CS 189: Homework 3

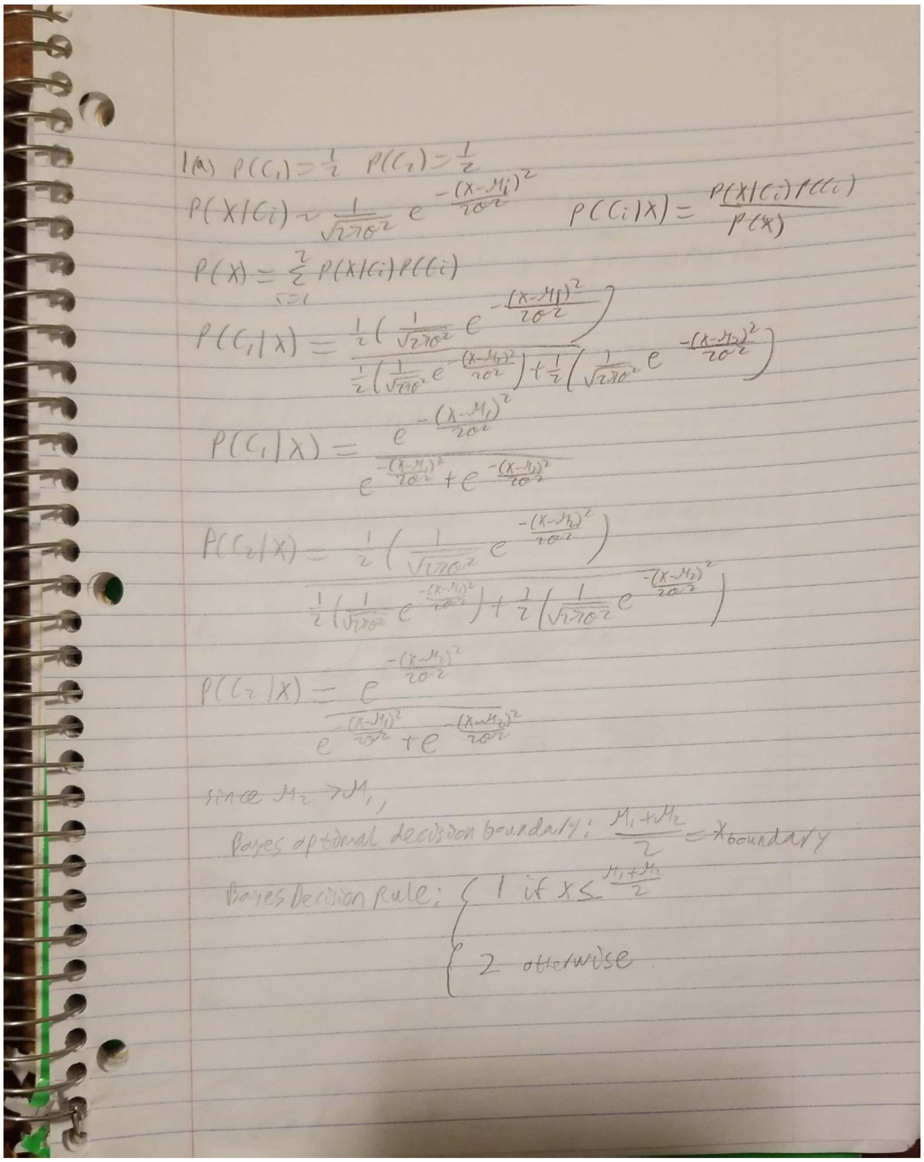
Vinay Maruri

February 21, 2019

I worked with Bob Feng and Neel Venugopal on this homework.
"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."
Signature: Vinay Maruri

1 Gaussian Classification

(a) Find the Bayes optimal decision boundary and the corresponding Bayes decision rule.
Solution: On next page in the image.



