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 $\mu_1 < b < \mu_2$.*

Section 2

$$L(z,y) = \begin{cases} 1 & z \neq y \\ 0 & o/w \end{cases}$$

1. Bayes' Optimal decision boundary:

$$P(Y=1|X) = P(Y=-1|X)$$

$$\frac{P(X|Y=1)}{P(X)} = \frac{P(X|Y=-1)}{P(X)}$$

$$P(X|Y=1) = P(X|Y=-1)$$

From gaussian:

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

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

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

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

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

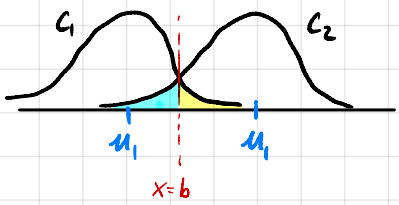
$$= \frac{\mu_2+\mu_1}{2}$$

Bayes' Decision Rule:

$$r^*(x) = \begin{cases} 1 & \text{if } x > \frac{\mu_2+\mu_1}{2} \\ -1 & o/w \end{cases}$$

given 1 is the class of $N(\mu_2, \sigma^2)$

2.



$$P_e(b) = \int_{-\infty}^{b} N(\mu_2, \sigma^2)\,dx + \int_{b}^{\infty} N(\mu_1, \sigma^2)\,dx$$

The highlighted area and eq. shows the probability of making a mistake. Essentially we are calculating the cumulative prob that the classifier will make a mistake.

The blue highlight is the prob that a sample that is C_2 will be classified as C_1. Similarly the yellow is the prob that a sample in C_1 will be classified as C_2.

3. Using the FTC

$$0 = e^{\frac{(b-\mu_2)^2}{2\sigma^2}} \cdot 1 - e^{\frac{(\infty-\mu_2)^2}{2\sigma^2}} \cdot \frac{d}{dx}\infty + e^{\frac{(\infty-\mu_1)^2}{2\sigma^2}} \cdot \frac{d}{dx}\infty - e^{\frac{(b-\mu_1)^2}{2\sigma^2}} \cdot 1$$

$$0 = e^{\frac{(b-\mu_2)^2}{2\sigma^2}} - e^{\frac{(b-\mu_1)^2}{2\sigma^2}}$$

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

The optimal value of b is the same.

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.

Section 3

1. In english: the predictor chooses i if the prob of x being in class i is greater or equal to any other class j.
   AND
   the prob of i is greater than $1 - \frac{\text{punishment for doubt}}{\text{punishment for wrong.}}$ (< 1)

Taking the complement:

$$-P(Y \neq i \mid x) \geq -\frac{\lambda_r}{\lambda_s}$$

$$\lambda_s P(Y \neq i \mid x) \leq \lambda_r$$

$$E[\text{loss} \mid \text{misclassified}] \leq E[\text{loss} \mid \text{classify as unsure}]$$

This means, if the expected loss for choosing $Y=i$ is less than the loss for being unsure, then classify Y as i.

Otherwise, if the loss of misclassifying is greater, then categorize Y as $c+1$ / unsure.

This method must obtain the minimum risk, as any other choices increases the expected loss for any x, y.

2. If $\lambda_r = 0$

$$\lambda_s P(Y \neq i \mid x) \leq 0$$

since $\lambda_s P(Y \neq i \mid x) > 0$, this case will never be true. Intuitively this makes it that the cost of getting the prediction wrong is always higher than classifying as unsure. Since all classifications have some uncertainty, that means the classifier will always classify y as $c+1$

If $\lambda_r > \lambda_s$

$$P(Y=i \mid x) \geq 1 - \frac{\lambda_r}{\lambda_s} \geq 0$$

This is guaranteed to be true, and so the classifier will never choose $c+1$ for y.
Intuitively this is when the cost of classifying as $c+1$ is always greater than getting it wrong.

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 $\hat{\mu}$?

Section 4

$X_1 \dots X_n$ where $X_i \sim N(\mu, (\frac{\sigma}{i})^2)$ $d=1$

1. $f_{X_i}(x) = \frac{1}{\sqrt{2\pi\frac{\sigma^2}{i}}} e^{-\frac{i(x_i-\mu)^2}{2\cdot\sigma^2}}$

$$\ln(f_{X_i}(x)) = -\tfrac{1}{2}\ln(2\pi) - \ln(\sigma) - \tfrac{1}{2}\ln(i) - \frac{i(x_i-\mu)^2}{2\sigma^2}$$

$$\downarrow$$

$$\ell(\mu, \sigma, x_1, \dots, x_n) = \sum_{i=1}^{n}\left(-\frac{i(x_i-\mu)^2}{2\sigma^2} - \ln(\sigma) - \tfrac{1}{2}\ln(2\pi) - \tfrac{1}{2}\ln(i)\right)$$

$$\frac{\partial}{\partial\mu}\ell(\mu,\sigma,x_1\dots x_n) = \sum_{i=1}^{n}\left(-\frac{i(x_i-\mu)^2}{2\sigma^2} - \ln(\sigma) - \tfrac{1}{2}\ln(2\pi) - \tfrac{1}{2}\ln(i)\right)$$

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

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

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

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

Find critical point: $0 = \dots$

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

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

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

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

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

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

Critical points

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

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

2. $\frac{2}{n(1+n)} \sum_{i=1}^{n} i x_i$

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

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

$= u$

$\text{Bias}(\hat{x}) = E[\hat{x}] - u$

$= 0$

$\therefore$ $\hat{u}$ is not biased.

3. $E[\hat{\sigma}^2] = \frac{1}{n}\left(E\left[\sum i x_i^2 - 2x_i u + i u^2\right]\right)$

$= \frac{1}{n}\left(E\left[\sum i x_i^2 - \sum 2x_i u + \sum i u^2\right]\right)$

$= \frac{1}{n}\left(E\left[\sum i x_i^2 - 2u \sum i x_i + \frac{u^2 n(1+n)}{2}\right]\right)$

$= \frac{1}{n}\left(E\left[\sum i x_i^2\right] - 2u \sum i u + \frac{u^2 n(1+n)}{2}\right)$

$= \frac{1}{n}\left(\sum_{i=1}^{n} i E[x_i^2] - 2u \cdot \frac{un(1+n)}{2} + \frac{u^2 n(1+n)}{2}\right)$

$= \frac{1}{n}\left(\frac{E[x_i^2] n(1+n)}{2} - \frac{u^2 n(1+n)}{2}\right)$

$= \frac{(1+n)}{2}\left(E[x_i^2] - E[x_i]^2\right)$

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

$\text{Bias}(\hat{\sigma}^2) = E[\hat{\sigma}^2] - \sigma^2$

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

$\neq 0$

$\therefore$ Variance is biased

4. $R(\hat{u}) = E[L(\hat{u}, u)]$

$= E[\hat{u}^2 - 2\hat{u}u + u^2]$

$= E[\hat{u}^2] - 2u\hat{u} + u^2$

$= u^2 - 2u\hat{u} + u^2$

$= 2u^2 - 2u\hat{u}$

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

$= E\left[\frac{4}{n^2(1+n)^2} \sum_{i=1}^{n} i x_i \cdot \sum_{i=1}^{n} i x_i\right]$

$= \frac{4}{n^2(1+n)^2} \sum_{i=1}^{n} i E[x_i] \cdot \sum_{i=1}^{n} i E[x_i]$ ($E[x_i] = u$)

$= \frac{4}{n^2(1+n)^2} \cdot \frac{un(1+n)}{2} \cdot \frac{un(1+n)}{2}$

$= u^2$

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 $y^\top X$?

Section 5

1. Collinear samples: When sample points lie on the same hyperplane.

   If the dimensions are > than observations: The cov matrix can't be full rank.

   If features are redundant: If points are the same so that the # of unique points are < d, then there will be dependent columns in cov matrix.

   Repeated Eigen-values: If samples cause the eigenvalues to be repeated.

2. I would first try to reduce dimensionality, to reduce redundant dimensions. This may change the covar. matrix a lot, but we aren't really losing anything since those dimensions were redundant.

   The other way (applied in q8), is to add a small constant across the diagonal. This # is limited by how fast the program runs. A smaller value makes the inverse have really big values, hence slowing down calculations.

3. Max f(x): Pick a x that is the unit vector of the eigen-vector with the largest corresponding eigen-value of Σ. This is direction of max variance.

   Min f(x): Similar to max f(x), but choose the smallest eigenvalue/eigenvector pair. This is the path of least variance.

4. $\text{Var}(p) = \text{Var}(y^T X) = y^T \Sigma y$

   λ_{max} tells us about a direction with high variance. This means that many points are concentrated at or on $y^T X$. If λ_{max} is small, that means the variance along $y^T X$ is small.

