Students that I collaborated with: Students on ED, Various Students in office hours. Madeline (never got last name/ SID: 3036751976

Honor Code.

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.

X

2) 1)

$$\frac{P(X|Y=C_1)\,\cancel{P(Y=C_1)}}{\cancel{P(X)}} = \frac{P(X|Y=C_2)\,\cancel{P(Y=C_2)}}{\cancel{P(X)}}$$

$$N(\mu_i, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

$$\cancel{\frac{1}{\sigma\sqrt{2\pi}}} \cdot e^{\frac{-(x-\mu_1)^2}{2\sigma^2}} = \cancel{\frac{1}{\sigma\sqrt{2\pi}}} \cdot e^{\frac{-(x-\mu_2)^2}{2\sigma^2}}$$

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

$(x-\mu_1)(x-\mu_1)$

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

$$\cancel{x^2} - 2x\mu_1 + \mu_1^2 = \cancel{x^2} + 2x\mu_2 + \mu_2^{x}$$

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

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

$$\boxed{x = \frac{\mu_2^2 - \mu_1^2}{-2\mu_1 - 2\mu_2}}$$

2) The probability of a point C_1 being on the wrong side of the boundary means it needs to be to the right of b, so b to ∞. The PDF of this is as shown

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

For C_2 being misclassified, C_2 needs to be on the C_1 side, meaning $-\infty$ to b. So the PDF of that is

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

Putting those two together and factoring out an $\frac{1}{2\sqrt{2\pi}\sigma}$ gives us the equation

Shown

3) b^* same as answer in part 1

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

3) 1) With those conditions, the program will only choose i if it is the most probable choice. The other condition also shows that if the probability is less than a constant, which represents a threshold of the doubt penalty vs. the failure penalty

When $\lambda_r \leq \lambda_s$ then $1 - \lambda_r / \lambda_s$ will be greater than 0.5 so the program will choose doubt more often due to the probability needs to be higher.

2) If $\lambda_r = 0$, that means there is no risk in choosing c+1 or doubt, so the model will be very likely to choose it since it will encurr no loss

Likewise, if $\lambda_r > \lambda_s$ then doubting would be worse off that missclassifing, so the model will choose a classification in order to minimize risk.

4) 1)

$$\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x_i-\mu)^2_i}{2\sigma^2}}$$

ln
do
$\frac{d}{d\mu}$ $\frac{d}{d\sigma}$
set $=0$
and solve
for μ and
σ

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

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

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

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

$$\frac{(x_i-\mu)_i}{\sigma^2} = 0 \qquad \frac{x_i i}{\sigma^2} = \frac{\mu_i}{\sigma^2} \qquad \boxed{\mu = x_i}$$

$$\frac{\partial}{\partial \sigma} = \sum_{i=1}^{n} \frac{-1}{\sqrt{2\pi}}\sigma^2 + \cancel{\frac{-(x_i-x_i)^2_i}{\sigma^3}} = 0$$

$$\sum_{i=1}^{n} \frac{-1}{\sqrt{2\pi}}\sigma^2 = 0$$

$$\sigma^2 = 0$$

2)

$$\int f(x) \cdot p(x)\, dx$$

$\hat{\mu}$ in terms of x

PDF of x

$$\int x_i \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$$\frac{e^{\frac{\mu}{2\sigma}}}{\sqrt{2}\sqrt{\pi}\,\sigma} \int x\, e^{-\frac{x}{2\sigma}} \qquad u = -\frac{x}{2\sigma} \quad du = -\frac{1}{2\sigma}\, dx$$

$$4\sigma^2 \int u e^u\, du$$

$$u e^u - \int e^u\, du$$

$$-2\sigma x e^{-\frac{x}{2\sigma}} - 4\sigma^2 e^{-\frac{x}{2\sigma}}$$

$$= \frac{e^{\frac{\mu}{2\sigma}}}{\sqrt{2}\sqrt{\pi}\,\sigma} \int x e^{-\frac{x}{2\sigma}}\, dx$$

$$= -\frac{2^{3/2} x e^{\frac{\mu}{2\sigma} - \frac{x}{2\sigma}}}{\sqrt{\pi}} - \frac{2^{\frac{3}{2}} \sigma e^{\frac{\mu}{2\sigma} - \frac{x}{2\sigma}}}{\sqrt{\pi}} = 0$$

unbiased

3)

$$\int f(x) \cdot p(x)\, dx$$

σ^2 in terms of x

PDF of x

$$\int f(x) \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Subbing in this which is σ^2 in terms of x, and Solving this integral. If integral equals 0, then unbiased, else, biased

4) Risk is expectation of loss function

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

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

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

$E[X_i^2] - E[X_i]^2$

5) 1) X_i is singular, then it does not span across the entire space. This as a result means the columns that make up X_i don't cover the entire space and thus, cannot be linearly independent, and thus means the matrix is not invertible. This means $\hat{\Sigma}_i^{-1}$ will also not be linearly independent and not invertible

2) We will take all the colums that are not linearly independent and will add on a very small constant. A good way of doing this is to add 0.0001% of the column to each column. This ensures there is minimal error due to adding.

3) The vector x that maximizes the PDF

$$f(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_i|}} \exp(-((x-\overset{0}{\cancel{\mu}})^T \Sigma_i^{-1} (x-\overset{0}{\cancel{\mu}}))/2$$

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

The vector that gives the largest value of $\dfrac{-x^T \Sigma_i^{-1} x}{2}$ would maximize the PDF.