1(a) $P(C_1) = \frac{1}{2}$ $P(C_2) = \frac{1}{2}$

$P(x|C_i) \sim \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu_i)^2}{2\sigma^2}}$ $P(C_i|x) = \frac{P(x|C_i)P(C_i)}{P(x)}$

$P(x) = \sum_{i=1}^{2} P(x|C_i)P(C_i)$

$$P(C_1|x) = \frac{\frac{1}{2}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}\right)}{\frac{1}{2}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}\right) + \frac{1}{2}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}\right)}$$

$$P(C_1|x) = \frac{e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}}{e^{-\frac{(x-\mu_1)^2}{2\sigma^2}} + e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}}$$

$$P(C_2|x) = \frac{\frac{1}{2}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}\right)}{\frac{1}{2}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}\right) + \frac{1}{2}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}\right)}$$

$$P(C_2|x) = \frac{e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}}{e^{-\frac{(x-\mu_1)^2}{2\sigma^2}} + e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}}$$

since $\mu_2 > \mu_1$,

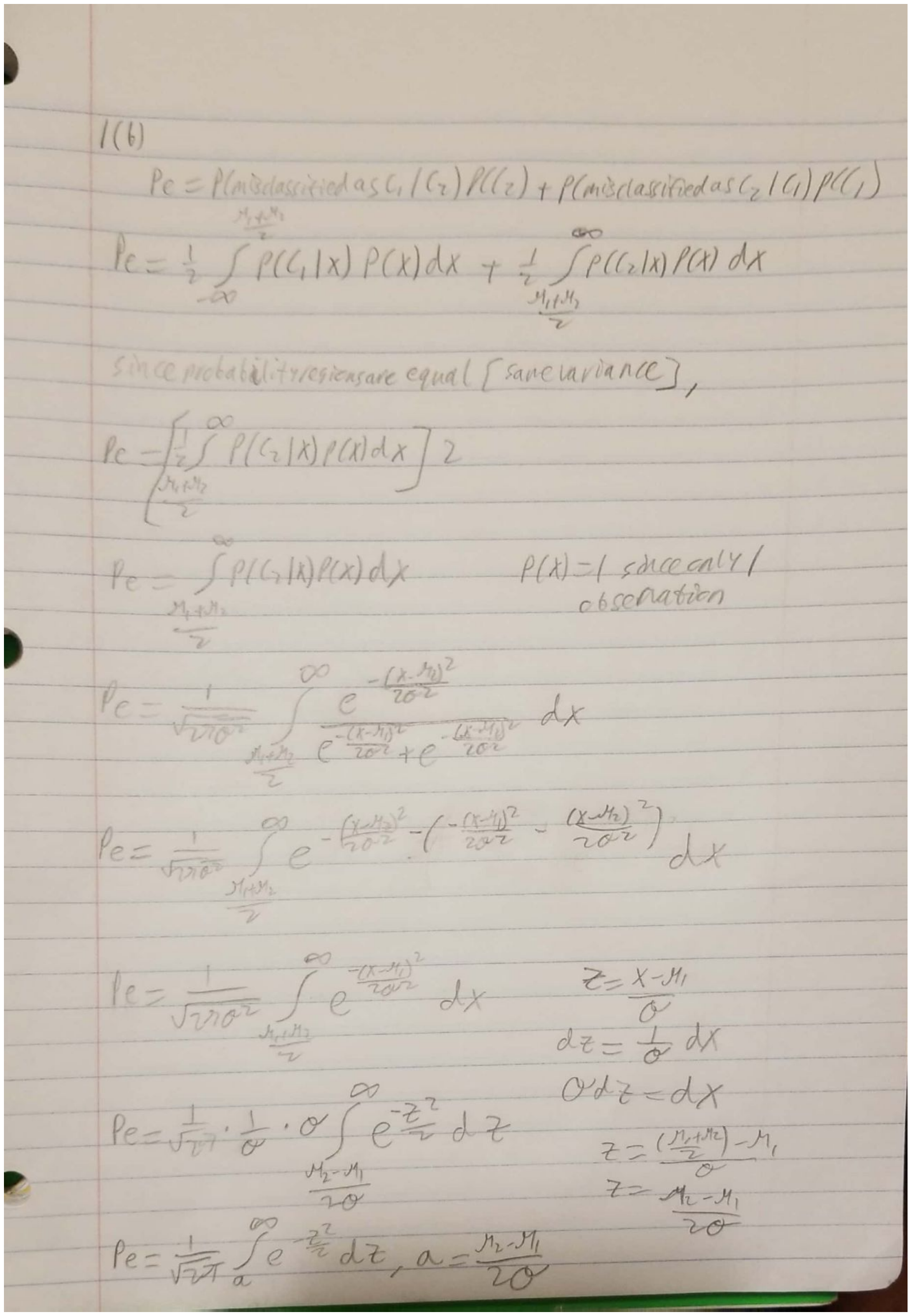
Bayes optimal decision boundary: $\frac{\mu_1+\mu_2}{2} = x_{boundary}$