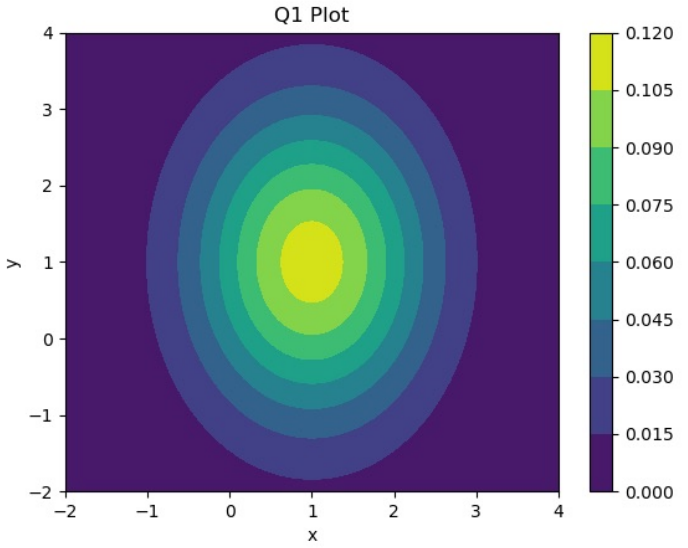
6 Isocontours of Normal Distributions

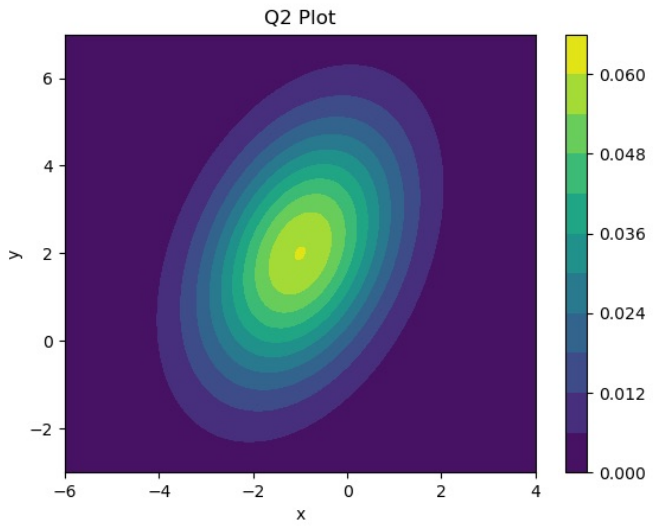
Let $f(\mu, \Sigma)$ be the probability density function of a normally distributed random variable in $\mathbb{R}^2$. Write code to plot the isocontours of the following functions, each on its own separate figure. Make sure it is clear which figure belongs to which part. You're free to use any plotting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

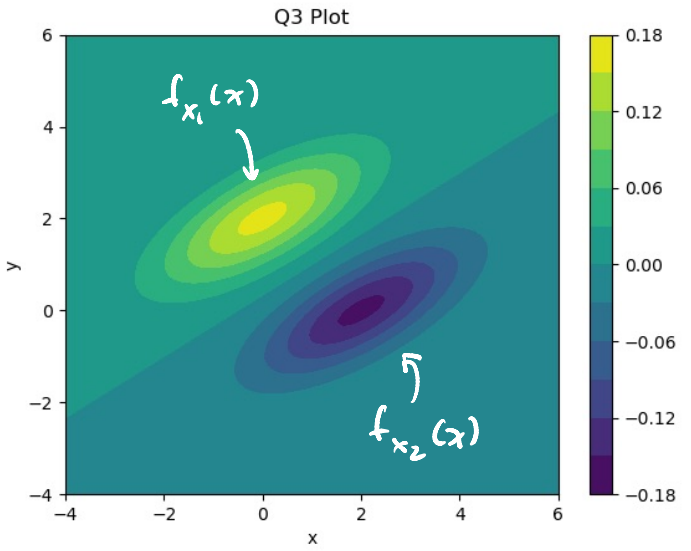
1. $f(\mu, \Sigma)$, where $\mu = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$.
2. $f(\mu, \Sigma)$, where $\mu = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$.
3. $f(\mu_1, \Sigma_1) - f(\mu_2,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

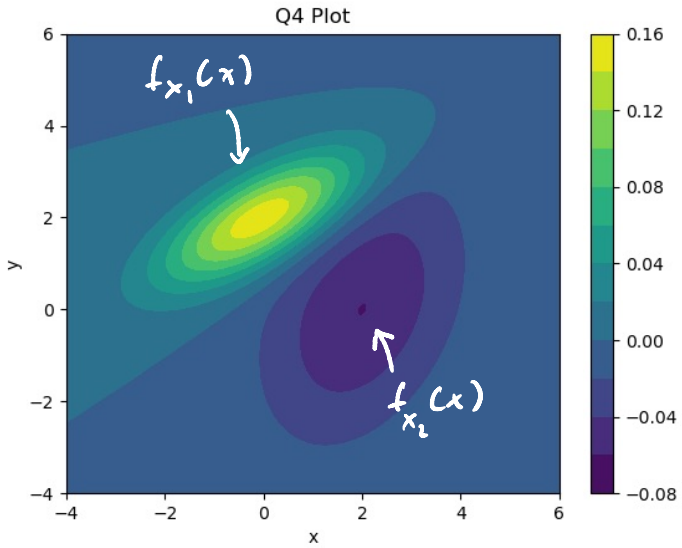
   (Yes, this is a difference between two PDFs. No, it is not itself a valid PDF. Just plot its isocontours anyway.)
4. $f(\mu_1, \Sigma_1) - f(\mu_2, \Sigma_2)$, where $\mu_1 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 2 & 1 \\ 1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

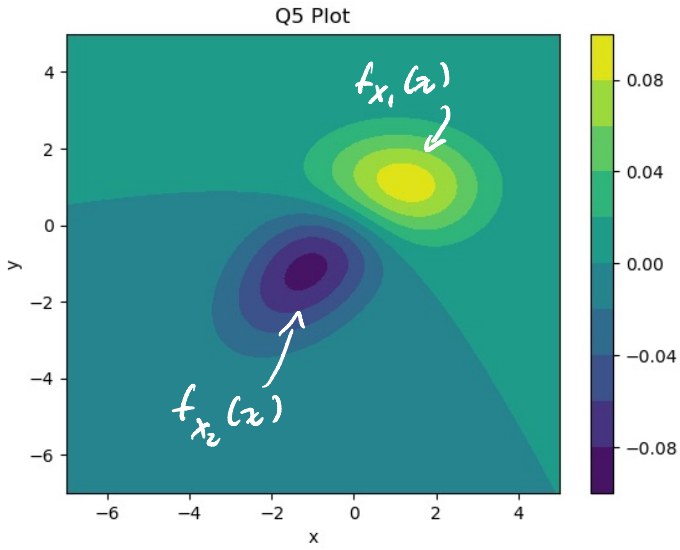
Section 6


Q1 Plot
x
y
0.120
0.105
0.090
0.075
0.060
0.045
0.030
0.015
0.000


Q2 Plot
x
y
0.060
0.048
0.036
0.024
0.012
0.000


Q3 Plot
$f_{x_1}(x)$
$f_{x_2}(x)$
x
y
0.18
0.12
0.06
0.00
−0.06
−0.12
−0.18


Q4 Plot
$f_{x_1}(x)$
$f_{x_2}(x)$
x
y
0.16
0.12
0.08
0.04
0.00
−0.04
−0.08


Q5 Plot
$f_{x_1}(x)$
$f_{x_2}(x)$
x
y
0.08
0.04
0.00
−0.04
−0.08

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

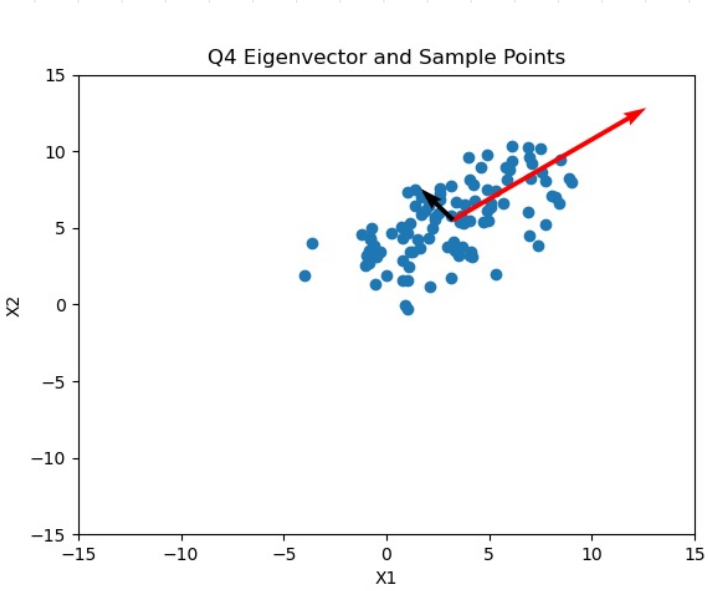
   (i) all $n = 100$ data points, and

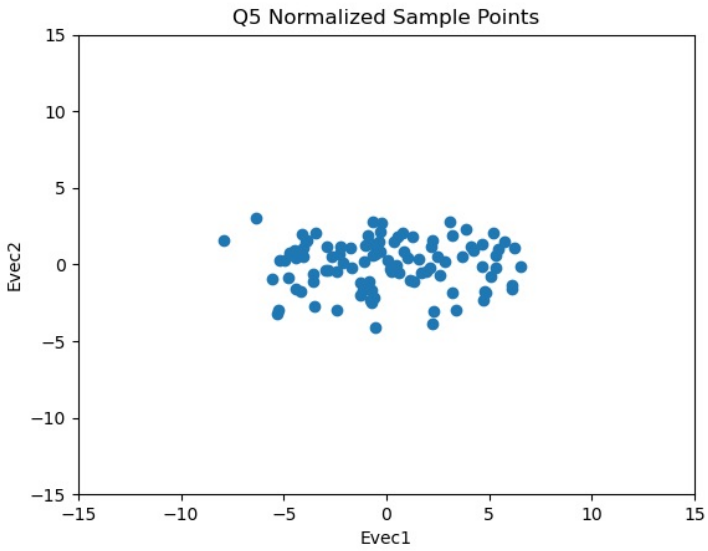
   (ii) arrows representing both covariance eigenvectors. The eigenvector arrows should originate at the mean and have magnitudes equal to their corresponding eigenvalues.

