CS 189 Introduction to Machine Learning
Summer 2019 Marc Khoury & Brijen Thananjeyan HW3

Due: Monday, July 15 at 11:59 pm
Deliverables:

1. Submit your predictions for the test sets to Kaggle as early as possible. Include your Kaggle scores in your write-up (see below). The Kaggle competition for this assignment can be found at:
   - MNIST: `https://www.kaggle.com/t/403584fc1c7b47b480122f21ceb722c4`
   - SPAM: `https://www.kaggle.com/t/5c74291c27bd4f478dc82337e415a900`
2. Submit a PDF of your homework, **with an appendix listing all your code**, to the Gradescope assignment entitled "HW3 Write-Up". You may typeset your homework in LaTeX or Word (submit PDF format, **not** .doc/.docx format) or submit neatly handwritten and scanned solutions. **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.
   - In your write-up, please state with whom you worked on the homework.
   - In your write-up, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadverdently cheats.

     "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."
3. Submit all the code needed to reproduce your results to the Gradescope assignment entitled "HW3 Code". Yes, you must submit your code twice: once in your PDF write-up (above) so the readers can easily read it, and once in compilable/interpretable form so the readers can easily run it. Do **NOT** include any data files we provided. Please include a short file named README listing your name, student ID, and instructions on how to reproduce your results. Please take care that your code doesn't take up inordinate amounts of time or memory. If your code cannot be executed, your solution cannot be verified.
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 Gaussian Classification

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

(a) Find the Bayes optimal decision boundary and the corresponding Bayes decision rule.

(b) The Bayes error is the probability of misclassification,

$$P_e = P((\text{misclassified as } C_1) \mid C_2)\, P(C_2) + P((\text{misclassified as } C_2) \mid C_1)\, P(C_1).$$

Show that the Bayes error associated with this decision rule is

$$P_e = \frac{1}{\sqrt{2\pi}} \int_a^\infty e^{-z^2/2} dz$$

where $a = \dfrac{\mu_2 - \mu_1}{2\sigma}$.

Solution:

(a)

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

This yields the Bayes decision boundary: $x = \frac{\mu_1 + \mu_2}{2}$.

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.

(b)

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

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

Therefore:

$$P((\text{misclassified as } C_1) \mid C_2)P(C_2) + P((\text{misclassified as } C_2) \mid C_1)P(C_1) = P_e \cdot \frac{1}{2} + P_e \cdot \frac{1}{2} = P_e$$

2 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. You're free to use any plotting libraries available in your programming language; for instance, in Python you can use Matplotlib.

(a) $f(\mu, \Sigma)$, where $\mu = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$.

(b) $f(\mu, \Sigma)$, where $\mu = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ and $\Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$.

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

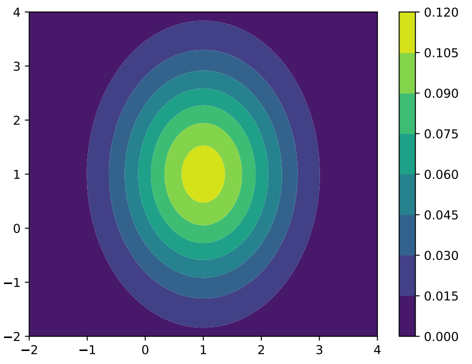
(d) $f(\mu_1, \Sigma_1) - f(\mu_2, \Sigma_2)$, where $\mu_1 = \begin{bmatrix} 0 \\ 2 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$ and $\Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}$.

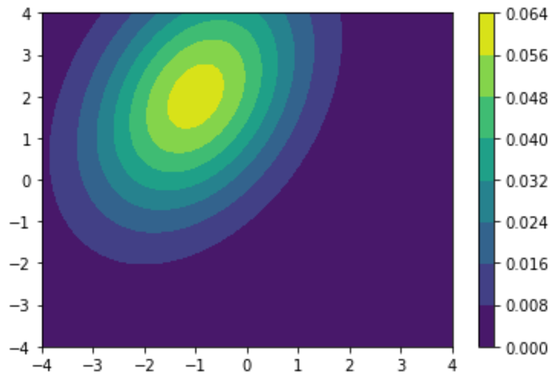
(e) $f(\mu_1, \Sigma_1) - f(\mu_2, \Sigma_2)$, where $\mu_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\mu_2 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ and $\Sigma_2 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$.

