CS 189 Introduction to Machine Learning
Spring 2021 Jonathan Shewchuk

HW3

Due: Wednesday, February 24 at 11:59 pm

This homework consists of coding assignments and math problems.
Begin 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/c/spring21-cs189-hw3-mnist
   - SPAM: https://www.kaggle.com/c/spring21-cs189-hw3-spam
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".
   - **Set a seed for all pseudo-random numbers generated in your code.** This ensures your results are replicated when readers run your code.
   - Include a README with your name, student ID, the values of random seed (above) you used, and any instructions for compilation.
   - Do NOT provide 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 or b) compiles but does not produce the file submitted to Kaggle.

4. 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.

1 Honor Code

Declare and sign the following statement (Mac Preview 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 $f(x \mid C_i) \sim \mathcal{N}(\mu_i, \sigma^2)$ for a two-class, one-dimensional classification problem with classes C_1 and C_2, $P(C_1) = P(C_2) = 1/2$, 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.

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 $\mu_1 < b < \mu_2$.*

Solution:

1.

$$\begin{array}{ccccc} P(C_1 \mid x) & = & P(C_2 \mid x) & \Leftrightarrow \\ f(x \mid C_1)\frac{P(C_1)}{f(x)} & = & f(x \mid C_2)\frac{P(C_2)}{f(x)} & \Leftrightarrow \\ f(x \mid C_1) & = & f(x \mid C_2) & \Leftrightarrow \\ \mathcal{N}(\mu_1, \sigma^2) & = & \mathcal{N}(\mu_2, \sigma^2) & \Leftrightarrow \\ (x-\mu_1)^2 & = & (x-\mu_2)^2 & \end{array}$$

Thus, the decision boundary is the point equidistant to μ_1 and μ_2: $b = \frac{\mu_1+\mu_2}{2}$.

The corresponding decision rule is, given a data point $x \in \mathbb{R}$:

- if $x < \frac{\mu_1+\mu_2}{2}$, then classify x in class 1;
- otherwise, classify x in class 2.

Note that this is the centroid method.

2. Applying basic rules of probability, we have

$$\begin{aligned} P_e &= P((C_1 \text{ misclassified as } C_2) \cup (C_2 \text{ misclassified as } C_1)) \\ &= P(C_1 \text{ misclassified as } C_2) + P(C_2 \text{ misclassified as } C_1) \\ &= P(\text{misclassified as } C_2 \mid C_1)P(C_1) + P(\text{misclassified as } C_1 \mid C_2)P(C_2). \end{aligned}$$

We know $P(C_1) = P(C_2) = \frac{1}{2}$. Thus, we use the PDF to calculate

$$P(\text{misclassified as } C_1 \mid C_2) = \int_{-\infty}^{b} \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu_2)^2/(2\sigma^2)} dx.$$

$$P(\text{misclassified as } C_2 \mid C_1) = \int_{b}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu_1)^2/(2\sigma^2)} dx.$$

Therefore,

$$\begin{aligned} P_e(b) &= \frac{1}{2}\left(P(\text{misclassified as } C_1 \mid C_2) + P(\text{misclassified as } C_1 \mid C_2)\right) \\ &= \frac{1}{2\sqrt{2\pi}\sigma}\left(\int_{-\infty}^{b} e^{-\frac{(x-\mu_2)^2}{2\sigma^2}} dx + \int_{b}^{\infty} e^{-\frac{(x-\mu_1)^2}{2\sigma^2}} dx\right). \end{aligned}$$

3. We know the optimal decision boundary must lie between μ_1 and μ_2 (convince yourself this is the case!). Thus, we can minimize $P_e(b)$ by taking the derivative and setting it equal to zero. Letting $C = \frac{1}{2\sqrt{2\pi}\sigma}$, $f_1(x) = e^{-(x-\mu_2)^2/(2\sigma^2)}$, and $f_2(x) = e^{-(x-\mu_1)^2/(2\sigma^2)}$, we have

$$\begin{aligned} \frac{dP_e(b)}{db} &= \frac{d}{db} C\left[\int_{-\infty}^{b} f_1(x)dx + \int_{b}^{\infty} f_2(x)dx\right] \\ &= C\left[\frac{d}{db}\left(\int_{-\infty}^{b} f_1(x)dx\right) + \frac{d}{db}\left(\int_{b}^{\infty} f_2(x)dx\right)\right]. \end{aligned}$$

From the fundamental theorem of calculus, we know that, for continuous functions $f(x)$ such that $|f| \to 0$ for $x \to -\infty$,

$$\frac{d}{dx}\int_{-\infty}^{x} f(t)\,dt = f(x).$$

Noting that f_1 and f_2 satisfy these properties, we apply that here to get

$$\frac{dP_e(b)}{db} = C\left[f_1(b) - f_2(b)\right].$$

Setting $\frac{dP_e(b^*)}{db}$ to zero yields

$$\begin{aligned} f_1(b^*) &= f_2(b^*) \implies \\ \log(f_1(b^*)) &= \log(f_2(b^*)) \implies \\ \frac{(b^*-\mu_2)^2}{2\sigma^2} &= \frac{(b^*-\mu_1)^2}{2\sigma^2} \implies \\ (b^*-\mu_2)^2 &= (b^*-\mu_1)^2. \end{aligned}$$

This is the same constraint we saw in part (1) and thus the optimal decision boundary is equivalent. That is,