Bayes Decision Rule: $\begin{cases} 1 & \text{if } x \leq \frac{\mu_1+\mu_2}{2} \\ 2 & \text{otherwise} \end{cases}$

Figure 1: Proof for question 1a.

(b) Show that the Bayes error associated with this decision rule is $\int_a^\infty e^{-\frac{z^2}{2}}\,dz$, where $a = \frac{\mu_2 - \mu_1}{2}$.

Solution: On next page in the image.



1(b)

$$P_e = P(\text{misclassified as } C_1 | C_2) P(C_2) + P(\text{misclassified as } C_2 | C_1) P(C_1)$$

$$P_e = \frac{1}{2} \int_{-\infty}^{\frac{\mu_1+\mu_2}{2}} P(C_1|x) P(x)\, dx + \frac{1}{2} \int_{\frac{\mu_1+\mu_2}{2}}^{\infty} P(C_2|x) P(x)\, dx$$

since probability regions are equal [same variance],

$$P_e = \left[\frac{1}{2} \int_{\frac{\mu_1+\mu_2}{2}}^{\infty} P(C_2|x) P(x)\, dx\right] 2$$

$$P_e = \int_{\frac{\mu_1+\mu_2}{2}}^{\infty} P(C_2|x) P(x)\, dx \qquad P(x) = 1 \text{ since only 1 observation}$$

$$P_e = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{\frac{\mu_1+\mu_2}{2}}^{\infty} \frac{e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}}{e^{-\frac{(x-\mu_1)^2}{2\sigma^2}} + e^{-\frac{(x-\mu_2)^2}{2\sigma^2}}}\, dx$$

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

$$P_e = \frac{1}{\sqrt{2\pi\sigma^2}} \int_{\frac{\mu_1+\mu_2}{2}}^{\infty} e^{\frac{-(x-\mu_1)^2}{2\sigma^2}}\, dx \qquad z = \frac{x-\mu_1}{\sigma}$$

$$dz = \frac{1}{\sigma} dx$$

$$\sigma dz = dx$$

$$P_e = \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{\sigma} \cdot \sigma \int_{\frac{\mu_2-\mu_1}{2\sigma}}^{\infty} e^{\frac{-z^2}{2}}\, dz \qquad z = \frac{\left(\frac{\mu_1+\mu_2}{2}\right) - \mu_1}{\sigma}$$

$$z = \frac{\mu_2-\mu_1}{2\sigma}$$

$$P_e = \frac{1}{\sqrt{2\pi}} \int_a^{\infty} e^{-\frac{z^2}{2}}\, dz, \quad a = \frac{\mu_2-\mu_1}{2\sigma}$$

Figure 2: Proof for question 1b.

Out[3]: <matplotlib.colorbar.Colorbar at 0x1f34f276898>

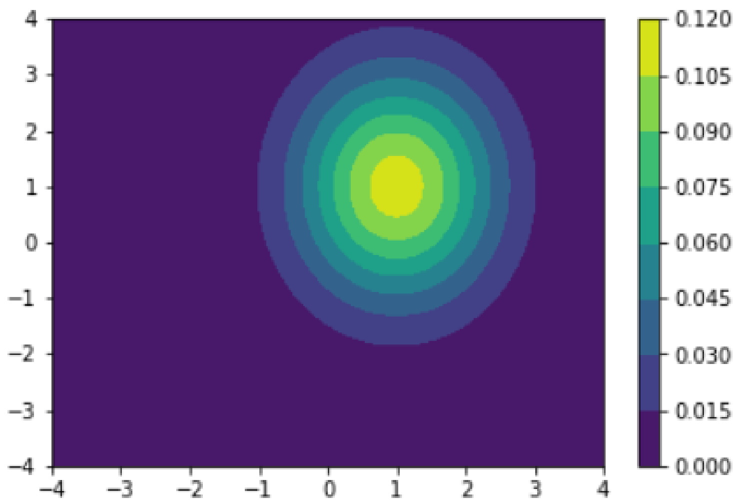


Figure 3: Graph for question 2a.