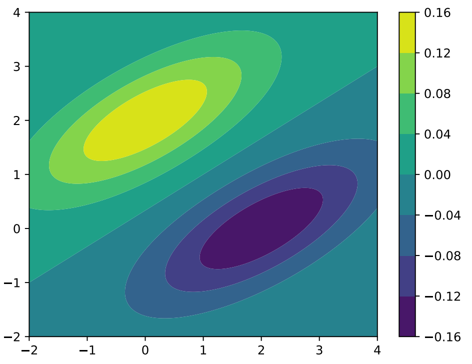
Solution:

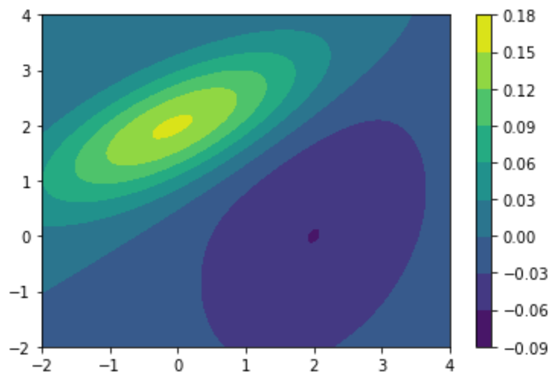
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats

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
```

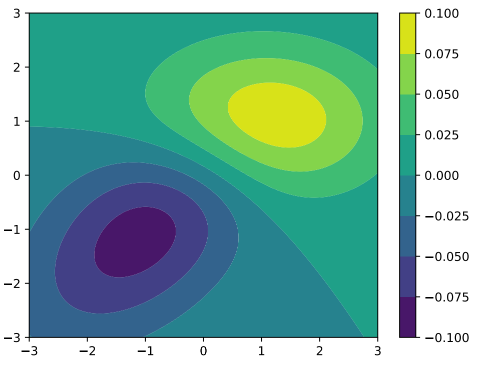
```
x = np.linspace(-2, 4, 500)
y = np.linspace(-2, 4, 500)
X,Y = np.meshgrid(x, y)
pos = np.array([Y, X]).T
rv = scipy.stats.multivariate_normal([1, 1], [[1, 0], [0, 2]])
Z = rv.pdf(pos)

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)

3 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 specify the Random Number Generator seed that was used so your simulation is reproducible. 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

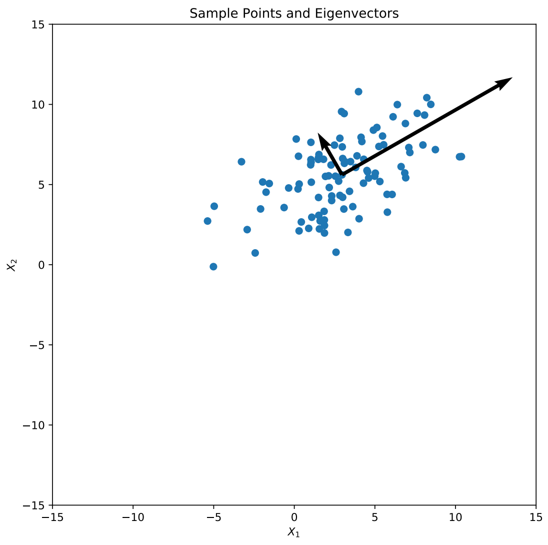
   (ii) arrows representing both covariance eigenvectors. The eigenvector arrows should originate at the mean and have magnitudes equal to their corresponding eigenvalues.

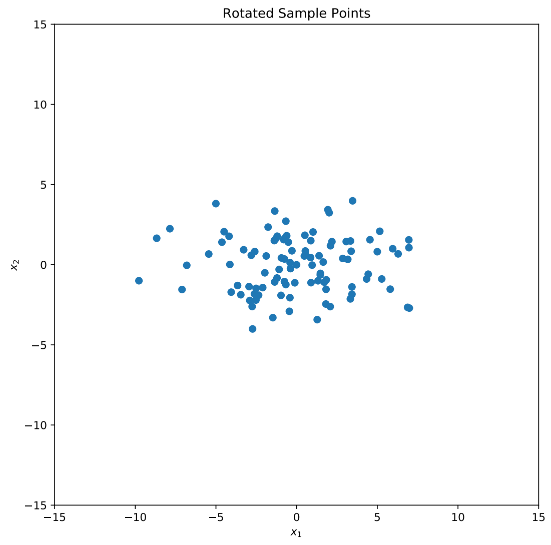
(e) Let $U = [v_1 \quad v_2]$ be a 2×2 matrix whose columns are the eigenvectors of the covariance matrix, where v_1 is the eigenvector with the larger eigenvalue. We use $U^\top$ as a rotation matrix to rotate each sample point from the (X_1, X_2) coordinate system to a coordinate system aligned with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