   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.

Section 7


Q4 Eigenvector and Sample Points
X2
X1
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15


Q5 Normalized Sample Points
Evec2
Evec1
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15

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.

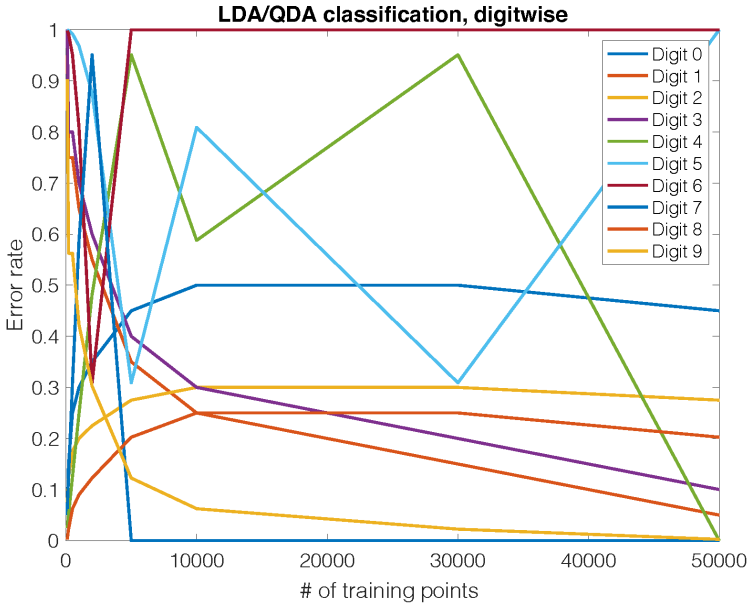


Figure 1: Sample graph with 10 plots

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!

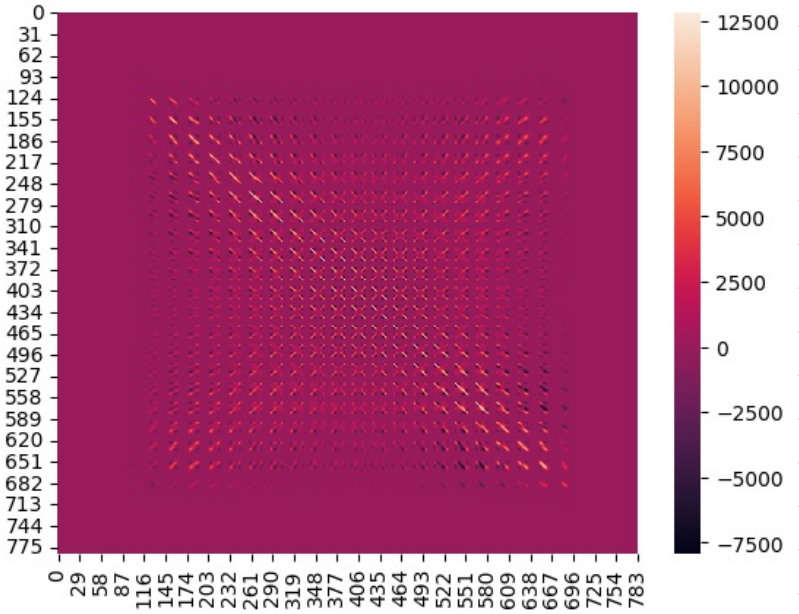
   You are welcome to compute extra features for the Kaggle competition, as long as they do not use an exterior learned model for their computation (no transfer learning!). If you do so, please describe your implementation in your assignment. Please use extra features **only** for the Kaggle portion of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

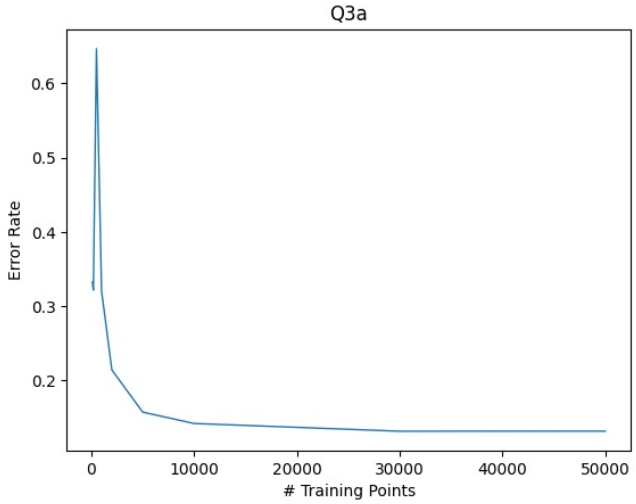
   Optional: If you use the defaults, expect relatively low classification rates. We suggest using a Bag-Of-Words model. You are encouraged to explore alternative hand-crafted features, and are welcome to use any third-party library to implement them, as long as they do not use a separate model for their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

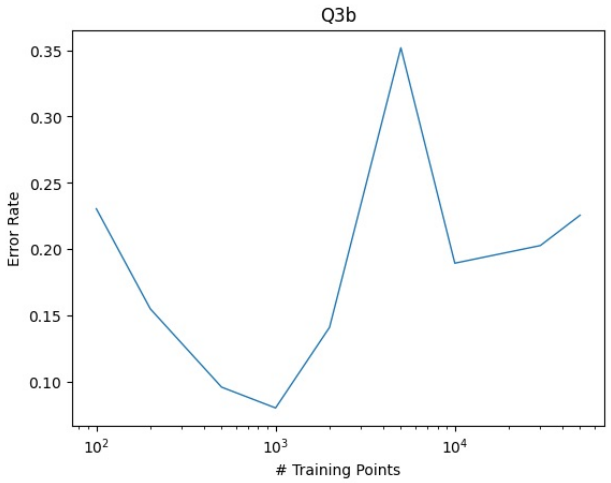
2. Looked at the covariance matrix with 0

The diagonal terms seem to be larger than the other terms. Indicating a strong signal



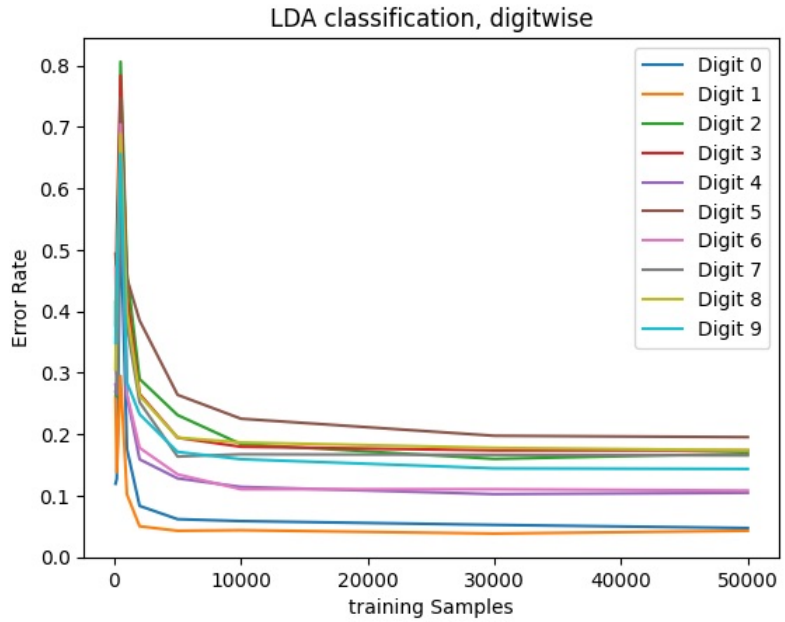
3.





c) QDA model was better at around 1000 training points. Then overfit.
LDA never overfit but was not as good.

d)



Digits that have a more common way of writing and are less complicated are easier to classify.

This is b/c there are less variance in what they look like.

4. kaggle username: Frank Jin (fj0228)

   Mnist: 0.919

5. kaggle username: Frank Jin (fj0228)
   SPAM: 0.793

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.

q6

February 23, 2024

1 Appendix

[ ]:
```
#Imports
import matplotlib.pyplot as plt
import matplotlib.ticker as tick
import numpy as np
import scipy.stats as stats
import statistics
```

[ ]:
```
np.random.seed(0)
```

[ ]:
```
def normalDistGen(mean, covar, axisRange, fineness):
    normal = stats.multivariate_normal(mean, covar)
    xUpperWindow = axisRange[0] + axisRange[2]
    yUpperWindow = axisRange[1] + axisRange[2]
    x = np.linspace(axisRange[0], xUpperWindow, fineness)
    y = np.linspace(axisRange[1], yUpperWindow, fineness)
    X, Y = np.meshgrid(x,y)
    pos = np.empty(X.shape + (2,))
    pos[:, :, 0] = X; pos[:, :, 1] = Y

    return (normal, X, Y, pos)
```

[ ]:
```
#Question 1
mean = np.array([1, 1])
covar = np.array([[1, 0], [0, 2]])
normal, X, Y, pos = normalDistGen(mean, covar, [-2, -2, 6], 500)

print(pos.shape)
#Plot Surface
# fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
# surf = ax.plot_surface(X, Y, normal.pdf(pos), cmap='viridis',
#                        linewidth=0, antialiased=False)
# fig.colorbar(surf)

#Plot Contour
fig, ax = plt.subplots(1,1)
```

```
contour = ax.contourf(X, Y, normal.pdf(pos), levels = 10, cmap = 'viridis')
fig.colorbar(contour)

plt.title("Q1 Plot")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

```
(500, 500, 2)
```

[ ]:
```
#Question 2
mean = np.array([-1, 2])
covar = np.array([[2, 1], [1, 4]])
normal, X, Y, pos = normalDistGen(mean, covar, [-6, -3, 10], 500)