2 Isocontours of Normal Distributions

(a) **Graph for this question is above.**

```
x1, y1 = np.mgrid[-4:4:.01, -4:4:.01]
#setting the size of the grid.
mean1 = [1, 1]
cov1 = [[1, 0], [0, 2]]
#setting the mean and covariance matrix.
pos1 = np.dstack((x1, y1))
#stacking into 3-dimensions for the contourplot
rv1 = multivariate_normal(mean1, cov1)
plt.contourf(x1, y1, rv1.pdf(pos1))
plt.colorbar()
```

Out[5]: <matplotlib.colorbar.Colorbar at 0x1f352d19080>

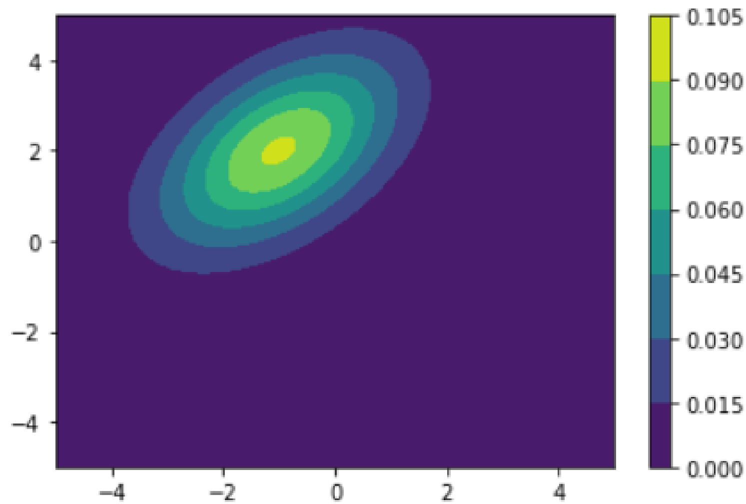


Figure 4: Graph for question 2b.

(b) **Graph for this question is above.**

```
x2, y2 = np.mgrid[-5:5:.01, -5:5:.01]
#setting the size of the grid.
mean2 = [-1, 2]
cov2 = [[2, 1], [1, 2]]
#setting the mean and covariance matrix.
pos2 = np.dstack((x2, y2))
#stacking into 3-dimensions for the contourplot
rv2 = multivariate_normal(mean2, cov2)
plt.contourf(x2, y2, rv2.pdf(pos2))
plt.colorbar()
```

Out[7]: <matplotlib.colorbar.Colorbar at 0x1f356394fd0>

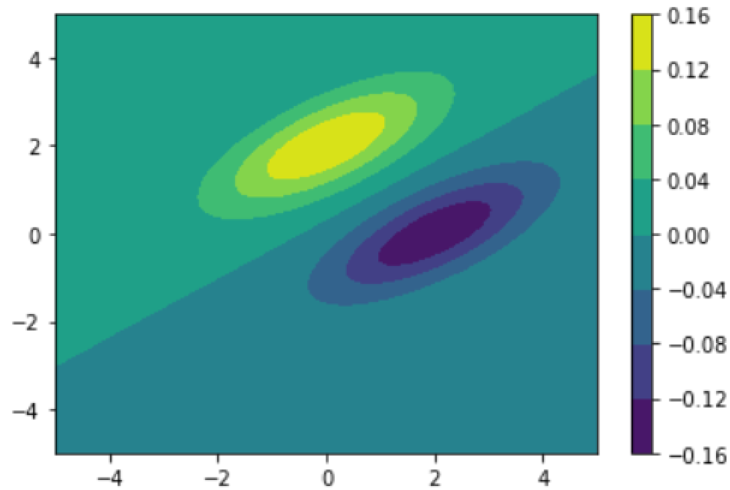


Figure 5: Graph for question 2c.