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()
```





4 Classification

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

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

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 $i \in \{1, 2, \dots, c+1\}$ is

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

(a) Show that the following policy obtains the minimum risk when $\lambda_r \leq \lambda_s$.

(1) Choose class i if $P(Y = i|x) \geq P(Y = j|x)$ for all j and $P(Y = i|x) \geq 1 - \lambda_r/\lambda_s$;

(2) Choose doubt otherwise.

(b) 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:

(a) Let $r : \mathbb{R}^d \to \{1, \dots, c+1\}$ 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:

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

  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.

(b) If $\lambda_r = 0$, then case (1) will hold iff there exists an $i \in \{1, \dots, c\}$ 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 $i \in \{1, \dots, c\}$ 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.

5 Maximum Likelihood Estimation

Let $X_1, \dots, X_n \in \mathbb{R}^d$ be n sample points drawn independently from a multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$.

(a) Suppose the normal distribution has an unknown diagonal covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_1^2 & & & & \\ & \sigma_2^2 & & & \\ & & \sigma_3^2 & & \\ & & & \ddots & \\ & & & & \sigma_d^2 \end{bmatrix}$$

and an unknown mean μ. Derive the maximum likelihood estimates, denoted $\hat{\mu}$ and $\hat{\sigma}_i$, for μ and σ_i. Show all your work.

Solution:

$$\begin{aligned}
\mathcal{L}(\mu,\Sigma;X) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi^d}\sqrt{\prod_{j=1}^{d}\sigma_j^2}} \exp\left(-\frac{1}{2}\sum_{j=1}^{d}\frac{(X_{ij}-\mu_j)^2}{\sigma_j^2}\right) \\
\ell(\mu,\Sigma;X) &= -\frac{nd}{2}\ln(2\pi) - n\sum_{j=1}^{d}\ln\sigma_j - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{d}\frac{(X_{ij}-\mu_j)^2}{\sigma_j^2} \\
\frac{\partial \ell}{\partial \mu_j} &= \sum_{i=1}^{n}\frac{X_{ij}-\mu_j}{\sigma_j^2} = 0 \to \hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i \\
\frac{\partial \ell}{\partial \sigma_j} &= -n\frac{1}{\sigma_j} + \sum_{i=1}^{n}\frac{(X_{ij}-\mu_j)^2}{\sigma_j^3} = 0 \\
\to \hat{\sigma}_j^2 &= \frac{1}{n}\sum_{i=1}^{n}(X_{ij}-\mu_j)^2
\end{aligned}$$

(b) Suppose the normal distribution has a known covariance matrix Σ and an unknown mean $A\mu$, where Σ and A are known $d \times d$ matrices, Σ is positive definite, and A is invertible. Derive the maximum likelihood estimate, denoted $\hat{\mu}$, for μ.

Solution:

$$\begin{aligned}
\mathcal{L}(\mu,\Sigma;X) &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi^d}\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(X_i - A\mu)^\top \Sigma^{-1}(X_i - A\mu)\right) \\
\ell(\mu,\Sigma;X) &= -\frac{nd}{2}\ln(2\pi) - \frac{n}{2}\ln|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(X_i - A\mu)^\top \Sigma^{-1}(X_i - A\mu).
\end{aligned}$$

Recall that $\nabla_x(c^\top x) = c$ and for a square matrix B, $\nabla_x(x^\top Bx) = B^\top x + Bx$, which is $2Bx$ if B is symmetric. As Σ^{-1} is symmetric,