4) $Var(p) = E[(p - \cancel{E(p)})(p - \cancel{E(p)})^T]$

$E[pp]$

$E[y^T x x^T y]$

$y^T y$

$y^T y \underbrace{E[xx^T]}_{\Sigma_1}$

$\boxed{y^T y \Sigma_1}$

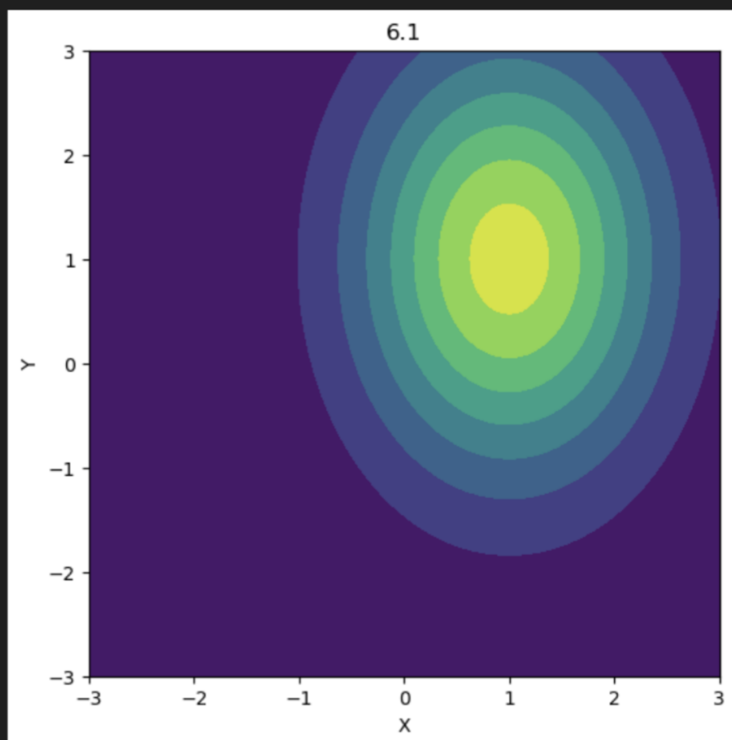
The max eigenvalue will give us the max $y^T X$ or projection on to the unit vector.

Question 6:

6.1

```
#Question 6.1
b = np.array([1, 1])
c = np.array([[1, 0], [0, 2]])

randvar = multivariate_normal(b, c)
zed = randvar.pdf(a)

plot.figure(figsize=(6,6))
plot.contourf(x, y, zed)
plot.title('6.1')
plot.xlabel('X')
plot.ylabel('Y')
plot.show()
```

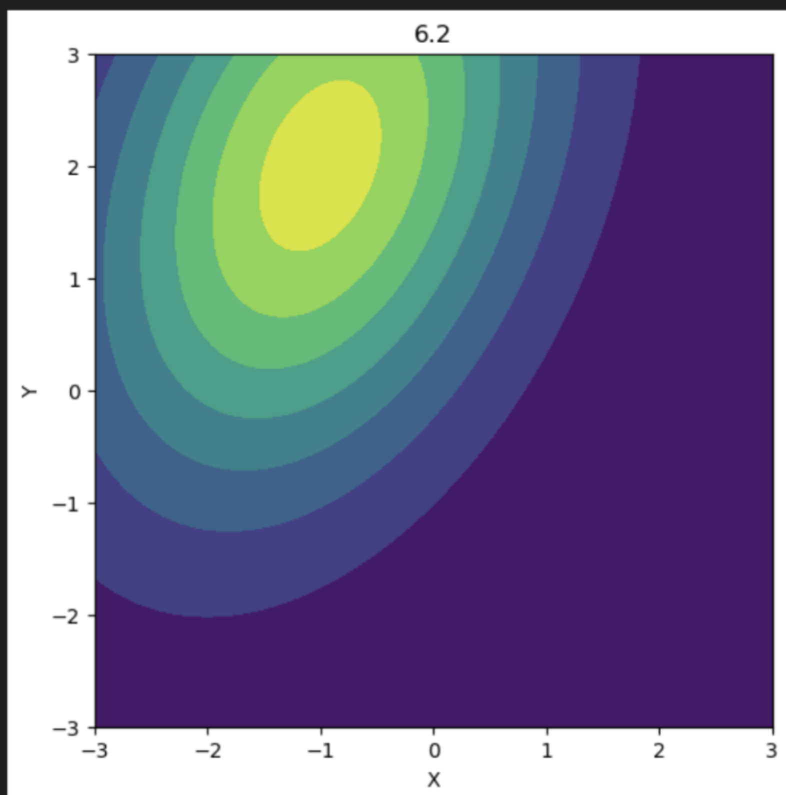
[19] ✓ 0.2s



6.2

```
#Question 6.2
b = np.array([-1, 2])
c = np.array([[2, 1], [1, 4]])

randvar = multivariate_normal(b, c)
zed = randvar.pdf(a)

plot.figure(figsize=(6,6))
plot.contourf(x, y, zed)
plot.title('6.2')
plot.xlabel('X')
plot.ylabel('Y')
plot.show()
```