(c) **Graph for this question is above.**

```
x3, y3 = np.mgrid[-5:5:.01, -5:5:.01]
#setting the size of the grid.
mean3 = [0, 2]
mean4 = [2, 0]
cov3 = [[2, 1], [1, 1]]
#setting the means and covariance matrix.
pos3 = np.dstack((x3, y3))
#stacking into 3-dimensions for the contourplot
rv3 = multivariate_normal(mean3, cov3)
rv4 = multivariate_normal(mean4, cov3)
plt.contourf(x3, y3, (rv3.pdf(pos3) - rv4.pdf(pos3)))
plt.colorbar()
```

Out[9]: <matplotlib.colorbar.Colorbar at 0x1f35644f588>

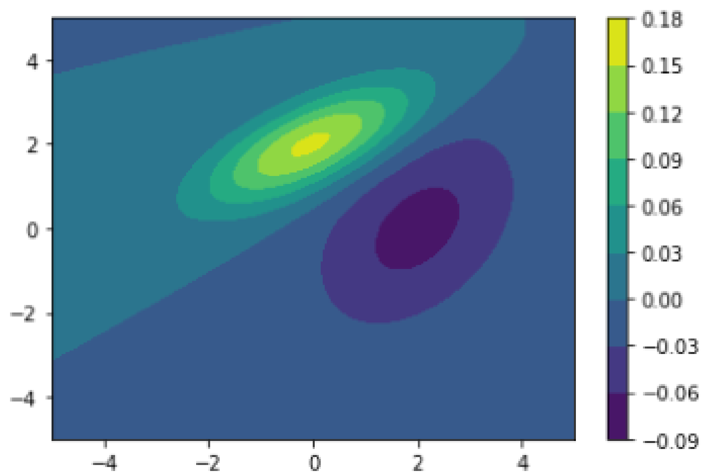


Figure 6: Graph for question 2d.

(d) **Graph for this question is above.**

```
x4, y4 = np.mgrid[-5:5:.01, -5:5:.01]
#setting the size of the grid.
mean5 = [0, 2]
mean6 = [2, 0]
cov4 = [[2, 1], [1, 1]]
cov5 = [[2, 1], [1, 3]]
#setting the means and covariance matrices.
pos4 = np.dstack((x4, y4))
#stacking into 3-dimensions for the contourplot
rv5 = multivariate_normal(mean5, cov4)
rv6 = multivariate_normal(mean6, cov5)
plt.contourf(x4, y4, (rv5.pdf(pos4) - rv6.pdf(pos4)))
plt.colorbar()
```

```
Out[11]: <matplotlib.colorbar.Colorbar at 0x1f356502c18>
```

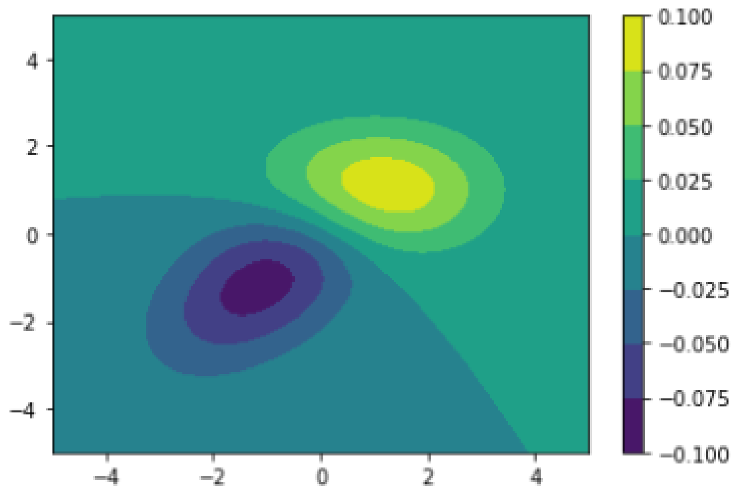


Figure 7: Graph for question 2e.