$$\begin{aligned}
(X_i - A\mu)^\top \Sigma^{-1}(X_i - A\mu) &= X_i^\top \Sigma^{-1} X_i - 2(A\mu)^\top \Sigma^{-1} X_i + \mu^\top A^\top \Sigma^{-1} A\mu \\
\nabla_\mu\left((X_i - A\mu)^\top \Sigma^{-1}(X_i - A\mu)\right) &= -2\Sigma^{-1}A\mu + 2A^\top \Sigma^{-1} A\mu.
\end{aligned}$$

Therefore

$$\begin{aligned}
\nabla_\mu \ell &= A^\top \Sigma^{-1}\sum_{i=1}^{n}(X_i - A\mu) = 0 \\
\to \hat{\mu} &= A^{-1}\frac{\sum_{i=1}^{n} X_i}{n}
\end{aligned}$$

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

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

which is also known as the 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.

7 Gaussian Classifiers for Digits and Spam

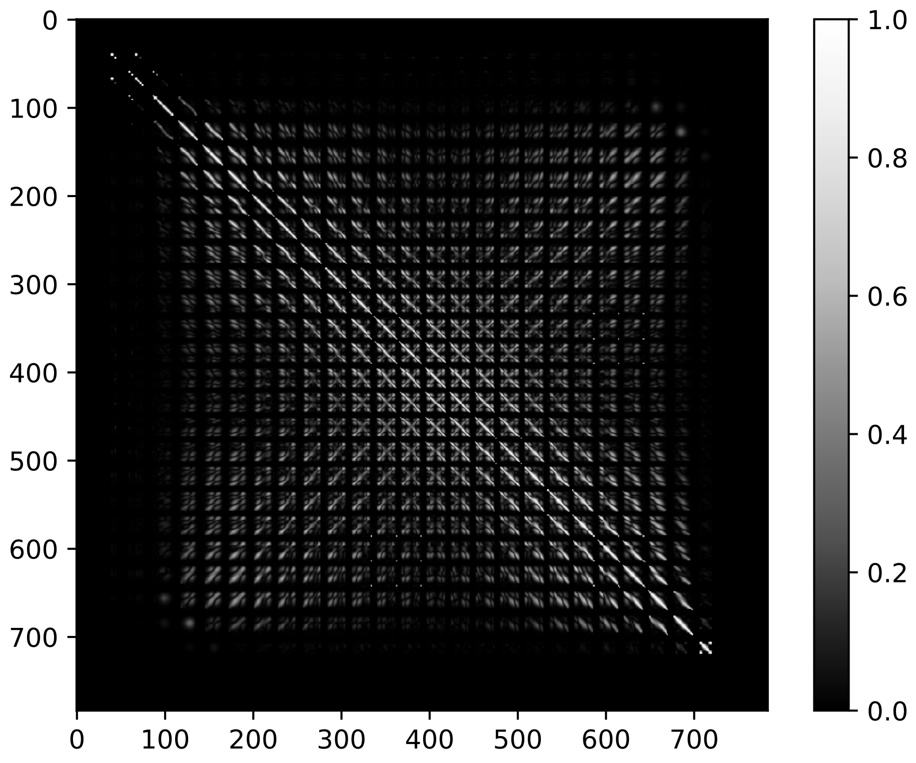
In this problem, you will build classifiers based on Gaussian discriminant analysis. Unlike Homework 1, you are NOT allowed to use any libraries for out-of-the-box classification (e.g. `sklearn`). You may use anything in `numpy` and `scipy`.

The training and test data can be found on in the post corresponding to this homework. Don't use the training/test data from Homework 1, as they have changed for this homework. 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.

(a) 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.

(b) (Written answer) 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.

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

(1) 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** try to compute the inverse of Σ (nor the determinant of Σ). 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. Classify each image in the validation set into one of the 10 classes (with a 0-1 loss function). Compute the error rate 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 variance in your error rate when few training points are used.)

(2) Quadratic discriminant analysis (QDA). Model the class conditionals 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 Q6.(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.

(3) (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.

(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. If you do so, please describe your implementation in your assignment. Please use extra features **only** for the Kaggle portion of the assignment.

In your submission, include plots of error rate versus number of training examples for both LDA and QDA. Similarly, include a plot of validation error for each digit across the training points. Create one graph with the plot for each digit. Which digit is easiest to classify? Include written answers where indicated.

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.

(d) 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. The TAs suggest using a bag-of-words model. You may use third-party packages to implement that if you wish. Also, normalizing your vectors might help.

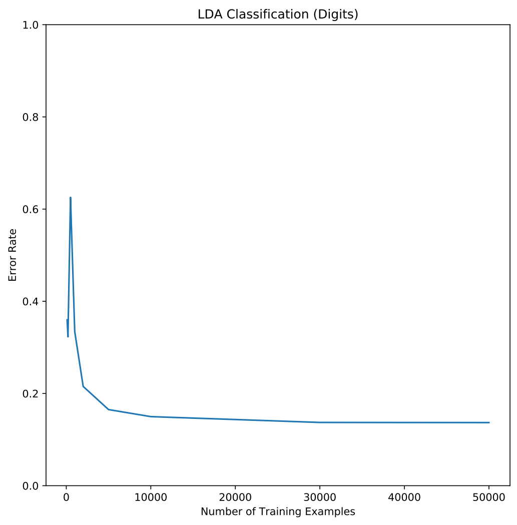
Solution:

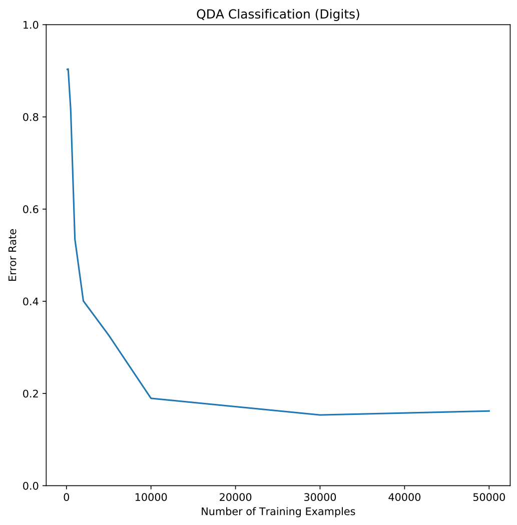
```
import numpy as np
import scipy.cluster
import scipy.io
import scipy.ndimage
import matplotlib
import matplotlib.pyplot as plt