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

3 Isocontours of Normal Distributions

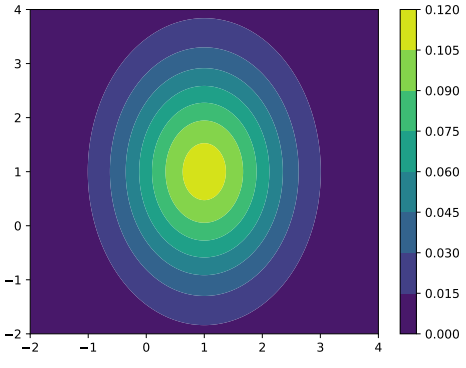
Let $f(\mu, \Sigma)$ be the probability density function of a normally distributed random variable in $\mathbb{R}^2$. Write code to plot the isocontours of the following functions, each on its own separate figure. Make sure it is clear which figure belongs to which part. You're free to use any plotting libraries or stats utilities available in your programming language; for instance, in Python you can use Matplotlib and SciPy.

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 $\Sigma_1 = \Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$.
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 $\Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.

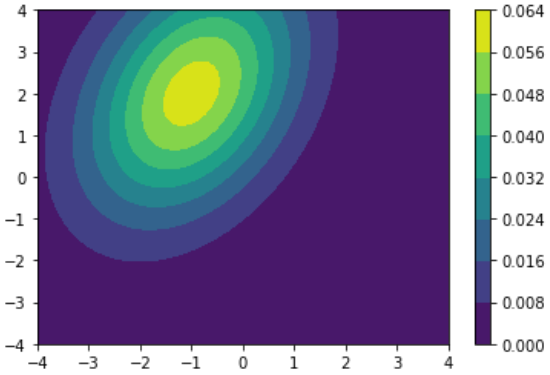
Solution:

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

def plot_contours():
    fig = plt.figure(figsize=(10,10))
    ax0 = fig.add_subplot(111)
    ax0.contour(rv.pdf(pos).reshape(500,500))
    plt.show()

# Part a
```

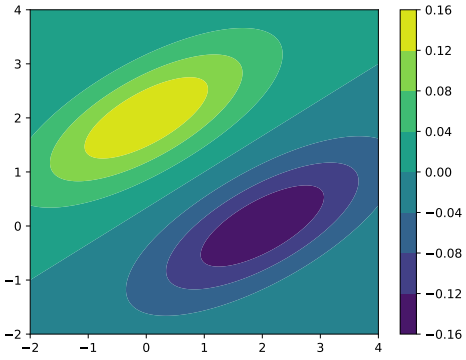
```
# Generate grid of points at which to evaluate pdf
x = np.linspace(-2, 4, 500)
y = np.linspace(-2, 4, 500)
X,Y = np.meshgrid(x, y)
pos = np.array([Y, X]).T
rv = scipy.stats.multivariate_normal([1, 1], [[1, 0], [0, 2]])
Z = rv.pdf(pos)

plt.contourf(X, Y, Z)
plt.colorbar()
plt.show()

# Part b
x = np.linspace(-4, 4, 500)
y = np.linspace(-4, 4, 500)
X,Y = np.meshgrid(x, y)
pos = np.array([Y, X]).T
rv = scipy.stats.multivariate_normal([-1, 2], [[2, 1], [1, 4]])
Z = rv.pdf(pos)

plt.contourf(X, Y, Z)
plt.colorbar()
plt.show()

# Part c
x = np.linspace(-2, 4, 500)
y = np.linspace(-2, 4, 500)
X,Y = np.meshgrid(x, y)
pos = np.array([Y, X]).T
rv1 = scipy.stats.multivariate_normal([0, 2], [[2, 1], [1, 1]])
rv2 = scipy.stats.multivariate_normal([2, 0], [[2, 1], [1, 1]])
Z = rv1.pdf(pos) - rv2.pdf(pos)

plt.contourf(X, Y, Z)
plt.colorbar()
plt.show()

# Part d
x = np.linspace(-2, 4, 500)
y = np.linspace(-2, 4, 500)
X,Y = np.meshgrid(x, y)
pos = np.array([Y, X]).T
rv1 = scipy.stats.multivariate_normal([0, 2], [[2, 1], [1, 1]])
rv2 = scipy.stats.multivariate_normal([2, 0], [[2, 1], [1, 4]])
Z = rv1.pdf(pos) - rv2.pdf(pos)

plt.contourf(X, Y, Z)
plt.colorbar()
plt.show()

# Part e
x = np.linspace(-3, 3, 500)
y = np.linspace(-3, 3, 500)
X,Y = np.meshgrid(x, y)
pos = np.array([Y, X]).T
rv1 = scipy.stats.multivariate_normal([1, 1], [[2, 0], [0, 1]])
rv2 = scipy.stats.multivariate_normal([-1, -1], [[2, 1], [1, 2]])
Z = rv1.pdf(pos) - rv2.pdf(pos)