#Plot
# fig, ax = plt.subplots(1,1) #plt.subplots(subplot_kw={"projection": "3d"})
# surf = ax.plot_surface(X, Y, normal.pdf(pos), cmap='viridis',
#                        linewidth=0, antialiased=False)
# fig.colorbar(surf)

#Plot Contour
fig, ax = plt.subplots(1,1)
contour = ax.contourf(X, Y, normal.pdf(pos), levels = 10, cmap = 'viridis')
fig.colorbar(contour)

plt.title("Q2 Plot")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

[ ]:
```
#Question 3
meanOne = np.array([0, 2])
meanTwo = np.array([2, 0])
covar = np.array([[2, 1], [1, 1]])
normalOne, X, Y, pos = normalDistGen(meanOne, covar, [-4, -4, 10], 500)
normalTwo, X, Y, pos = normalDistGen(meanTwo, covar, [-4, -4, 10], 500)

#Plot
# fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
# surf = ax.plot_surface(X, Y, normalOne.pdf(pos) - normalTwo.pdf(pos),␣
 ↪cmap='viridis',
#                        linewidth=0, antialiased=False)
# fig.colorbar(surf)

#Plot Contour
fig, ax = plt.subplots(1,1)
```

```
contour = ax.contourf(X, Y, normalOne.pdf(pos) - normalTwo.pdf(pos), levels =␣
 ↪10, cmap = 'viridis')
fig.colorbar(contour)

plt.title("Q3 Plot")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

[ ]:
```
#Question 4
meanOne = np.array([0, 2])
meanTwo = np.array([2, 0])
covarOne = np.array([[2, 1], [1, 1]])
covarTwo = np.array([[2, 1], [1, 4]])
normalOne, X, Y, pos = normalDistGen(meanOne, covarOne, [-4, -4, 10], 500)
normalTwo, X, Y, pos = normalDistGen(meanTwo, covarTwo, [-4, -4, 10], 500)

#Plot
# fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
# surf = ax.plot_surface(X, Y, normalOne.pdf(pos) - normalTwo.pdf(pos),␣
 ↪cmap='viridis',
#                        linewidth=0, antialiased=False)
# fig.colorbar(surf)

#Plot Contour
fig, ax = plt.subplots(1,1)
locator = tick.MaxNLocator(prune='both',nbins=5)
contour = ax.contourf(X, Y, normalOne.pdf(pos) - normalTwo.pdf(pos), levels =␣
 ↪10)
fig.colorbar(contour)

plt.title("Q4 Plot")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

[ ]:
```
#Question 5
meanOne = np.array([1, 1])
meanTwo = np.array([-1, -1])
covarOne = np.array([[2, 0], [0, 1]])
covarTwo = np.array([[2, 1], [1, 2]])
normalOne, X, Y, pos = normalDistGen(meanOne, covarOne, [-7, -7, 12], 500)
normalTwo, X, Y, pos = normalDistGen(meanTwo, covarTwo, [-7, -7, 12], 500)

#Plot
# fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
```

```
# surf = ax.plot_surface(X, Y, normalOne.pdf(pos) - normalTwo.pdf(pos),␣
 ↪cmap='viridis',
#                        linewidth=0, antialiased=False)
# fig.colorbar(surf)

#Plot Contour
fig, ax = plt.subplots(1,1)
locator = tick.MaxNLocator(prune='both',nbins=5)
contour = ax.contourf(X, Y, normalOne.pdf(pos) - normalTwo.pdf(pos), levels =␣
 ↪10)
fig.colorbar(contour)

plt.title("Q5 Plot")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

[ ]:

q7

February 23, 2024

1 Section 7

[ ]:
```
import matplotlib.pyplot as plt
import matplotlib.ticker as tick
import numpy as np
import scipy.stats as stats
```

[ ]:
```
N = 100
seed = 0
rng = np.random.Generator(np.random.PCG64(seed))
#Generating samples
xOne = rng.normal(3, 3, N)
xTwoBias = rng.normal(4, 2, N)
xTwo = 0.5*xOne + xTwoBias
```

1.1 Question 1

[ ]:
```
#Question 1
samples = np.dstack((xOne, xTwo))[0]
print("Shape of samples: ", np.shape(samples))
sampleMean = np.mean(samples, axis=0)
print("Mean of samples: ", sampleMean)

# print(xOne)
# print(xTwo)
#print('\n', samples)
```

```
Shape of samples:  (100, 2)
Mean of samples:  [3.24329008 5.52050421]
```

1.2 Question 2

[ ]:
```
#Question 2
cov = np.cov(samples.T)
print(cov)
```

```
[[8.41540931 4.51673469]
 [4.51673469 6.10325205]]
```

1.3 Question 3

[ ]:
```
#Question 3
eigenvalues, normalizedEigenvectors = np.linalg.eig(cov)
print("Normalized eigenvector: ", normalizedEigenvectors)
```

```
Normalized eigenvector:  [[ 0.78992438 -0.61320427]
 [ 0.61320427  0.78992438]]
```

1.4 Question 4

[ ]:
```
#Question 4
eigenvectors = np.array(normalizedEigenvectors)
eigenvectors[: ,0] = normalizedEigenvectors[:, 0] * eigenvalues[0]
eigenvectors[: ,1] = normalizedEigenvectors[:, 1] * eigenvalues[1]

print("Eigenvector with eigenvalue magnitude: \n", eigenvectors)
print("Eigenvalue: \n", eigenvalues)

startingPoint = np.array([sampleMean, sampleMean]).T
print("SampleMean: \n", startingPoint)

plt.scatter(xOne, xTwo)
plt.quiver(startingPoint[0], startingPoint[1], eigenvectors[0],␣
 ↪eigenvectors[1], scale= 1, scale_units = 'xy', angles = 'xy', color = ['r',␣
 ↪'k'])
plt.xlim(-15, 15)
plt.ylim(-15, 15)
plt.title("Q4 Eigenvector and Sample Points")
plt.xlabel("X1")
plt.ylabel("X2")
plt.show()
```

```
Eigenvector with eigenvalue magnitude:
 [[ 9.41721797 -1.59248604]
 [ 7.31041905  2.05142661]]
Eigenvalue:
 [11.92167023  2.59699113]
SampleMean:
 [[3.24329008 3.24329008]
 [5.52050421 5.52050421]]
```

1.5 Question 5

[ ]:
```
normSamples = samples - sampleMean
#np.shape(normalizedEigenvectors.T)
transposedNormEigenvec = normalizedEigenvectors.T
```

```
for i in range(0, len(normSamples)):
    normSamples[i] = np.dot(transposedNormEigenvec, normSamples[i])

print(np.shape(normSamples.T))
normSamplesT = normSamples.T
plt.scatter(normSamplesT[0], normSamplesT[1])
plt.xlim(-15, 15)
plt.ylim(-15, 15)
plt.title("Q5 Normalized Sample Points")
plt.xlabel("Evec1")
plt.ylabel("Evec2")
plt.show()
```

```
(2, 100)
```

[ ]:

q8

February 23, 2024

[ ]:
```
import sys
!{sys.executable} -m pip install opencv-python
!{sys.executable} -m pip install seaborn
!{sys.executable} -m pip install sklearn
!{sys.executable} -m pip install scipy
!{sys.executable} -m pip install ipympl
```

```
Requirement already satisfied: opencv-python in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages
(4.8.0.76)
Requirement already satisfied: numpy>=1.21.2 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
opencv-python) (1.25.2)


[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
Requirement already satisfied: seaborn in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages
(0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
seaborn) (1.25.2)
Requirement already satisfied: pandas>=1.2 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
seaborn) (2.1.0)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
seaborn) (3.8.2)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.47.2)
```

```
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (23.1)
Requirement already satisfied: pillow>=8 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
pandas>=1.2->seaborn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)


[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

Collecting sklearn
  Using cached sklearn-0.0.post12.tar.gz (2.6 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'error'

  error: subprocess-exited-with-error

  × Getting requirements to build wheel did not run successfully.
    exit code: 1
   > [15 lines of output]
      The 'sklearn' PyPI package is deprecated, use 'scikit-learn'
      rather than 'sklearn' for pip commands.

      Here is how to fix this error in the main use cases:
      - use 'pip install scikit-learn' rather than 'pip install sklearn'
      - replace 'sklearn' by 'scikit-learn' in your pip requirements files
        (requirements.txt, setup.py, setup.cfg, Pipfile, etc …)
      - if the 'sklearn' package is used by one of your dependencies,
```

```
        it would be great if you take some time to track which package uses
        'sklearn' instead of 'scikit-learn' and report it to their issue tracker
      - as a last resort, set the environment variable
        SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True to avoid this
error

      More information is available at
      https://github.com/scikit-learn/sklearn-pypi-package
      [end of output]

  note: This error originates from a subprocess, and is likely not a problem
with pip.
error: subprocess-exited-with-error

× Getting requirements to build wheel did not run successfully.
  exit code: 1
 > See above for output.

note: This error originates from a subprocess, and is likely not a problem with
pip.