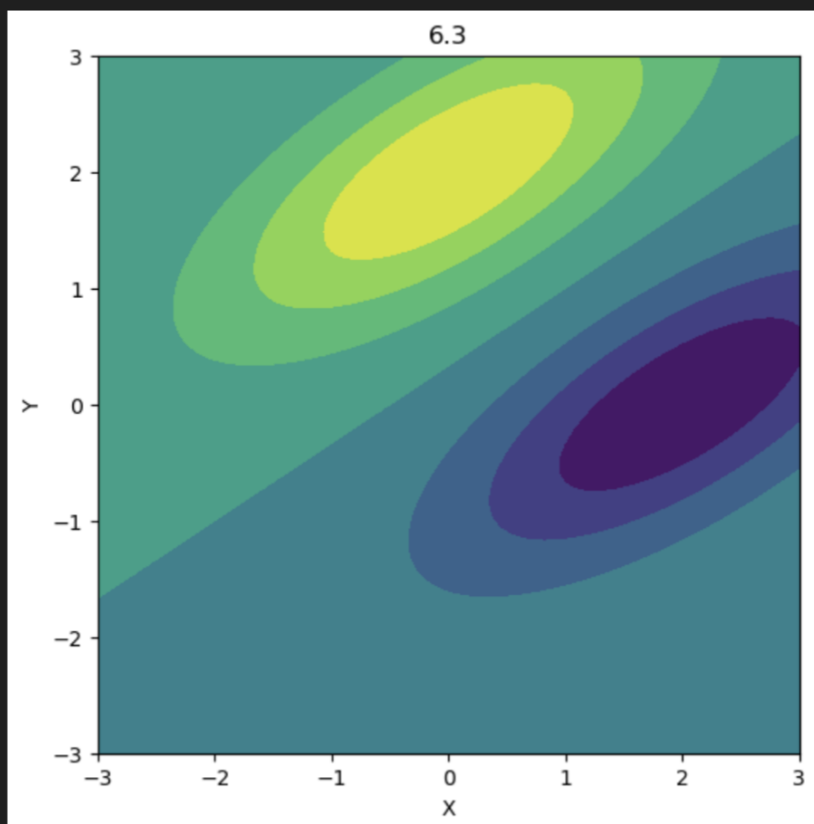
[20] ✓ 0.0s



6.3

```
#Question 6.3
b1 = np.array([0, 2])
b2 = np.array([2, 0])
c1 = c2 = np.array([[2, 1], [1, 1]])

randvar1 = multivariate_normal(b1, c1)
randvar2 = multivariate_normal(b2, c2)
zed = randvar1.pdf(a) - randvar2.pdf(a)

plot.figure(figsize=(6,6))
plot.contourf(x, y, zed)
plot.title('6.3')
plot.xlabel('X')
plot.ylabel('Y')
plot.show()
```

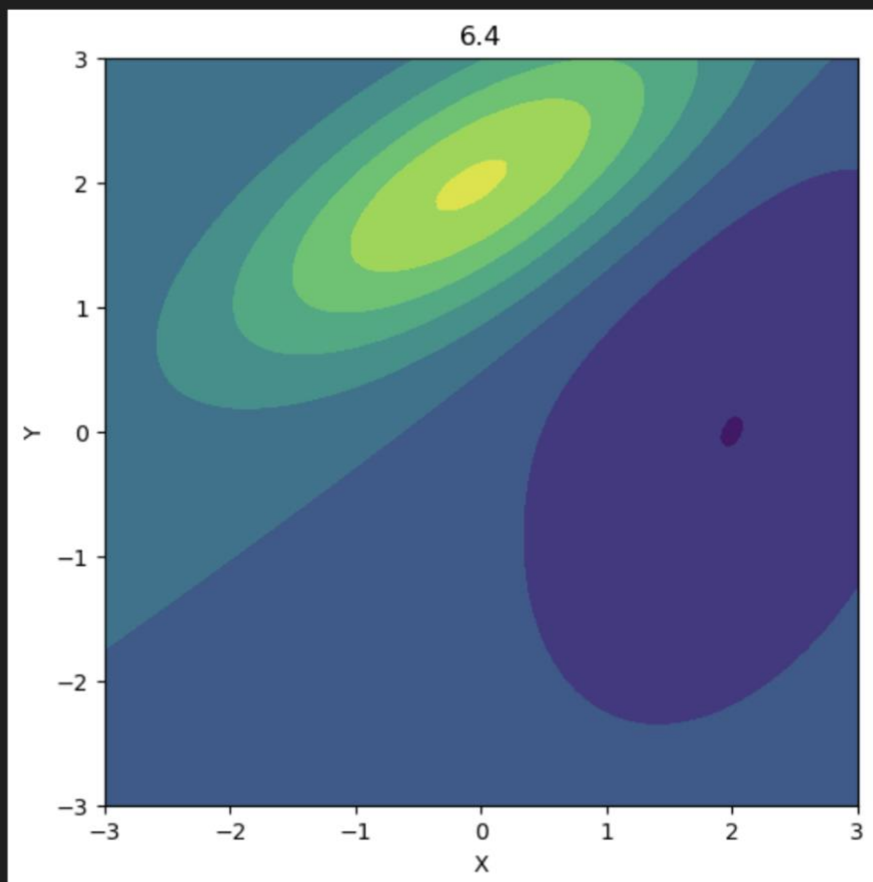
✓ 0.0s



6.4

```python
#Question 6.4
c2 = np.array([[2, 1], [1, 4]])

randvar2 = multivariate_normal(b2, c2)
zed = randvar1.pdf(a) - randvar2.pdf(a)

plot.figure(figsize=(6,6))
plot.contourf(x, y, zed)
plot.title('6.4')
plot.xlabel('X')
plot.ylabel('Y')
plot.show()
```