plt.contourf(X, Y, Z)
plt.colorbar()
plt.show()
```



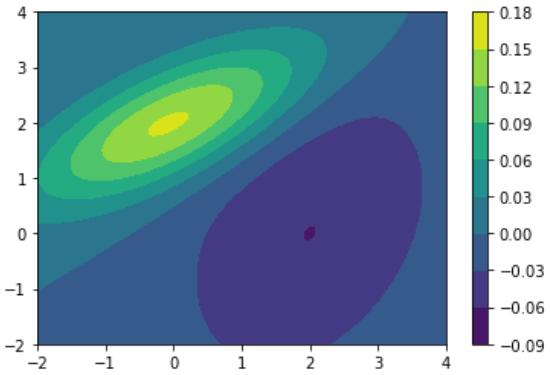
(a)



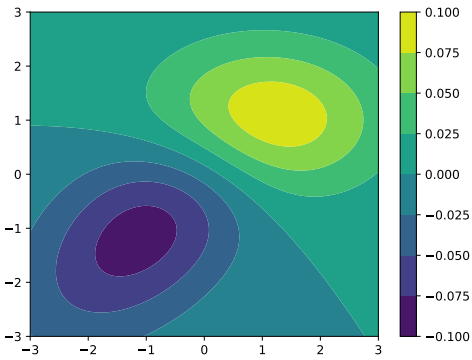
(b)



(c)



(d)



(e)

4 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. Write a program that draws $n = 100$ random two-dimensional sample points from (X_1, X_2) such that the ith value sampled from X_2 is calculated based on the ith value sampled from X_1. 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.

(a) Compute the mean (in $\mathbb{R}^2$) of the sample.

(b) Compute the 2×2 covariance matrix of the sample.

(c) Compute the eigenvectors and eigenvalues of this covariance matrix.

(d) 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.

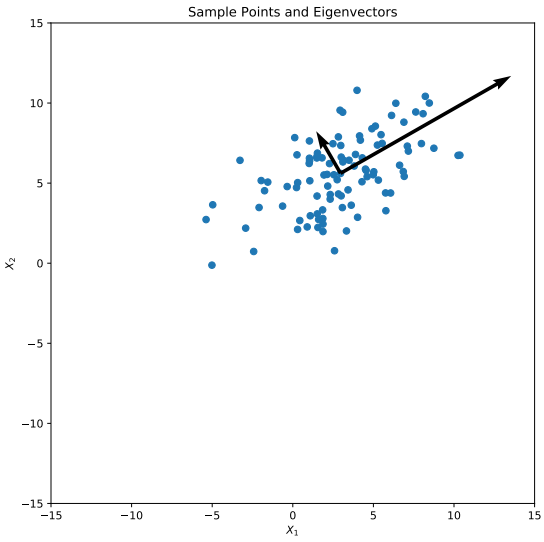
(e) Let $U = [v_1 \quad v_2]$ be a 2×2 matrix whose columns are the eigenvectors of the covariance matrix, where v_1 is the eigenvector with the larger eigenvalue. We use $U^\top$ as a rotation matrix to rotate each sample point from the (X_1, X_2) coordinate system to a coordinate system aligned with 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 $[-15, 15]$.

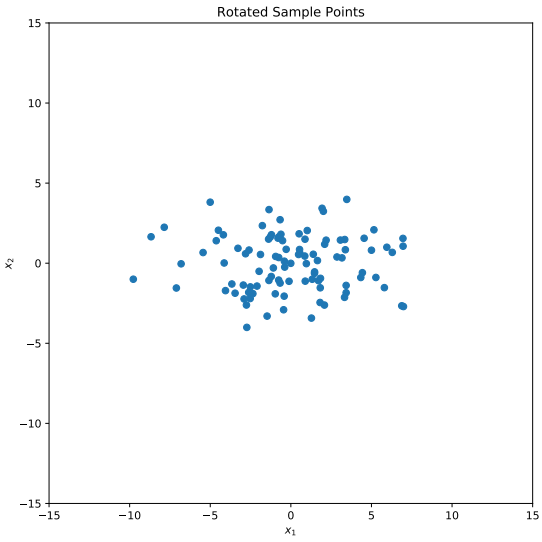
In your plots, **clearly label the axes and include a title**. Moreover, **make sure the horizontal and vertical axis have the same scale!** The aspect ratio should be one.

Solution:

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

np.random.seed(9)

X = np.random.normal(loc=3, scale=3, size=100)
Y = np.random.normal(loc=4, scale=2, size=100)
sample = np.array([np.array((x, 0.5 * x + y)) for (x, y) in zip(X, Y)])

# Part a (compute the sample mean)
sample_mean = np.mean(sample, axis=0)
print('Sample Mean = {0}'.format(sample_mean))

#Sample Mean = [2.96143749 5.61268062]
```

```
# Part b (compute the sample covariance matrix)
sample_cov = np.cov(sample.T)
print('Sample Covariance')
print(sample_cov)

#Sample Covariance
#[[9.93191037 3.96365428]
# [3.96365428 5.30782634]]

# Part c (compute the eigenvalues and eigenvectors)
eigen_values, eigen_vectors = np.linalg.eig(sample_cov)
print('Eigenvalues = {0}'.format(eigen_values))
print('Eigenvectors (columns)')
print(eigen_vectors)

#Eigenvalues = [12.20856027  3.03117644]
#Eigenvectors (columns)
# [[ 0.86713795 -0.49806804]
#  [ 0.49806804  0.86713795]]


# Part d (plot data and eigenvectors scaled by eigenvalues)
plt.figure(figsize=(8, 8))
plt.scatter(sample[:, 0], sample[:, 1])
plt.xlim(-15, 15)
plt.ylim(-15, 15)
plt.xlabel(r"$X_1$")
plt.ylabel(r"$X_2$")
plt.title("Sample Points and Eigenvectors")
vec_X = [sample_mean[0], sample_mean[0]]
vec_Y = [sample_mean[1], sample_mean[1]]
vec_U = [eigen_vectors[0][0] * eigen_values[0], eigen_vectors[0][1] * eigen_values[1]]
vec_V = [eigen_vectors[1][0] * eigen_values[0], eigen_vectors[1][1] * eigen_values[1]]
plt.quiver(vec_X, vec_Y, vec_U, vec_V, angles="xy", scale_units="xy", scale=1)
plt.show()

# Part e (plot rotated data in coorinate system defined by eigenvectors)
rotated = np.dot(eigen_vectors.T, (sample - sample_mean).T).T
plt.figure(figsize=(8, 8))
plt.scatter(rotated[:, 0], rotated[:, 1])
plt.xlim(-15, 15)
plt.ylim(-15, 15)
plt.xlabel(r"$x_1$")
plt.ylabel(r"$x_2$")
plt.title("Rotated Sample Points")
plt.show()
```