import learners
```

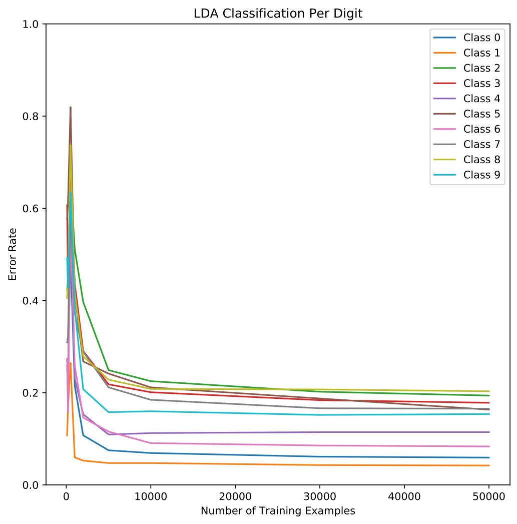
```
def train_val_split(data, labels, val_size):
    num_items = len(data)
    assert num_items == len(labels)
    assert val_size >= 0
    if val_size < 1.0:
        val_size = int(num_items * val_size)
    train_size = num_items - val_size
    idx = np.random.permutation(num_items)
    data_train =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

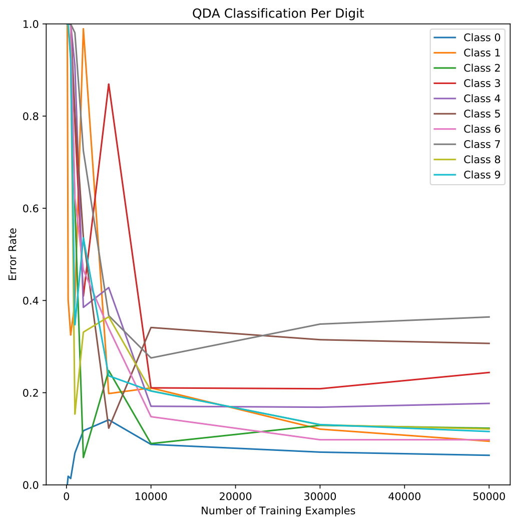
```python
# Part c.2

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]
```

```python
## 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 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


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