[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

Requirement already satisfied: scipy in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages
(1.12.0)
Requirement already satisfied: numpy<1.29.0,>=1.22.4 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
scipy) (1.25.2)


[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

Requirement already satisfied: ipympl in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (0.9.3)


[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip


Requirement already satisfied: ipython<9 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (8.14.0)
Requirement already satisfied: numpy in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (1.25.2)
Requirement already satisfied: ipython-genutils in
```

```
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (0.2.0)
Requirement already satisfied: pillow in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (10.2.0)
Requirement already satisfied: traitlets<6 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (5.9.0)
Requirement already satisfied: ipywidgets<9,>=7.6.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (8.1.0)
Requirement already satisfied: matplotlib<4,>=3.4.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipympl) (3.8.2)
Requirement already satisfied: backcall in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.2.0)
Requirement already satisfied: decorator in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (5.1.1)
Requirement already satisfied: jedi>=0.16 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.19.0)
Requirement already satisfied: matplotlib-inline in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.1.6)
Requirement already satisfied: pickleshare in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (3.0.39)
Requirement already satisfied: pygments>=2.4.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (2.16.1)
Requirement already satisfied: stack-data in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.6.2)
Requirement already satisfied: colorama in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipython<9->ipympl) (0.4.6)
Requirement already satisfied: comm>=0.1.3 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipywidgets<9,>=7.6.0->ipympl) (0.1.4)
Requirement already satisfied: widgetsnbextension~=4.0.7 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipywidgets<9,>=7.6.0->ipympl) (4.0.8)
Requirement already satisfied: jupyterlab-widgets~=3.0.7 in
```

```
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
ipywidgets<9,>=7.6.0->ipympl) (3.0.8)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (4.47.2)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (23.1)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
matplotlib<4,>=3.4.0->ipympl) (2.8.2)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
jedi>=0.16->ipython<9->ipympl) (0.8.3)
Requirement already satisfied: wcwidth in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30->ipython<9->ipympl) (0.2.6)
Requirement already satisfied: six>=1.5 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
python-dateutil>=2.7->matplotlib<4,>=3.4.0->ipympl) (1.16.0)
Requirement already satisfied: executing>=1.2.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
stack-data->ipython<9->ipympl) (1.2.0)
Requirement already satisfied: asttokens>=2.1.0 in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
stack-data->ipython<9->ipympl) (2.2.1)
Requirement already satisfied: pure-eval in
c:\users\frank\appdata\local\programs\python\python311\lib\site-packages (from
stack-data->ipython<9->ipympl) (0.2.2)
```

[ ]:
```python
import cv2
import seaborn
import sys
if sys.version_info[0] < 3:
        raise Exception("Python 3 not detected.")
```

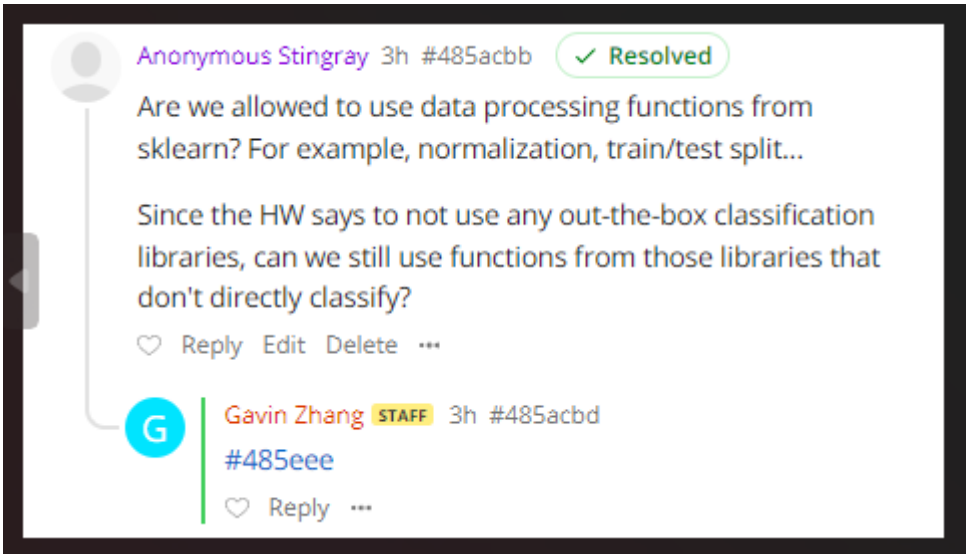
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import io
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
import scipy.stats as stats
from PIL import Image
from collections import defaultdict
```

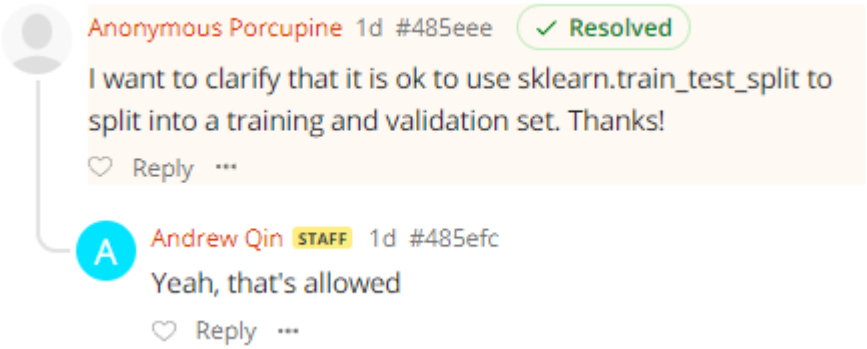
[ ]:
```
mnistData = np.load(f"../data/mnist-data-hw3.npz")
print("\nloaded %s data!" % "minst")
fields = "test_data", "training_data", "training_labels"
for field in fields:
    print(field, mnistData[field].shape)
mnistTest = mnistData["test_data"]
mnistAllData = mnistData["training_data"]
mnistAllLabels = mnistData["training_labels"]

spamData = np.load(f"../data/spam-data-hw3.npz")
print("\nloaded %s data!" % "spam")
fields = "test_data", "training_data", "training_labels"
for field in fields:
    print(field, spamData[field].shape)
spamTest = spamData["test_data"]
spamAllData = spamData["training_data"]
spamAllLabels = spamData["training_labels"]
```

```
loaded minst data!
test_data (10000, 1, 28, 28)
training_data (60000, 1, 28, 28)
training_labels (60000,)

loaded spam data!
test_data (1000, 32)
training_data (4171, 32)
training_labels (4171,)
```

[ ]:
```
SEED = 0
```

[ ]:
```
mnistXReshaped = np.reshape(mnistAllData, (60000, 28*28))
mnistTestReshaped = np.reshape(mnistTest, (10000, 28*28))
# print(mnistAllData[0][0][20])
# print(mnistXReshaped[0])
```





[ ]:
```
#Data processing
mnistXReshaped = normalize(mnistXReshaped, norm='l2')
mnistTestReshaped = normalize(mnistTestReshaped, norm='l2')
```

0.1 Question 1

[ ]:
```
#Fit Gaussian to Data
mnistTrainCount = np.zeros(10)
mnistTrainRunningSum = np.zeros((10, 784))
mnistTrainMean = np.zeros((10, 784))

for (data, label) in zip(mnistXReshaped, mnistAllLabels):
    mnistTrainCount[label] += 1
    mnistTrainRunningSum[label] += data

for i in range(0,10):
    mnistTrainMean[i] = mnistTrainRunningSum[i] / mnistTrainCount[i]
```

[ ]:
```
print(mnistTrainRunningSum.shape)
print(mnistTrainMean.shape)
print(mnistTrainCount[0])
```

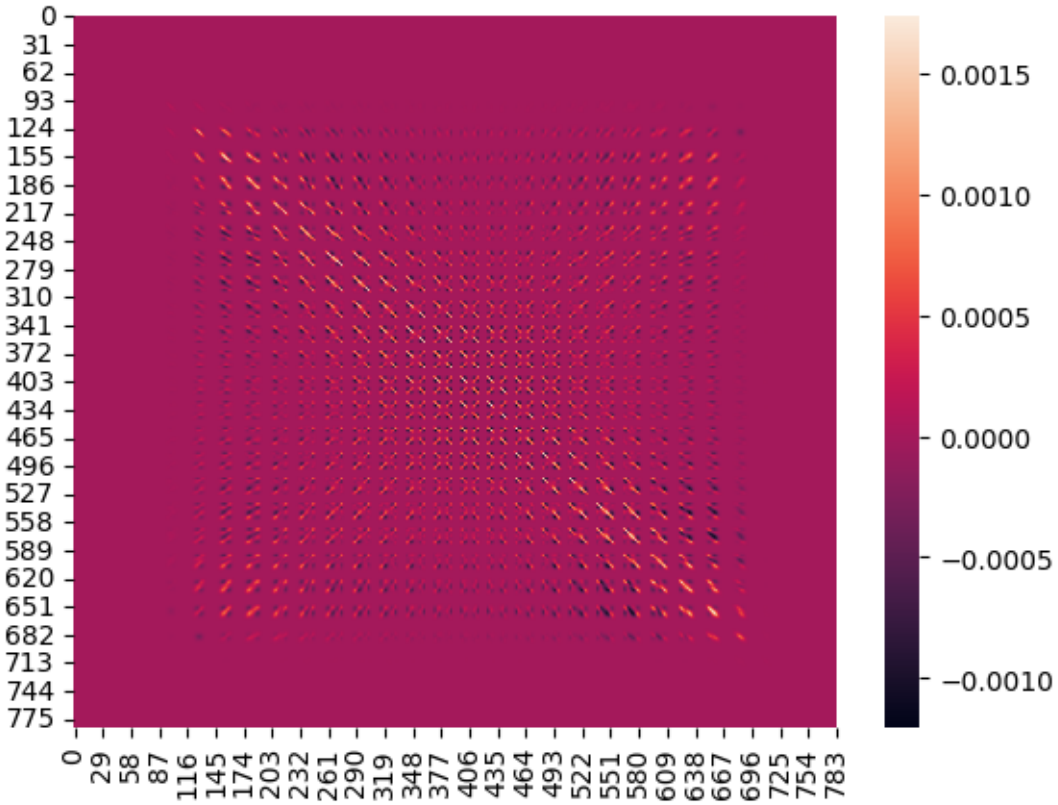
```
(10, 784)
(10, 784)
5902.0
```

[ ]:
```
#QDA Estimation of Varience
mnistTrainVar = np.zeros((10, 784, 784))
mnistTrainQDAVar = np.zeros((10, 784, 784))

for (data, label) in zip(mnistXReshaped, mnistAllLabels):
    meanDiff = data - mnistTrainMean[label]
    mnistTrainVar[label] += np.outer(meanDiff, meanDiff)

for i in range(0,10):
    mnistTrainQDAVar[i] = mnistTrainVar[i] / mnistTrainCount[i]
```