Sample Points and Eigenvectors
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15
X_1
X_2


Rotated Sample Points
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15
x_1
x_2

5 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 and $\lambda_s \geq 0$ is the loss incurred for making a misclassification. Hence the risk of classifying a new data point x as class $i \in \{1, 2, \ldots, c+1\}$ is

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

1. Show that the following policy obtains the minimum risk when $\lambda_r \leq \lambda_s$.
   (a) Choose class i if $P(Y = i|x) \geq P(Y = j|x)$ for all j and $P(Y = i|x) \geq 1 - \lambda_r/\lambda_s$;
   (b) 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.

Solution:

1. Let $r : \mathbb{R}^d \to \{1, \ldots, c+1\}$ be the decision rule which implements the described policy. We will prove that in expectation the rule r is at least as good as the arbitrary rule f. Let $x \in \mathbb{R}^d$ be a data point, which we want to classify.
   - Assume that case (1) holds and so $r(x) = i$, $P(Y = i|x) \geq P(Y = j|x)$ for all j, and $P(Y = i|x) \geq 1 - \lambda_r/\lambda_s$. Then:

$$\begin{aligned} R(r(x) = i|x) &= \sum_{j=1}^{c} \ell(r(x) = i, y = j) P(Y = j|x) \\ &= \lambda_s \sum_{j=1, j\neq i} P(Y = j|x) \\ &= \lambda_s \left(1 - P(Y = i|x)\right). \end{aligned}$$

     We are going to consider two cases for $f(x)$:
     – Let $f(x) = k$, with $k \neq i, c+1$. Then:

$$R(f(x) = k|x) = \lambda_s \left(1 - P(Y = k|x)\right)$$

   Hence $R(r(x) = i|x) \leq R(f(x) = k|x)$, since $P(Y = i|x) \geq P(Y = k|x)$.

– Let $f(x) = c + 1$. Then:

$$R(f(x) = k|x) = \lambda_r$$

Hence $R(r(x) = i|x) \leq R(f(x) = k|x)$, since $P(Y = i|x) \geq 1 - \lambda_r/\lambda_s$.

- Assume that case (2) holds and so $r(x) = c + 1$. Then:

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

Let $f(x) = k$, with $k \neq c + 1$. Then:

$$R(f(x) = k|x) = \lambda_s (1 - P(Y = k|x))$$

case (1) does **not** hold which means that:

$$\max_{j \in \{1,\ldots,c\}} P(Y = j|x) < 1 - \lambda_r/\lambda_s$$

hence $P(Y = k|x) < 1 - \lambda_r/\lambda_s$, and so $R(r(x) = i|x) < R(f(x) = k|x)$.

Therefore in every case we proved that the rule r is at list as good as the arbitrary rule f, which proves that r is an optimal rule.

2. If $\lambda_r = 0$, then case (1) will hold iff there exists an $i \in \{1, \ldots, c\}$ such that $P(r(x) = i|x) = 1$. So we will either classify x in class i if we are 100% sure about this, or else we will choose doubt. Of course this is completely consistent with our intuition, because choosing doubt does not have any penalty at all, since $\lambda_r = 0$.

   If $\lambda_r > \lambda_s$, then we will always classify x in the class $i \in \{1, \ldots, c\}$ which gives the highest probability of correct classification. Once again this makes sense, since the cost of choosing doubt is higher than classifying x in any of the classes, hence our best option is to classify x in the class which gives the highest probability for a correct classification.

6 Maximum Likelihood Estimation and Bias

Let $X_1, \ldots, X_n \in \mathbb{R}$ be n sample points drawn independently from 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.)

(a) Derive the maximum likelihood estimates, denoted $\hat{\mu}$ and $\hat{\sigma}$, for the mean μ and the parameter σ. 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.

Solution:

$$\mathcal{L}(\mu, \sigma; X) = \prod_{i=1}^{n} \frac{\sqrt{i}}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(X_i - \mu)^2 i}{2\sigma^2}\right).$$

$$\ell(\mu, \sigma; X) = \frac{1}{2} \sum_{i=1}^{n} \ln \frac{i}{2\pi} - n \ln \sigma - \sum_{i=1}^{n} \frac{(X_i - \mu)^2 i}{2\sigma^2}.$$

$$\frac{\partial \ell}{\partial \mu} = \sum_{i=1}^{n} \frac{(X_i - \mu)i}{\sigma^2} = 0 \implies$$
$$\hat{\mu} = \frac{\sum_{i=1}^{n} X_i i}{\sum_{i=1}^{n} i}.$$
$$\frac{\partial \ell}{\partial \sigma} = -\frac{n}{\sigma} + \sum_{i=1}^{n} \frac{(X_i - \mu)^2 i}{\sigma^3} = 0 \implies$$
$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (X_i - \mu)^2 i.$$

But we don't know the value of μ, so our MLE estimate for σ is $\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (X_i - \hat{\mu})^2 i$.