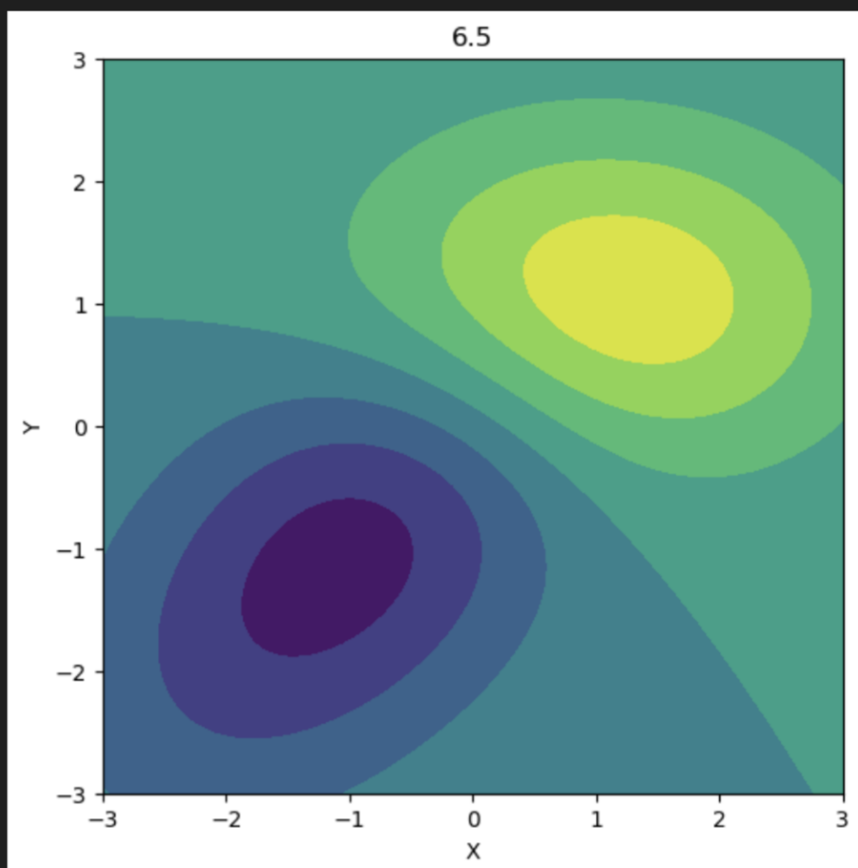
✓ 0.0s



6.5

```
#Question 6.5
b1 = np.array([1, 1])
b2 = np.array([-1, -1])
c1 = np.array([[2, 0], [0, 1]])
c2 = np.array([[2, 1], [1, 2]])

randvar1 = multivariate_normal(b1, c1)
randvar2 = multivariate_normal(b2, c2)
zed = randvar1.pdf(a) - randvar2.pdf(a)

plot.figure(figsize=(6,6))
plot.contourf(x, y, zed)
plot.title('6.5')
plot.xlabel('X')
plot.ylabel('Y')
plot.show()
```

✓ 0.0s



Question 7:

```
#Question 7
np.random.seed(0)

n = 100
X1_mu = 3
X1_sigma = 3
X2_mu = 4
X2_sigma = 2
a = 0.5

X1 = np.random.normal(X1_mu, X1_sigma, n)
print("X1:\n", X1, "\n\n")
X2 = a * X1 + np.random.normal(X2_mu, X2_sigma, n)
print("X2:\n", X2, "\n\n")
X = np.stack((X1, X2), axis=1)
print("X:\n", X, "\n\n")

print("X[:,0]:\n", X[:,0], "\n\n")
print("X[:,1]:\n", X[:,1], "\n\n")



mean = np.mean(X, axis=0)
print("Mean:\n", mean, "\n")

cov = np.cov(X, rowvar=False)
print("Covariance:\n", cov, "\n")

evals, evecs = np.linalg.eig(cov)
print("Eigenvalues:\n", evals, "\n")
print("Eigenvectors:\n", evecs, "\n")



plot.figure(figsize=(5,5))
plot.scatter(X[:,0], X[:,1], label='Data points')

plot.quiver(mean[0], mean[1], evecs[0,0], evecs[1,0], color='red', scale=evals[0], label='Evec1')
plot.quiver(mean[0], mean[1], evecs[0,1], evecs[1,1], color='green', scale=evals[1], label='Evec2')

plot.xlim(-15, 15)
plot.ylim(-15, 15)

plot.xlabel('X1')
plot.ylabel('X2')

plot.title('7.4')

plot.legend()
plot.show()



X_rot = evecs.T @ (X - mean).T
X_rot = X_rot.T

plot.figure(figsize=(5,5))
plot.scatter(X_rot[:,0], X_rot[:,1])

plot.xlim(-15, 15)
plot.ylim(-15, 15)

plot.xlabel('X1_rot')
plot.ylabel('X2_rot')

plot.title('7.5')
plot.show()
```

✓ 0.2s

7.1

Mean:
[3.17942405 5.75373796]

7.2

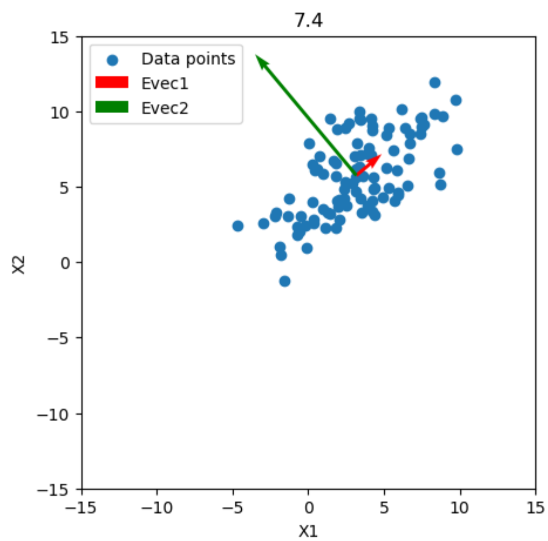
Covariance:
[[9.23478745 5.32353749]
[5.32353749 7.34023779]]