0.2 Question 2

[ ]:
```
#Heatmap
ax = seaborn.heatmap(mnistTrainQDAVar[0])
print(mnistTrainQDAVar[1].shape)
```

```
(784, 784)
```



0.3 Question 3

0.3.1 Part a

[ ]:
```
mnistTrainX, mnistTestX, mnistTrainY, mnistTestY =␣
↪train_test_split(mnistXReshaped, mnistAllLabels, test_size=10000,␣
↪random_state=SEED, shuffle=True)
```

[ ]:
```
print("mnist trainData shape: ", mnistTrainX.shape)
print("mnist testData shape: ", mnistTestX.shape)
print("mnist trainLabel shape: ", mnistTrainY.shape)
print("mnist testLabel shape: ", mnistTestY.shape)
```

```
mnist trainData shape:  (50000, 784)
mnist testData shape:  (10000, 784)
mnist trainLabel shape:  (50000,)
mnist testLabel shape:  (10000,)
```

[ ]:
```
def computeMean(inputData, inputLabels):
    #Fit Gaussian to Data
    mnistTrainCount = np.zeros(10)
    mnistTrainRunningSum = np.zeros((10, 784))
    mnistTrainMean = np.zeros((10, 784))

    for (data, label) in zip(inputData, inputLabels):
        mnistTrainCount[label] += 1
        mnistTrainRunningSum[label] += data

    for i in range(0,10):
        mnistTrainMean[i] = mnistTrainRunningSum[i] / mnistTrainCount[i]

    # print(mnistTrainRunningSum.shape)
    # print(mnistTrainMean.shape)
    # print(mnistTrainCount[0])

    return mnistTrainCount, mnistTrainMean
```

[ ]:
```
def computeClassVarience(inputData, inputLabels, mean):
    #QDA Estimation of Varience
    classVar = np.zeros((10, 784, 784))

    for (data, label) in zip(inputData, inputLabels):
        meanDiff = data - mean[label]
        classVar[label] += np.outer(meanDiff, meanDiff)
    return classVar
```

[ ]:
```
def computeLDAVar(classVar, totalPoints):
    #Find Pooled Varience
    pooledVarience = np.zeros((784, 784))
```

```
    for i in range(0, 10):
        pooledVarience += classVar[i]

    mnistTrainLDAVar = pooledVarience / totalPoints

    return mnistTrainLDAVar
```

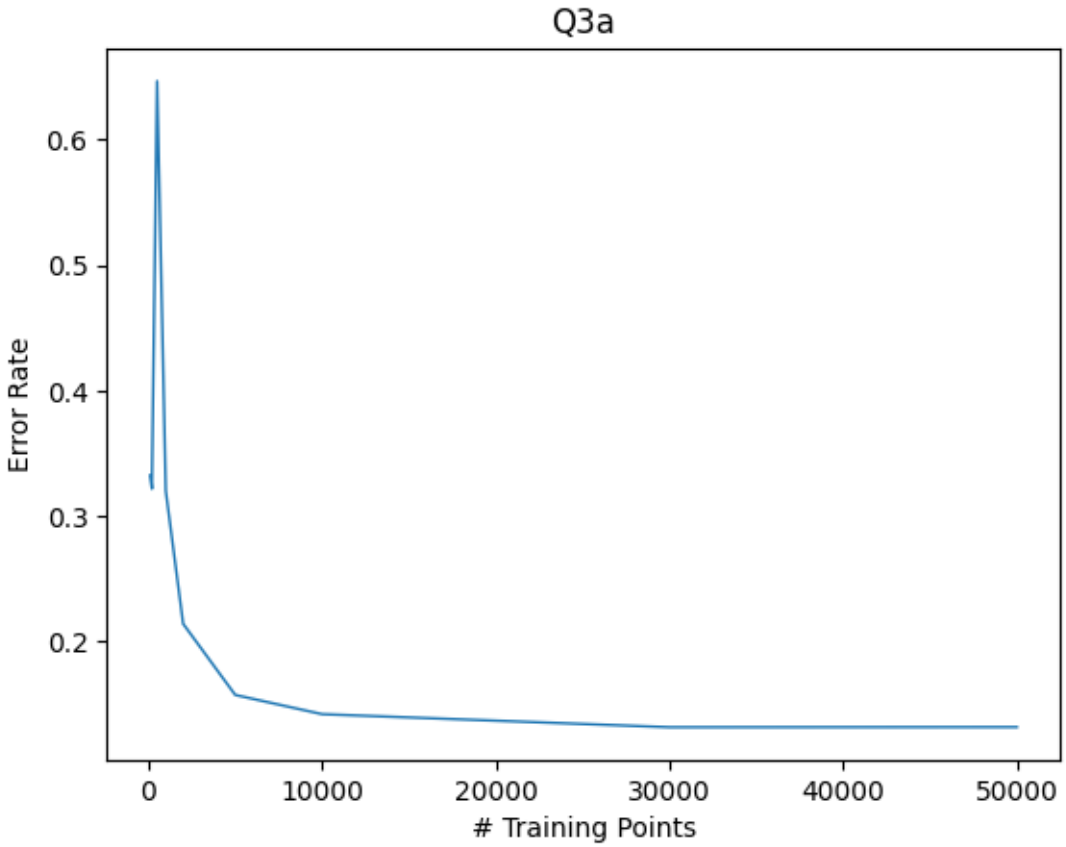
[ ]:
```
def linDiscrim(classIndex, classMean, point, classPreCalc, priorProb):
    result = np.dot(classPreCalc[classIndex], point)
    result = result - (np.dot(classPreCalc[classIndex], classMean[classIndex]))/
 ↪2
    result = result + np.log(priorProb)
    return result

errorLin = defaultdict(lambda:np.zeros((2, 10)))
for numTestPoints in [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]: #,␣
 ↪500, 1000, 2000, 5000, 10000, 30000, 30000]:
    #Train on i points
    trainSetX = mnistTrainX[:numTestPoints]
    trainSetY = mnistTrainY[:numTestPoints]

    classCount, classMean = computeMean(trainSetX, trainSetY)
    classVar = computeClassVarience(trainSetX, trainSetY, classMean)
    LDAVar = computeLDAVar(classVar, numTestPoints)
    LDAVarPseudoInv = np.linalg.pinv(LDAVar)

    classPreCalc = {}
    #Precalculate some values
    for i in range(0, 10):
        classPreCalc[i] = np.dot(classMean[i].T, LDAVarPseudoInv)

    #Using linear Disc
    #Test
    errorCount = 0
    for testPointIndex in range(0, len(mnistTestX)):
        linDisc = np.empty((10))
        for i in range(0, 10):
            pi = classCount[i]/numTestPoints
            linDisc[i] = linDiscrim(i, classMean, mnistTestX[testPointIndex],␣
 ↪classPreCalc, pi)
        pred = np.argmax(linDisc)
        if (pred != mnistTestY[testPointIndex]):
            errorLin[numTestPoints][0][mnistTestY[testPointIndex]] += 1
            errorCount+=1
        errorLin[numTestPoints][1][mnistTestY[testPointIndex]] += 1

    print(errorCount/(len(mnistTestX)))
```

```
    # errorLin.append((numTestPoints, errorCount/(errorCount+correctCount)))
```

```
0.3325
0.3218
0.6465
0.3203
0.2143
0.1576
0.1423
0.1319
0.132
```

[ ]:
```
for i in errorLin:
    print(f"{i[0]} Training Points, with {i[1]} error rate")

errorX = [pts[0] for pts in errorLin]
errorY = [pts[1] for pts in errorLin]

fig, ax = plt.subplots()
ax.plot(errorX, errorY, linewidth=1)
plt.title("Q3a")
plt.xlabel("# Training Points")
plt.ylabel("Error Rate")
plt.show()
```

```
100 Training Points, with 0.3325 error rate
200 Training Points, with 0.3218 error rate
500 Training Points, with 0.6465 error rate
1000 Training Points, with 0.3203 error rate
2000 Training Points, with 0.2143 error rate
5000 Training Points, with 0.1576 error rate
10000 Training Points, with 0.1423 error rate
30000 Training Points, with 0.1319 error rate
50000 Training Points, with 0.132 error rate
```