(b) 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,

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

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.*

Solution: The MLE sample estimator μ is unbiased.

$$\begin{aligned} bias(\hat{\mu}) &= \mathbb{E}[\hat{\mu}] - \mu \\ &= \mathbb{E}\left[\frac{\sum_{i=1}^{n} X_i i}{\sum_{i=1}^{n} i}\right] - \mu \\ &= \frac{\sum_{i=1}^{n} \mathbb{E}[X_i] i}{\sum_{i=1}^{n} i} - \mu \\ &= \frac{\sum_{i=1}^{n} \mu i}{\sum_{i=1}^{n} i} - \mu \\ &= \mu \frac{\sum_{i=1}^{n} i}{\sum_{i=1}^{n} i} - \mu \\ &= \mu - \mu \\ &= 0. \end{aligned}$$

1. 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.*

   Solution: The MLE sample estimator for σ^2 is biased. We begin with some helpful identities. Let X be any random variable. Then we have

   $$Var[X] = \mathbb{E}[X^2] + \mathbb{E}[X]^2$$

Now let $z = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$. Because $\hat{\mu} = \frac{1}{z}\sum X_i i$, we have $Var[\hat{\mu}] = \frac{\sigma^2}{z}$. Thus, we get

$$\mathbb{E}[\hat{\mu}^2] = \frac{\sigma^2}{z} + \mu^2$$

We now plug into the definition of bias and simplify

$$\begin{aligned}
bias(\hat{\sigma^2}) &= \mathbb{E}[\hat{\sigma^2}] - \sigma^2 \\
&= \mathbb{E}[\frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{\mu})^2 i] - \sigma^2 \\
&= \frac{1}{n}\mathbb{E}[\sum_{i=1}^{n} X_i^2 i + \sum_{i=1}^{n}\hat{\mu}^2 i - 2\hat{\mu}\sum_{i=1}^{n} X_i i] - \sigma^2 \\
&= \frac{1}{n}\mathbb{E}[\sum_{i=1}^{n} X_i^2 i + z\hat{\mu}^2 - 2z\hat{\mu}^2] - \sigma^2 \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}\mathbb{E}[X_i^2]i - z\mathbb{E}[\hat{\mu}^2]\right] - \sigma^2 \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}(\frac{\sigma^2}{i} + \mu^2)i - z(\frac{\sigma^2}{z} + \mu^2)\right] - \sigma^2 \\
&= \frac{1}{n}\left[n\sigma^2 + z\mu^2 - \sigma^2 - z\mu^2\right] - \sigma^2 \\
&= \frac{n-1}{n}\sigma^2 - \sigma^2 \\
&= -\frac{1}{n}\sigma^2
\end{aligned}$$

We therefore see that the MLE sample estimator for variance systematically *underestimates* the true variance. It turns out that we can correct this error by sightly increasing our estimate by dividing by $n - 1$ instead of n. Thus, the new sample estimator

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

is unbiased. This can be shown by repeating the above algebra with the new denominator and observing that the bias does in fact equal zero.

7 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 μ 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 $X_1, X_2, \ldots, X_n$ 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 covariance matrix of the sample.

(a) 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? Make sure your answer is complete; i.e., it includes all cases in which the covariance matrix of the sample is singular. Express your answer in terms of the geometric arrangement of the sample points X_i.

(b) 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.

(c) 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.

Solution:

(a) $\tilde{\Sigma}$ is not invertible if and only if all the points lie on a common hyperplane in feature space(i.e. they don't span all of $\mathbb{R}^{dxd}$)

(b) Note: This question is worded to allow lots of room for creativity, including answers we might not have thought of. So when grading, don't demand a perfectly precise answer; we're more interested in an intuitive understanding of what the modification means.

We can create a new matrix $\tilde{\Sigma} = \hat{\Sigma} + \alpha I$. We want to make all the eigenvalues of the new covariance matrix positive so it will be invertible. Notice that the original $\hat{\Sigma}$ had all nonnegative eigenvalues. We ideally want α to be a very small number so not to bias our covariance matrix. Since the eigenvalues of $\tilde{\Sigma}^{-1}$ are the reciprocals of the originals. Then we want α to be large enough so that the eigenvalues of $\tilde{\Sigma}^{-1}$ don't blow up. The best way of selecting α is hyperparameter tuning.

Another acceptable answer is projecting the covariance matrix onto a lower dimensional subspace by removing the features that are linear combinations of the remaining features.

We continue this process until all the features are independent. This process will leave a full rank matrix that will be invertible.

(c) x will maximize the PDF $f(x)$ if it is the eigenvector of Σ with the largest eigenvalue. Similarly, x will minimize the PDF $f(x)$ if it is the eigenvector of Σ with the smallest eigenvalue. The distance metric is Σ^{-1}, which means it penalizes deviation from the mean least along the direction of its eigenvector with the smallest eigenvalue(which is paired with the largest eigenvalue of Σ) and vice versa.