7.3

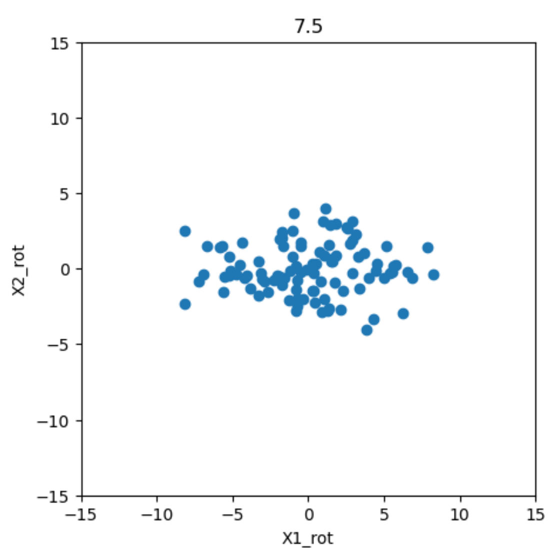
Eigenvalues:
[13.69467278 2.88035245]

Eigenvectors:
[[0.76654712 -0.64218807]
[0.64218807 0.76654712]]

7.4



7.5



Question 8:

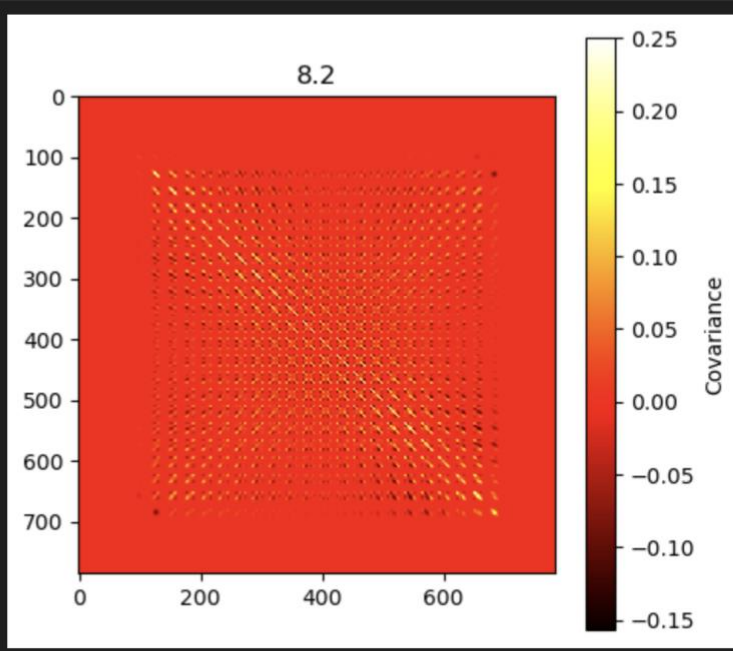
8.1

```python
#Question 8
data_npz = np.load('data/mnist-data-hw3.npz')
train_data = data_npz['training_data']
train_labels = data_npz['training_labels']

mean = {}
covariance = {}

for i in range(0,10):
    num_data = train_data[train_labels == i]

    l2 = np.linalg.norm(num_data, axis=1) + 0.0001

    num_data = num_data / l2[:, np.newaxis]
    num_data = num_data.reshape(num_data.shape[0], -1

    mean[i] = np.mean(num_data, axis=0)
    covariance[i] = np.cov(num_data, rowvar=False)
```

8.2

```python
covariance_matrix = covariance[0]

# Visualize the covariance matrix
plot.figure(figsize=(5, 5))
plot.imshow(covariance_matrix, cmap='hot', interpolation='nearest')
plot.title('8.2')
plot.colorbar(label='Covariance')
plot.show()
```

✓ 0.2s



8)2) Digit 0. The covariance in the diagonal and off-diagonal terms are larger, with the diagonal terms generally being the largest. This means the variance and covariance of the feature are linked together.

8.3a

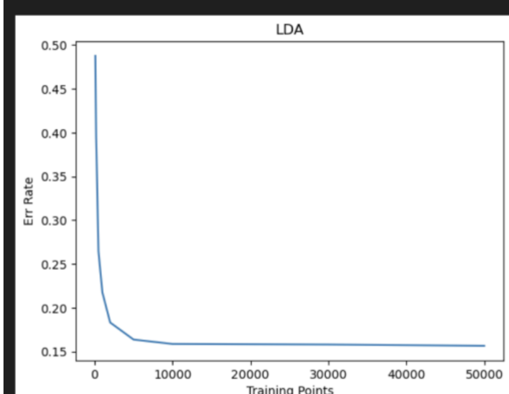
```python
#Question 8 LDA Working
data_npz = np.load('data/mnist-data-hw3.npz')
train_data = data_npz['training_data']
train_labels = data_npz['training_labels']

mean_dict = {}
cov_dict = {}

for j in range(0,10):
    num_data = train_data[train_labels == j]
    l2 = np.linalg.norm(num_data, axis=1) + 0.0001
    num_data = num_data / l2[:, np.newaxis]
    num_data = num_data.reshape(num_data.shape[0], -1)
    mean_dict[j] = np.mean(num_data, axis=0)
    cov_dict[j] = np.cov(num_data, rowvar=False)

mean_mat = np.array(list(mean_dict.values())).T
p_cov = np.mean(list(cov_dict.values()), axis=0)
p_cov += 0.000001 * np.eye(p_cov.shape[0])

class_prior = np.array([np.mean(train_labels == i) for i in range(0,10)])

val_indicies = np.random.choice(len(train_data), size=10000, replace=False)
val_data = train_data[val_indicies]
val_label = train_labels[val_indicies]

err_rates = []

train_size = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
for i in train_size:
    sub_indicies = np.random.choice(len(train_data), size=i, replace=False)
    sub_data = train_data[sub_indicies]
    sub_label = train_labels[sub_indicies]
    sub_mean_mat = np.array([np.mean(sub_data[sub_label == i], axis=0) for i in range(0,10)]).T

    sub_cov = []
    for j in range(0,10):
        data_label = sub_data[sub_label == j]
        if data_label.ndim == 2:
            sub_cov.append(np.cov(data_label, rowvar=False))
        else:
            sub_cov.append(p_cov)
    sub_cov = np.array(sub_cov)
    sub_inv_p_cov = np.linalg.inv(np.mean(sub_cov, axis=0) + 0.000001 * np.eye(sub_cov.shape[1]))
    sub_val_predict = lda(val_data, sub_mean_mat, sub_inv_p_cov, class_prior)

    sub_err_rate = 1 - np.sum(sub_val_predict == val_label) / len(val_label)
    err_rates.append(sub_err_rate)
```