0.3.2 Part b

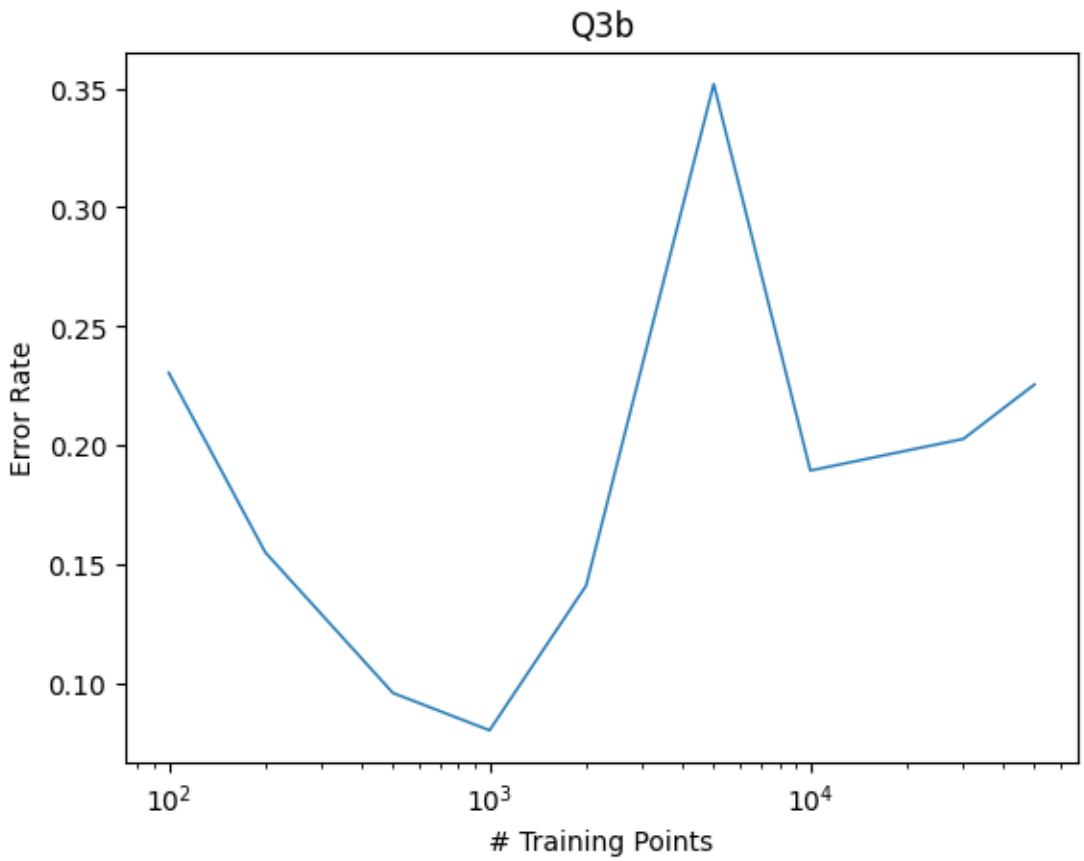
```
[ ]: def computeQDAClassVar(inputData, inputLabels, classMean, classCount):
         #QDA Estimation of Varience
         mnistTrainVar = np.zeros((10, 784, 784))
         mnistTrainQDAVar = np.zeros((10, 784, 784))

         for (data, label) in zip(inputData, inputLabels):
             meanDiff = data - classMean[label]
             mnistTrainVar[label] += np.outer(meanDiff, meanDiff)
             #print("mnistTrain: ", mnistTrainVar[label][300])

         for i in range(0,10):
             mnistTrainQDAVar[i] = (mnistTrainVar[i] / classCount[i])
             mnistTrainQDAVar[i] = mnistTrainQDAVar[i] + 1e-10*np.identity(784)
             #print(mnistTrainQDAVar)
             #print(mnistTrainQDAVar[i][2])
         return mnistTrainQDAVar
```

```
[ ]: def quadDiscrim(index, inverseQDAVar, pointMeanDiff):
         result = (np.linalg.multi_dot([pointMeanDiff[index].T,␣
      ↪inverseQDAVar[index], pointMeanDiff[index]]))/(-2)
         return result

     error = []
     for numTestPoints in [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]: #,␣
      ↪500, 1000, 2000, 5000, 10000, 30000, 30000]:
         #Train on i points
         trainSetX = mnistTrainX[:numTestPoints]
         trainSetY = mnistTrainY[:numTestPoints]

         # print(trainSetX.shape)
         # print(trainSetY.shape)

         classCount, classMean = computeMean(trainSetX, trainSetY)
         QDAClassVar = computeQDAClassVar(trainSetX, trainSetY, classMean,␣
      ↪classCount)

         preCalc = np.empty((10))
         inverseQDAVar = np.empty((10, 784, 784))
         pi = np.empty((10))

         for i in range(0,10):
             #print(np.linalg.slogdet(QDAClassVar[i])[1])
             preCalc[i]= (np.linalg.slogdet(QDAClassVar[i])[1])/(-2) + np.
      ↪log(classCount[i]/numTestPoints)
             inverseQDAVar[i] = np.linalg.inv(QDAClassVar[i])
             pi[i] = classCount[i]/numTestPoints

         # print("classmean: ", classMean.shape)
         # print(mnistTestX[testPointIndex].shape)
         #Using quadratic Disc
         #Test
         errorCount = 0
         correctCount = 0
         for testPointIndex in range(0, len(mnistTestX)):
             print(testPointIndex)
             point = mnistTestX[testPointIndex]
             pointMeanDiff = point - classMean
             # print("pointmeandiff: ", pointMeanDiff.shape)
             quadDisc = np.empty(10)
             for i in range(0,10):
                 quadDisc[i] = quadDiscrim(i, inverseQDAVar, pointMeanDiff)
             quadDisc = quadDisc + preCalc

             pred = np.argmax(quadDisc)
```

```
        if (pred != mnistTestY[testPointIndex]):
            errorCount += 1
        else:
            correctCount += 1
    print(errorCount/(errorCount+correctCount))
    error.append((numTestPoints, errorCount/(errorCount+correctCount)))
```

[ ]:
```
for i in error:
    print(f"{i[0]} Training Points, with {i[1]} error rate")

errorX = [pts[0] for pts in error]
errorY = [pts[1] for pts in error]

fig, ax = plt.subplots()
ax.plot(errorX, errorY, linewidth=1)
plt.title("Q3b")
plt.xlabel("# Training Points")
plt.xscale("log")
plt.ylabel("Error Rate")
plt.show()
```

```
100 Training Points, with 0.2304 error rate
200 Training Points, with 0.1549 error rate
500 Training Points, with 0.0958 error rate
1000 Training Points, with 0.0801 error rate
2000 Training Points, with 0.1409 error rate
5000 Training Points, with 0.3518 error rate
10000 Training Points, with 0.1893 error rate
30000 Training Points, with 0.2026 error rate
50000 Training Points, with 0.2255 error rate
```