8 Gaussian Classifiers for Digits and Spam

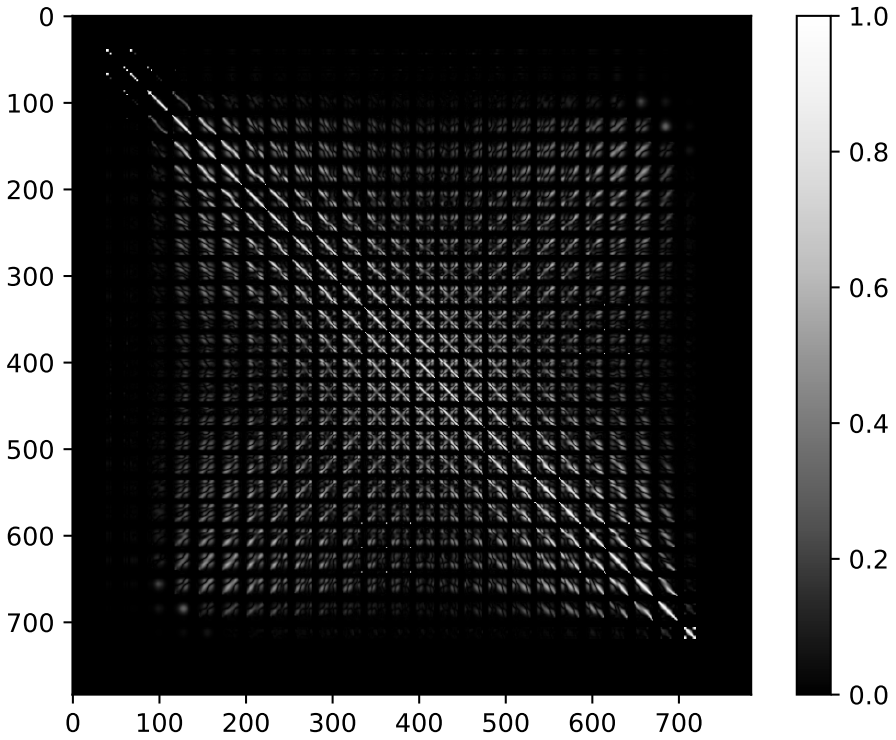
In this problem, you will build classifiers based on Gaussian discriminant analysis. Unlike Homework 1, you are NOT allowed to use any libraries for out-of-the-box classification (e.g. `sklearn`). You may use anything in `numpy` and `scipy`.

The training and test data can be found with this homework. Don't use the training/test data from Homework 1, as they have changed for this homework. You can verify that your data files and python environment are setup properly by running `sanity.py`. 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.

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.

   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). How do the diagonal terms compare with the off-diagonal terms? What do you conclude from this?



Solution:

This is technically a visualization of the Pearson product-moment correlation coefficients, a normalization of the covariance matrix such that $R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii}C_{jj}}}$, where C is the covariance matrix. Every element of R is in $[-1, 1]$, and so it is easier to understand the correlation between different pixels when visualizing this matrix. Furthermore we've visualized the absolute value of the entries in R.

Things to notice include that the diagonal elements are generally larger than the off-diagonal elements, and that the entries further away from the diagonal are generally lower than entries closer to the diagonal. We interpret this to mean that the correlation between neighboring pixels is generally higher than pixels that are further apart.

3. Classify the digits in the test set on the basis of posterior probabilities with two different approaches. Feel free to either use the starter code provided in `starter.py` or write your own implementation from scratch

   (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 covariance matrix Σ, which you compute by averaging the 10 covariance matrices from the 10 classes.

   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 invertable. Instead, you should find a

way to solve the implied linear system without computing the inverse. *Hint: How do we solve OLS when the data matrix is singular?*

Hold out 10,000 randomly chosen training points for a validation set (You may re-use 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 some variation 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 k-fold cross 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?

Solution: In our experiments on MNIST, both LDA and QDA had similar error rates on the validation data for training sets of size greater than 10000. However we generally expect QDA to perform better because it is more expressive. The drawback of QDA is that the larger number of free variables leaves it vulernable to possible overfitting.

(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? Include written answers where indicated.

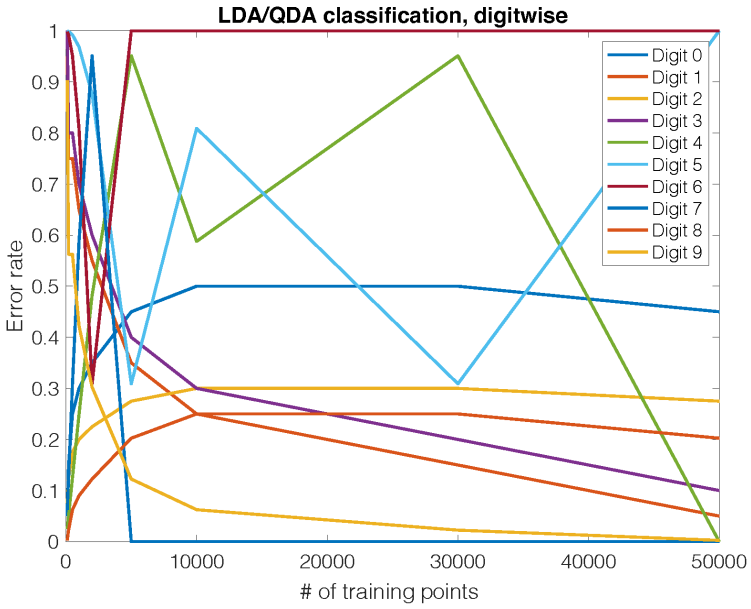


Figure 1: Sample graph with 10 plots

Solution: In our experiments on MNIST, digit 1 was easiest for LDA and digit 0 was easiest for QDA, as measured by validation error. The plots for validation error and per-digit validation error for both LDA and QDA are available below.