✓ 5.6s

```python
print("Err Rate: ", err_rates)
plot.plot(train_size, err_rates)
plot.xlabel('Training Points')
plot.ylabel('Err Rate')
plot.title('LDA')
plot.show()
```

✓ 0.0s

```
Err Rate:  [0.4877, 0.3954, 0.2643, 0.21750000000000003, 0.18310000000000004, 0.16359999999999997, 0.15859999999999996, 0.15790000000000004, 0.15649999999999997]
```



8.3b

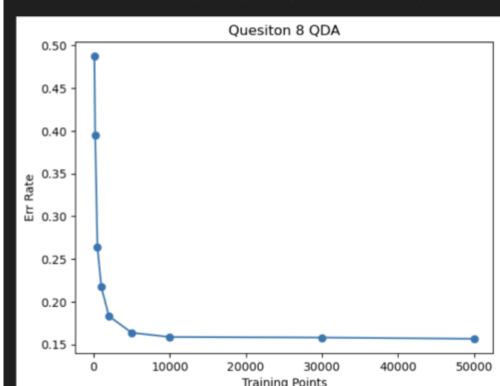
```python
#Question 8 QDA Working
data_npz = np.load('data/mnist-data-hw3.npz')
train_data = data_npz['training_data']
train_labels = data_npz['training_labels']

mean_dict = {}
cov_dict = {}

for j in range(10):
    num_data = train_data[train_labels == j]
    l2 = np.linalg.norm(num_data, axis=1) + 0.0001
    num_data = num_data / l2[:, np.newaxis]
    num_data = num_data.reshape(num_data.shape[0], -1)
    mean_dict[j] = np.mean(num_data, axis=0)
    cov_dict[j] = np.cov(num_data, rowvar=False)

mean_mat = np.array(list(mean_dict.values())).T
p_cov = np.mean(list(cov_dict.values()), axis=0)
p_cov += 0.000001 * np.eye(p_cov.shape[0])

class_prior = np.array([np.mean(train_labels == i) for i in range(0,10)])

val_indicies = np.random.choice(len(train_data), size=10000, replace=False)
val_data = train_data[val_indicies]
val_label = train_labels[val_indicies]

err_rates = []
for j in train_size:
    sub_indicies = np.random.choice(len(train_data), size=j, replace=False)
    sub_data = train_data[sub_indicies]
    sub_label = train_labels[sub_indicies]
    sub_mean_mat = np.array([np.mean(sub_data[sub_label == i], axis=0) for i in range(0,10)]).T

    sub_cov = []
    for i in range(0,10):
        data_label = sub_data[sub_label == i]
        if data_label.ndim == 2:
            sub_cov.append(np.cov(data_label, rowvar=False))
        else:
            sub_cov.append(p_cov)

    sub_cov = np.array(sub_cov)
    sub_inv_cov = [np.linalg.inv(cov + 0.00000001 * np.eye(cov.shape[0])) for cov in sub_cov]
    sub_val_predict = qda(val_data, sub_mean_mat, sub_inv_cov, class_prior)
    sub_err_rate = 1 - np.sum(sub_val_predict == val_label) / len(val_label)
    err_rates.append(sub_err_rate)
```

✓ 50.4s

```python
print(f'Validation Error Rate: {err_rates}')
plot.plot(train_size, err_rates, marker='o')
plot.xlabel('Training Points')
plot.ylabel('Err Rate')
plot.title('Quesiton 8 QDA')
plot.show()
```

✓ 0.0s

```
Validation Error Rate: [0.4877, 0.3954, 0.2643, 0.21750000000000003, 0.18310000000000004, 0.16359999999999997, 0.15859999999999996, 0.15790000000000004, 0.15649999999999997]
```



3) c) LDA performed better. My guess would be the digits are easier to handle if you handle the boundaries as straight lines rather than curves.

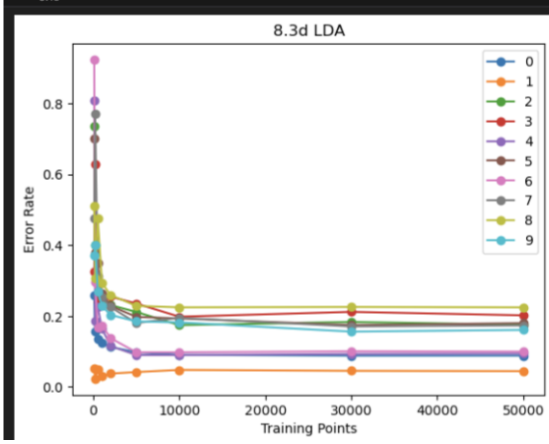
d) 1 was the easiest digit to classify. This is most likely because of how simple it is compared to other numbers.