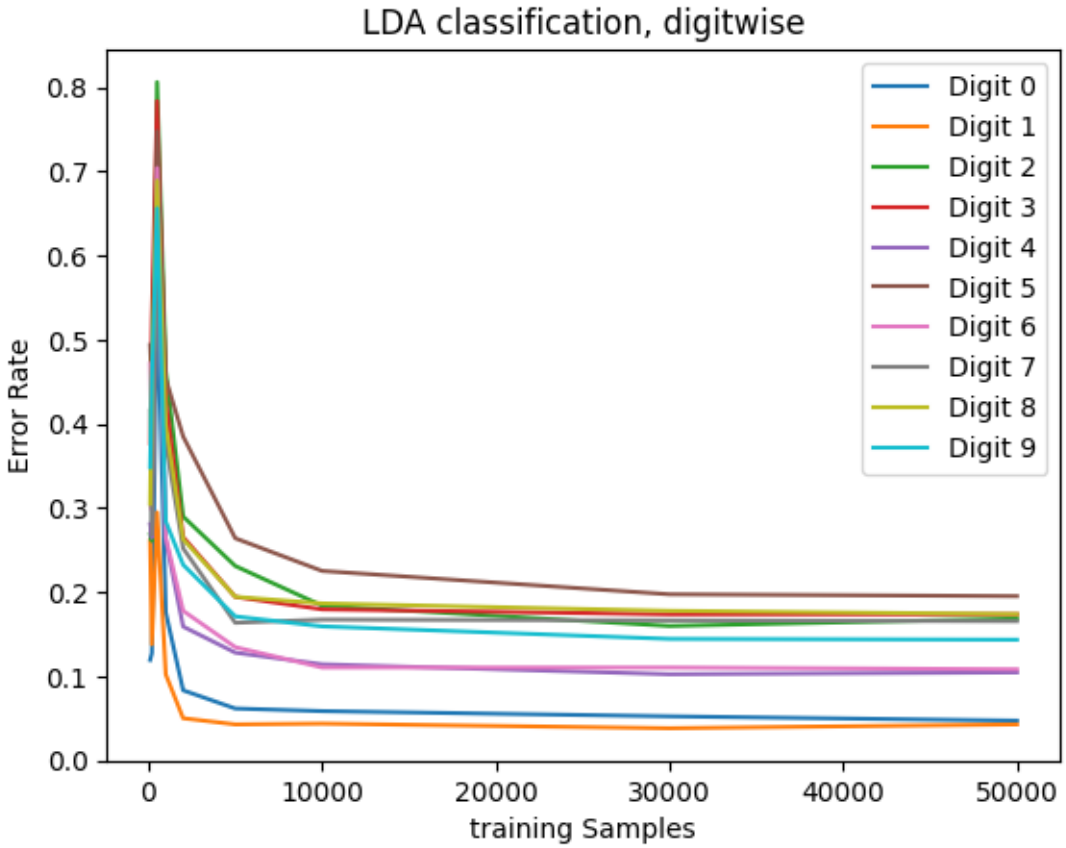


0.3.3 Part d

[ ]:
```
errorRate = np.zeros((9, 10))
numSample = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
for index in range(0,9):
    errorRate[index] = errorLin[numSample[index]][0]/
 ↪errorLin[numSample[index]][1]
print(errorRate)
errorRate = errorRate.T
for index in range(0,10):
    plt.plot(numSample, errorRate[index], label = f"Digit {index}")
    plt.legend()
plt.title("LDA classification, digitwise")
plt.xlabel("training Samples")
plt.ylabel("Error Rate")
plt.show()
```

```
[[0.11946447 0.26118721 0.26878613 0.37709773 0.28098291 0.49389567
  0.47157895 0.41580433 0.30498534 0.34851485]
 [0.1277034  0.13789954 0.261079   0.52517275 0.26495726 0.43840178
```

```
 0.29789474 0.26810913 0.46236559 0.45148515]
[0.56024717 0.29497717 0.80635838 0.78381046 0.65277778 0.74805771
 0.70421053 0.60677328 0.68914956 0.65643564]
[0.17610711 0.10228311 0.46339114 0.42941757 0.26282051 0.45394007
 0.26526316 0.37723424 0.3998045  0.28415842]
[0.08341916 0.05022831 0.28998073 0.26554788 0.15918803 0.38512764
 0.17789474 0.25211665 0.2629521  0.23267327]
[0.06179197 0.04292237 0.23121387 0.19447187 0.12820513 0.26415094
 0.13473684 0.16368768 0.1945259  0.17128713]
[0.05870237 0.04383562 0.18400771 0.17966436 0.11431624 0.22530522
 0.11052632 0.16745061 0.18670577 0.15940594]
[0.05252317 0.03835616 0.15992293 0.17374136 0.1025641  0.19755827
 0.11052632 0.16650988 0.17790811 0.14455446]
[0.04737384 0.04292237 0.16859345 0.17374136 0.10470085 0.19533851
 0.10842105 0.16556914 0.17399804 0.14356436]]
```



0.4 Question 4

[ ]:
```
#Train on i points
trainSetX = mnistTrainX[:1000]
trainSetY = mnistTrainY[:1000]

# print(trainSetX.shape)
# print(trainSetY.shape)

classCount, classMean = computeMean(trainSetX, trainSetY)
QDAClassVar = computeQDAClassVar(trainSetX, trainSetY, classMean, classCount)

preCalc = np.empty((10))
inverseQDAVar = np.empty((10, 784, 784))
pi = np.empty((10))

for i in range(0,10):
    #print(np.linalg.slogdet(QDAClassVar[i])[1])
    preCalc[i]= (np.linalg.slogdet(QDAClassVar[i])[1])/(-2) + np.
 ↪log(classCount[i]/numTestPoints)
    inverseQDAVar[i] = np.linalg.inv(QDAClassVar[i])
    pi[i] = classCount[i]/numTestPoints

# print("classmean: ", classMean.shape)
# print(mnistTestX[testPointIndex].shape)
#Using quadratic Disc
#Test
errorCount = 0
correctCount = 0
pred = []
for testIndex in range(0, len(mnistTestReshaped)):
    print(testIndex)
    point = mnistTestReshaped[testIndex]
    pointMeanDiff = point - classMean
    quadDisc = np.empty(10)
    for i in range(0,10):
        quadDisc[i] = quadDiscrim(i, inverseQDAVar, pointMeanDiff)
    quadDisc = quadDisc + preCalc

    pred.append(np.argmax(quadDisc))
```

[ ]:
```
import pandas as pd
list = [*range(1, len(pred)+1)]
outputDict = {"Id":list, "Category": pred}
df = pd.DataFrame(outputDict)
df.to_csv('mnistResult.csv', index=False)
```

q9spam

February 23, 2024

[ ]:
```
import cv2
import seaborn
import sys
if sys.version_info[0] < 3:
        raise Exception("Python 3 not detected.")
import numpy as np
import matplotlib.pyplot as plt
from scipy import io
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
import scipy.stats as stats
from PIL import Image
from collections import defaultdict

spamData = np.load(f"../data/spam-data-hw3.npz")
print("\nloaded %s data!" % "spam")
fields = "test_data", "training_data", "training_labels"
for field in fields:
    print(field, spamData[field].shape)
spamTest = spamData["test_data"]
spamAllData = spamData["training_data"]
spamAllLabels = spamData["training_labels"]
```

```
loaded spam data!
test_data (1000, 32)
training_data (4171, 32)
training_labels (4171,)
```

[ ]:
```
SEED = 0
```

[ ]:
```
spamTrainX, spamTestX, spamTrainY, spamTestY = train_test_split(spamAllData,␣
↪spamAllLabels, test_size=500, random_state=SEED, shuffle=True)
```

[ ]:
```
print("spam trainData shape: ", spamTrainX.shape)
print("spam testData shape: ", spamTestX.shape)
print("spam trainLabel shape: ", spamTrainY.shape)
print("spam testLabel shape: ", spamTestY.shape)
```

```
spam trainData shape:  (3671, 32)
spam testData shape:  (500, 32)
spam trainLabel shape:  (3671,)
spam testLabel shape:  (500,)
```

[ ]:
```python
def computeMean(inputData, inputLabels):
    #Fit Gaussian to Data
    spamTrainCount = np.zeros(2)
    spamTrainRunningSum = np.zeros((2, 32))
    spamTrainMean = np.zeros((2, 32))

    for (data, label) in zip(inputData, inputLabels):
        spamTrainCount[label] += 1
        spamTrainRunningSum[label] += data

    for i in range(0,2):
        spamTrainMean[i] = spamTrainRunningSum[i] / spamTrainCount[i]

    # print(mnistTrainRunningSum.shape)
    # print(mnistTrainMean.shape)
    # print(mnistTrainCount[0])

    return spamTrainCount, spamTrainMean
```

[ ]:
```python
def computeClassVarience(inputData, inputLabels, mean):
    #QDA Estimation of Varience
    classVar = np.zeros((10, 32, 32))

    for (data, label) in zip(inputData, inputLabels):
        meanDiff = data - mean[label]
        classVar[label] += np.outer(meanDiff, meanDiff)
    return classVar
```

[ ]:
```python
def computeLDAVar(classVar, totalPoints):
    #Find Pooled Varience
    pooledVarience = np.zeros((32, 32))
    for i in range(0, 2):
        pooledVarience += classVar[i]

    spamTrainLDAVar = pooledVarience / totalPoints

    return spamTrainLDAVar
```

[ ]:
```python
def linDiscrim(classIndex, classMean, point, classPreCalc, priorProb):
    result = np.dot(classPreCalc[classIndex], point)
    result = result - (np.dot(classPreCalc[classIndex], classMean[classIndex]))/
 ↪2
```

```
    result = result + np.log(priorProb)
    return result

errorLin = []
for numTestPoints in [100, 200, 500, 1000, 2000, 3671]:
    #Train on i points
    trainSetX = spamTrainX[:numTestPoints]
    trainSetY = spamTrainY[:numTestPoints]

    # print(trainSetX.shape)
    # print(trainSetY.shape)

    classCount, classMean = computeMean(trainSetX, trainSetY)
    classVar = computeClassVarience(trainSetX, trainSetY, classMean)
    LDAVar = computeLDAVar(classVar, numTestPoints)
    LDAVarPseudoInv = np.linalg.pinv(LDAVar)

    # print(classMean.shape)
    # print(LDAVarPseudoInv.shape)

    classPreCalc = {}
    #Precalculate some values
    for i in range(0, 2):
        classPreCalc[i] = np.dot(classMean[i].T, LDAVarPseudoInv)

    #Using linear Disc: Test
    errorCount = 0
    correctCount = 0
    for testPointIndex in range(0, len(spamTestX)):
        linDisc = np.empty((2))
        for i in range(0, 2):
            pi = classCount[i]/numTestPoints
            #print(pi)
            linDisc[i] = linDiscrim(i, classMean, spamTestX[testPointIndex],␣
 ↪classPreCalc, pi)
        pred = np.argmax(linDisc)
        if (pred != spamTestY[testPointIndex]):
            errorCount += 1
        else:
            correctCount += 1
    print(1-(correctCount/(errorCount+correctCount)))
    errorLin.append((numTestPoints, errorCount/(errorCount+correctCount)))
```

```
0.252
0.19799999999999995
0.18400000000000005
0.17800000000000005
0.17400000000000004
```

0.18000000000000005

[ ]:
```
for numTestPoints in [2000]:
    #Train on i points
    trainSetX = spamTrainX[:numTestPoints]
    trainSetY = spamTrainY[:numTestPoints]

    # print(trainSetX.shape)
    # print(trainSetY.shape)

    classCount, classMean = computeMean(trainSetX, trainSetY)
    classVar = computeClassVarience(trainSetX, trainSetY, classMean)
    LDAVar = computeLDAVar(classVar, numTestPoints)
    LDAVarPseudoInv = np.linalg.pinv(LDAVar)

    # print(classMean.shape)
    # print(LDAVarPseudoInv.shape)

    classPreCalc = {}
    #Precalculate some values
    for i in range(0, 2):
        classPreCalc[i] = np.dot(classMean[i].T, LDAVarPseudoInv)

    pred = []
    for testPointIndex in range(0, len(spamTest)):
        linDisc = np.empty((2))
        for i in range(0, 2):
            pi = classCount[i]/numTestPoints
            linDisc[i] = linDiscrim(i, classMean, spamTest[testPointIndex],␣
↪classPreCalc, pi)
        pred.append(np.argmax(linDisc))
```

[ ]:
```
print(pred)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
```

```
0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0]
```

[ ]:
```
import pandas as pd
list = [*range(1, len(pred)+1)]
outputDict = {"Id":list, "Category": pred}
df = pd.DataFrame(outputDict)
df.to_csv('spamResult.csv', index=False)
```

[ ]:
```

```