4. Using the `mnist_data.mat`, 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 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 transer 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. Next, apply LDA or QDA (your choice) to spam. 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.

   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 word-2-vec!).

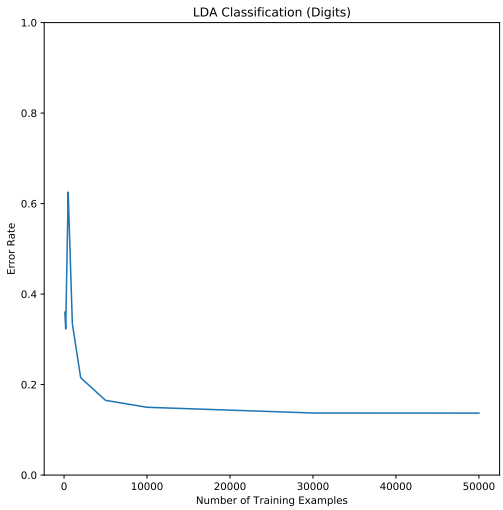
Solution:

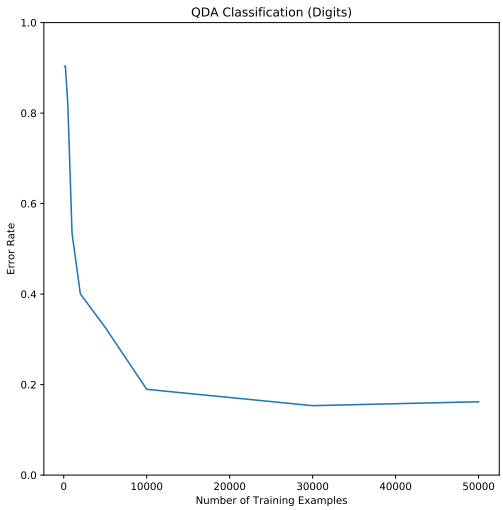
```
import numpy as np
import scipy.cluster
import scipy.io
import scipy.ndimage
import matplotlib
import matplotlib.pyplot as plt

import learners

def train_val_split(data, labels, val_size):
    num_items = len(data)
    assert num_items == len(labels)
    assert val_size >= 0
    if val_size < 1.0:
        val_size = int(num_items * val_size)
    train_size = num_items - val_size
    idx = np.random.permutation(num_items)
    data_train = data[idx][:train_size]
    label_train = labels[idx][:train_size]
    data_val = data[idx][train_size:]
    label_val = labels[idx][train_size:]
    return data_train, data_val, label_train, label_val

mnist = scipy.io.loadmat("./mnist-data/mnist_data.mat")

train_data, train_labels = mnist['training_data'], mnist['training_labels']
train_data_normalized = scipy.cluster.vq.whiten(train_data)
labels = np.unique(train_labels)

# Part a
mnist_fitted = {}
for label in labels:
    class_indices = (train_labels == label).flatten()
    data = train_data_normalized[class_indices]
    mean = np.mean(data, axis=0)
    cov = np.cov(data, rowvar=False)
    mnist_fitted[label] = (mean, cov)

# Part b
class_indices = (train_labels == labels[0]).flatten()
data = train_data_normalized[class_indices]
ncov = np.corrcoef(data, rowvar=False)
ncov[np.isnan(ncov)] = 0
ncov = np.abs(ncov)
```

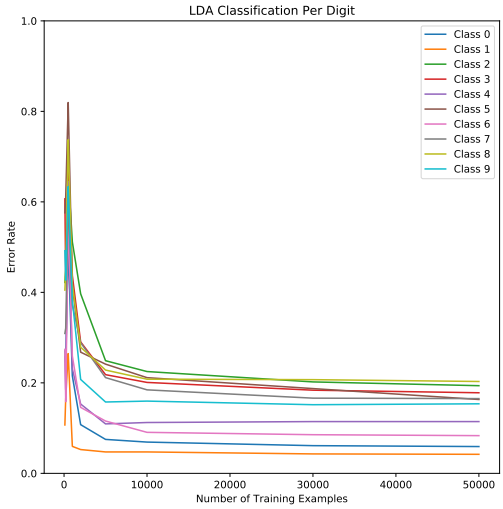
```
plt.imshow(ncov, cmap=matplotlib.cm.Greys_r)
plt.colorbar()
plt.show()


# Part c (Gaussian/linear discriminant analysis)
train_data, val_data, train_labels, val_labels = train_val_split(train_data_normalized, train_labels,
    ↪ val_size=10000)
val_labels.flatten()
num_training = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
lda_errors = []
qda_errors = []

for num in num_training:
    gda = learners.GDA()
    gda.fit(train_data[:num], train_labels[:num])
    err = 1 - gda.evaluate(val_data, val_labels)
    lda_errors.append(err)
    err = 1 - gda.evaluate(val_data, val_labels, mode="qda")
    qda_errors.append(err)

# Part c.1
plt.figure(figsize=(8, 8))
plt.plot(num_training, lda_errors)
plt.xlabel("Number of Training Examples")
plt.ylabel("Error Rate")
plt.title("LDA Classification (Digits)")
plt.ylim((0, 1))
plt.show()
print(lda_errors)

#[0.34, 0.346, 0.62, 0.36, 0.21, 0.16, 0.14, 0.13, 0.13]

# Part c.2

plt.figure(figsize=(8, 8))
plt.plot(num_training, qda_errors)
plt.xlabel("Number of Training Examples")
plt.ylabel("Error Rate")
plt.title("QDA Classification (Digits)")
plt.ylim((0, 1))
plt.show()
print(qda_errors)

#[0.90, 0.90, 0.78, 0.51, 0.40, 0.26, 0.17, 0.15, 0.15]
```