Code on next page

LDA:

```python
#Question 8.3d LDA
data_npz = np.load('data/mnist-data-hw3.npz')
train_data = data_npz['training_data']
train_labels = data_npz['training_labels']

mean_dict = {}
cov_dict = {}

for j in range(10):
    num_data = train_data[train_labels == j]
    l2 = np.linalg.norm(num_data, axis=1) + 0.0001
    num_data = num_data / l2[:, np.newaxis]
    num_data = num_data.reshape(num_data.shape[0], -1)
    mean_dict[j] = np.mean(num_data, axis=0)
    cov_dict[j] = np.cov(num_data, rowvar=False)

mean_mat = np.array(list(mean_dict.values())).T
p_cov = np.mean(list(cov_dict.values()), axis=0)
smalladd = 1e-5
p_cov += smalladd * np.eye(p_cov.shape[0])
inv_p_cov = np.linalg.inv(p_cov)

class_prior = np.array([np.mean(train_labels == i) for i in range(0,10)])

val_indicies = np.random.choice(len(train_data), size=10000, replace=False)
val_data = train_data[val_indicies]
val_label = train_labels[val_indicies]
val_predict = lda(val_data, mean_mat, inv_p_cov, class_prior)
error = 1 - np.sum(val_predict == val_label) / len(val_label)

error_rates = []
err_rate_digit = {i: [] for i in range(0,10)}

train_size = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
for i in train_size:
    sub_indices = np.random.choice(len(train_data), size=i, replace=False)
    sub_data = train_data[sub_indices]
    sub_label = train_labels[sub_indices]
    sub_mean_mat = np.array([np.mean(sub_data[sub_label == i], axis=0) for i in range(0,10)]).T

    sub_cov = []
    for j in range(0,10):
        data_label = sub_data[sub_label == j]
        if data_label.ndim == 2:
            sub_cov.append(np.cov(data_label, rowvar=False))
        else:
            sub_cov.append(p_cov)

    sub_cov_mat = np.array(sub_cov)
    sub_p_cov = np.linalg.inv(np.mean(sub_cov_mat, axis=0) + smalladd * np.eye(sub_cov_mat.shape[1]))
    sub_val_predict = lda(val_data, sub_mean_mat, sub_inv_p_cov, class_prior)

    for j in range(0,10):
        digit_mask = val_label == j
        digit_error = 1 - np.sum(sub_val_predict[digit_mask] == val_label[digit_mask]) / np.sum(digit_mask)
        err_rate_digit[j].append(digit_error)


for i in range(0,10):
    plot.plot(train_size, err_rate_digit[i], marker='o', label=i)

plot.xlabel('Training Points')
plot.ylabel('Error Rate')
plot.title('8.3d LDA')
plot.legend()
plot.show()
```

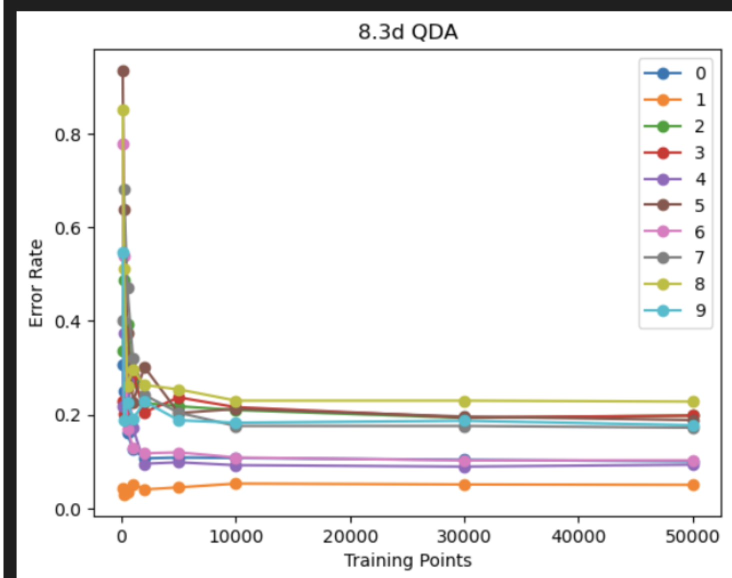
✓ 6.1s



8.3d

QDA

```
#Question 8.3d QDA
train = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
err_rate = []
err_rate_digit = {i: [] for i in range(0,10)}

for i in train:
    sub_indicies = np.random.choice(len(train_data), size=i, replace=False)
    sub_data = train_data[sub_indicies]
    sub_label = train_labels[sub_indicies]
    sub_mean_mat = np.array([np.mean(sub_data[sub_label == i], axis=0) for i in range(0,10)]).T

    sub_cov = []
    for j in range(0,10):
        data_label = sub_data[sub_label == j]
        if data_label.ndim == 2:
            sub_cov.append(np.cov(data_label, rowvar=False))
        else:
            sub_cov.append(p_cov)

    sub_cov = np.array(sub_cov)
    sub_inv_cov = [np.linalg.inv(cov + 0.000001 * np.eye(cov.shape[0])) for cov in sub_cov]
    sub_val_predict = qda(val_data, sub_mean_mat, sub_inv_cov, class_prior)

    for i in range(0,10):
        digit_indices = np.where(val_label == i)
        digit_predict = sub_val_predict[digit_indices]
        digit_label = val_label[digit_indices]
        digit_err = 1 - np.sum(digit_predict == digit_label) / len(digit_label)
        err_rate_digit[i].append(digit_err)

for i in range(0,10):
    plot.plot(train, err_rate_digit[i], marker='o', label=i)

plot.xlabel('Training Points')
plot.ylabel('Error Rate')
plot.title('8.3d QDA')
plot.legend()
plot.show()
```



