 3:[], 4:[],5:[], 6:[], 7:[], 8:[], 9:[]})

for s in training_sizes:
  sample_train_x = mnist_train_x[:s]
  sample_train_y = mnist_train_y[:s]
  sample_LDA = LDAModel()
  sample_LDA.fit(sample_train_x, sample_train_y)
  predicted = sample_LDA.predict(mnist_val_x)
  for label in labels:
    modified_y = np.array([mnist_val_y[i] if mnist_val_y[i] == label else -1 for i in range(mnist_val_y.shape[0])])
    error = 1 - sum(predicted == modified_y)/sum(modified_y == label)
    LDA_errors_per_digits.get(label).append(error)



plt.title("LDA Error Rates Per Digit")
plt.xlabel("Training Size")
plt.ylabel("Error Rate")
for label in labels:
  plt.plot(training_sizes, LDA_errors_per_digits.get(label), label=f"Class {label}")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ef9c075fe80>
```




```
QDA_errors_per_digits = dict({0:[], 1:[], 2:[], 3:[], 4:[],5:[], 6:[], 7:[], 8:[], 9:[]})

for s in training_sizes:
  sample_train_x = mnist_train_x[:s]
  sample_train_y = mnist_train_y[:s]
  sample_QDA = QDAModel()
  sample_QDA.fit(sample_train_x, sample_train_y)
  predicted = sample_QDA.predict(mnist_val_x)
  for label in labels:
    modified_y = np.array([mnist_val_y[i] if mnist_val_y[i] == label else -1 for i in range(mnist_val_y.shape[0])])
    error = 1 - sum(predicted == modified_y)/sum(modified_y == label)
    QDA_errors_per_digits.get(label).append(error)


plt.title("QDA Error Rates Per Digit")
plt.xlabel("Training Size")
plt.ylabel("Error Rate")
for label in labels:
  plt.plot(training_sizes, QDA_errors_per_digits.get(label), label=f"Class {label}")
plt.legend()
```