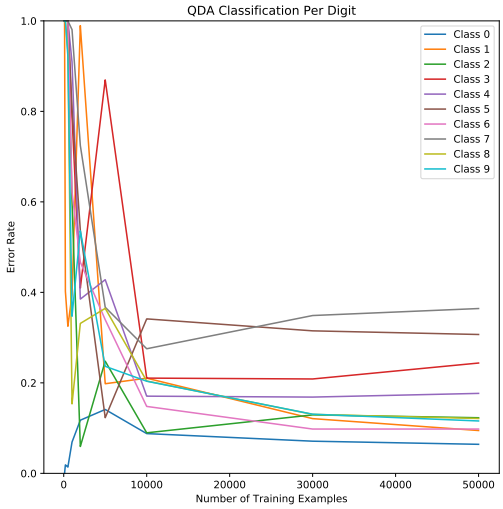
```
## learners.py
## Author: Sinho Chewi

import math
import numpy as np
import scipy.stats

class GDA:
    """Perform Gaussian discriminant analysis (both LDA and QDA)."""

    def evaluate(self, X, y, mode="lda"):
        """Predict and evaluate the accuracy using zero-one loss.

        Args:
            X (np.ndarray): The feature matrix.
            y (np.ndarray): The true labels.

        Optional:
            mode (str): Either "lda" or "qda".

        Returns:
            float: The zero-one loss of the learner.
```

```
        Raises:
            RuntimeError: If an unknown mode is passed into the method.
        """
        pred = self.predict(X, mode=mode)
        return np.sum(pred == y) / y.shape[0]

    def fit(self, X, y):
        """Train the GDA model (both LDA and QDA).

        Args:
            X (np.ndarray): The feature matrix.
            y (np.ndarray): The true labels.
        """
        self.classes = np.unique(y)
        num_examples = X.shape[0]
        num_features = X.shape[1]
        self.fitted_params = []
        self.pooled_cov = np.zeros((num_features, num_features))
        for c in self.classes:
            class_indices = (y == c).flatten()
            data = X[class_indices]
            mean = np.mean(data, axis=0)
            unscaled_cov = np.dot((data - mean).T, data - mean)
            self.pooled_cov = np.add(self.pooled_cov, unscaled_cov)
            scaled_cov = unscaled_cov / data.shape[0]
            prior = data.shape[0] / num_examples
            self.fitted_params.append((c, mean, scaled_cov, prior))
        self.pooled_cov = self.pooled_cov / num_examples

    def predict(self, X, mode="lda"):
        """Use the fitted model to make predictions.

        Args:
            X (np.ndarray): The feature matrix.

        Optional:
            mode (str): Either "lda" or "qda".

        Returns:
            np.ndarray: The array of predictions.

        Raises:
            RuntimeError: If an unknown mode is passed into the method.
        """
        if mode == "lda":
            pred = np.array([
                (scipy.stats.multivariate_normal.logpdf(
                    X, allow_singular=True, cov=self.pooled_cov, mean=mean)
                    + math.log(prior))
                for (c, mean, cov, prior) in self.fitted_params
            ])
            return self.classes[np.argmax(pred, axis=0)].reshape((-1, 1))
        elif mode == "qda":
            pred = np.array([
                (scipy.stats.multivariate_normal.logpdf(
                    X, allow_singular=True, cov=cov, mean=mean)
                    + math.log(prior))
                for (c, mean, cov, prior) in self.fitted_params
            ])
            return self.classes[np.argmax(pred, axis=0)].reshape((-1, 1))
        else:
            raise RuntimeError("Unknown mode!")
```


LDA Classification (Digits)
Error Rate
Number of Training Examples
0.0
0.2
0.4
0.6
0.8
1.0
0
10000
20000
30000
40000
50000


QDA Classification (Digits)
Error Rate
Number of Training Examples
0.0
0.2
0.4
0.6
0.8
1.0
0
10000
20000
30000
40000
50000


LDA Classification Per Digit
Error Rate
Number of Training Examples
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
0.0
0.2
0.4
0.6
0.8
1.0
0
10000
20000
30000
40000
50000


QDA Classification Per Digit
Error Rate
Number of Training Examples
Class 0
Class 1
Class 2
Class 3
Class 4
Class 5
Class 6
Class 7
Class 8
Class 9
0.0
0.2
0.4
0.6
0.8
1.0
0
10000
20000
30000
40000
50000