8.4

Kaggle Username: Christopher Avakian

Kaggle Score: 0.845

```python
#Question 8.4
data_npz = np.load('data/mnist-data-hw3.npz')
train_data = data_npz['training_data']
train_labels = data_npz['training_labels']

test_data = data_npz['test_data']

mean_dict = {}
cov_dict = {}

for j in range(10):
    num_data = train_data[train_labels == j]
    l2 = np.linalg.norm(num_data, axis=1) + 0.0001
    num_data = num_data / l2[:, np.newaxis]
    num_data = num_data.reshape(num_data.shape[0], -1)
    mean_dict[j] = np.mean(num_data, axis=0)
    cov_dict[j] = np.cov(num_data, rowvar=False)

mean_mat = np.array(list(mean_dict.values())).T
p_cov = np.mean(list(cov_dict.values()), axis=0)
smalladd = 1e-5
p_cov += smalladd * np.eye(p_cov.shape[0])
inv_p_cov = np.linalg.inv(p_cov)

class_prior = np.array([np.mean(train_labels == label) for label in range(10)])

error_rates = []

train_size = [50000]
for i in train_size:
    sub_indicies = np.random.choice(len(train_data), size=i, replace=False)
    sub_data = train_data[sub_indicies]
    sub_label = train_labels[sub_indicies]
    sub_mean_mat = np.array([np.mean(sub_data[sub_label == i], axis=0) for i in range(0,10)]).T

    sub_cov = []
    for j in range(0,10):
        data_label = sub_data[sub_label == j]
        if data_label.ndim == 2:
            sub_cov.append(np.cov(data_label, rowvar=False))
        else:
            sub_cov.append(p_cov)

    sub_cov = np.array(sub_cov)
    sub_p_cov = np.mean(sub_cov, axis=0) + smalladd * np.eye(sub_cov.shape[1])
    sub_inv_p_cov = np.linalg.inv(sub_p_cov)
    sub_val_predict = lda(val_data, sub_mean_mat, sub_inv_p_cov, class_prior)

    sub_err_rate = 1 - np.sum(sub_val_predict == val_label) / len(val_label)
    error_rates.append(sub_err_rate)

results_to_csv(lda(test_data, sub_mean_mat, sub_inv_p_cov, class_prior))
```
✓ 4.1s

5) Kaggle Username: Christopher Avakian
Kaggle Score : 0.376 (Yeah IDK how I did that)

Code on next page

8.5

Kaggle Username: Christopher Avakian

Kaggle Score: 0.375

```
data_npz = np.load('data/spam-data-hw3.npz')
train_data = data_npz['training_data']
train_labels = data_npz['training_labels']

test_data = data_npz['test_data']

class_prior = np.array([np.mean(train_labels == i) for i in range(0,10)])
mean_dict = {}
cov_dict = {}

for j in range(0,1):
    num_data = train_data[train_labels == j]
    l2 = np.linalg.norm(num_data, axis=1) + 0.0001
    num_data = num_data / l2[:, np.newaxis]
    num_data = num_data.reshape(num_data.shape[0], -1)
    mean_dict[j] = np.mean(num_data, axis=0)
    cov_dict[j] = np.cov(num_data, rowvar=False)

mean_mat = np.array(list(mean_dict.values())).T
p_cov = np.mean(list(cov_dict.values()), axis=0)
smalladd = 0.000001
p_cov += smalladd * np.eye(p_cov.shape[0])
inv_p_cov = np.linalg.inv(p_cov)
error_rates = []

sub_data = train_data
sub_label = train_labels
sub_mean_mat = np.array([np.mean(sub_data[sub_label == i], axis=0) for i in range(0,1)]).T

sub_cov = []
for j in range(0,1):
    data_label = sub_data[sub_label == j]
    if data_label.ndim == 2 and data_label.shape[0] > 1:
        sub_cov.append(np.cov(data_label, rowvar=False))
    else:
        sub_cov.append(p_cov)

sub_cov = np.array(sub_cov)
sub_p_cov = np.mean(sub_cov, axis=0) + 0.00001 * np.eye(sub_cov.shape[1])
sub_inv_p_cov = np.linalg.inv(sub_p_cov)

results_to_csv(lda(test_data, sub_mean_mat, sub_inv_p_cov, class_prior))
```

✓ 0.0s

References:

- https://en.wikipedia.org/wiki/Bayes_error_rate#:~:text=The%20Bayes%20error%20rate%20of%20the%20data%20distribution,knows%20the%20true%20class%20probabilities%20given%20the%20predictors.
- https://www.geeksforgeeks.org/gaussian-discriminant-analysis/
- https://cs229.stanford.edu/notes2021spring/notes2021spring/lecture5_live.pdf
- https://towardsdatascience.com/gaussian-discriminant-analysis-an-example-of-generative-learning-algorithms-2e336ba7aa5c
- https://en.wikipedia.org/wiki/Multivariate_normal_distribution
- https://kuleshov-group.github.io/aml-book/contents/lecture7-gaussian-discriminant-analysis.html
- https://aman.ai/cs229/gda/
- https://online.stat.psu.edu/stat508/book/export/html/645
- https://en.wikipedia.org/wiki/Loss_function
- https://en.wikipedia.org/wiki/Maximum_likelihood_estimation
- https://cs229.stanford.edu/section/gaussians.pdf
- https://kmoy1.github.io/ML_Book/chapters/Ch8/intro.html
- https://spectra.mathpix.com/article/2022.03.00195/introduction-to-the-multivariate-gaussian-distribution
- https://online.stat.psu.edu/stat505/lesson/10/10.3
- https://medium.com/swlh/linear-discriminant-analysis-basics-with-hands-on-practice-7b130a18d220
- http://www.stat.ucla.edu/~ywu/research/documents/BOOKS/LinearDiscriminantAnalysis.pdf
- https://en.wikipedia.org/wiki/Linear_discriminant_analysis
- https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html
- https://scikit-learn.org/stable/index.html#
- https://numpy.org/doc/1.26/index.html
-