(e) **Graph for this question is above.**

```
x5, y5 = np.mgrid[-5:5:.01, -5:5:.01]
#setting the size of the grid.
mean7 = [1, 1]
mean8 = [-1, -1]
cov6 = [[2, 0], [0, 1]]
cov7 = [[2, 1], [1, 2]]
#setting the means and covariance matrices.
pos5 = np.dstack((x5, y5))
#stacking into 3-dimensions for the contourplot
rv7 = multivariate_normal(mean7, cov6)
rv8 = multivariate_normal(mean8, cov7)
plt.contourf(x5, y5, (rv7.pdf(pos5) - rv8.pdf(pos5)))
plt.colorbar()
```

3 Eigenvectors of Gaussian Covariance Matrix

(a) Compute the mean (in R^2)
Solution: The mean in R^2 is (1.7012325212518036, 3.985641868266835)

```
np.random.seed(12)
#seed is 42.
x1 = np.random.normal(3, 9, 100)
```

```python
#generating the first normal RV
x2 = (0.5 * x1) + np.random.normal(4, 4, 100)
#generating the second normal RV
merged_x1_x2 = np.column_stack((x1, x2))
#created the merged 2d set of points from x1 and x2
#part (a) compute mean in R^2 of the sample
firstelems = [merged_x1_x2[i][0] for i in range(len(merged_x1_x2))]
secondelems = [merged_x1_x2[i][1] for i in range(len(merged_x1_x2))]
meanr2 = (np.mean(firstelems), np.mean(secondelems))
meanr2
#this is the mean
```

(b) Compute the 2 x 2 covariance matrix of the sample
Solution: [[89.51010777 40.16868947], [40.16868947 34.81704039]]

```python
#part (b) compute the 2x2 covariance matrix of the sample
covariance_matrix = np.cov(merged_x1_x2, rowvar = False)
print(covariance_matrix)
```

(c) Compute the eigenvalues and eigenvectors of this covariance matrix.
Eigenvalues: [110.75736503, 13.56978313], Eigenvectors: [[0.88395638, -0.46756937], [0.46756937, 0.88395638]]

```python
#part (c) compute the eigenvectors and eigenvalues of this covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
```

(d) On a two-dimensional grid with a horizontal 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.
Solution:

```python
plt.scatter(firstelems, secondelems)
plt.xlim(left = -15, right = 15)
plt.ylim(bottom = -15, top = 15)
# Add the mean value to the plot
plt.plot(meanr2[0], meanr2[1], marker='*', color='black', markersize=15)
# Add arrows showing the eigenvectors
plt.quiver([meanr2[0]]*2, [meanr2[1]]*2, eigenvectors[:,1],
eigenvectors[:,0], zorder=11, width=0.01, scale=3.75)
```

```
Out[38]: <matplotlib.quiver.Quiver at 0x19c0bbfe0b8>
```

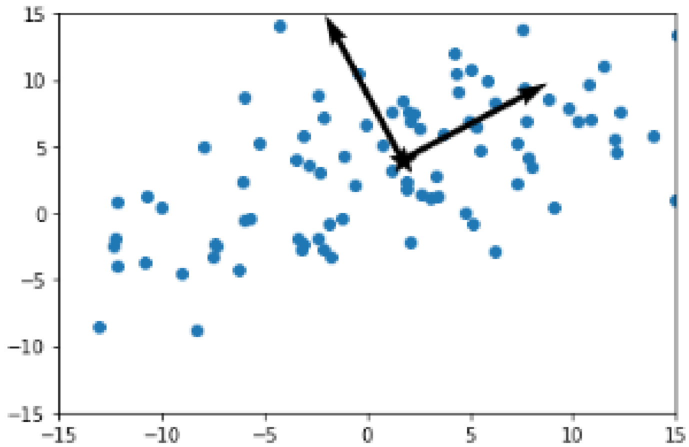


Figure 8: Graph for question 3d.

(e) Plot rotated points on a new two dimensional-grid, again with both axes having range [15, 15].

Solution:

```
ut = np.transpose(eigenvectors)
constant_matrix = np.array([x - meanr2 for x in merged_x1_x2])
x_rotated = np.transpose(ut @ np.transpose(constant_matrix))
x_rotated_first = [x_rotated[i][0] for i in range(len(x_rotated))]
x_rotated_second = [x_rotated[i][1] for i in range(len(x_rotated))]
plt.scatter(x_rotated_first, x_rotated_second)
plt.xlim(-15, 15)
plt.ylim(-15, 15)
```

Out[12]: (-15, 15)

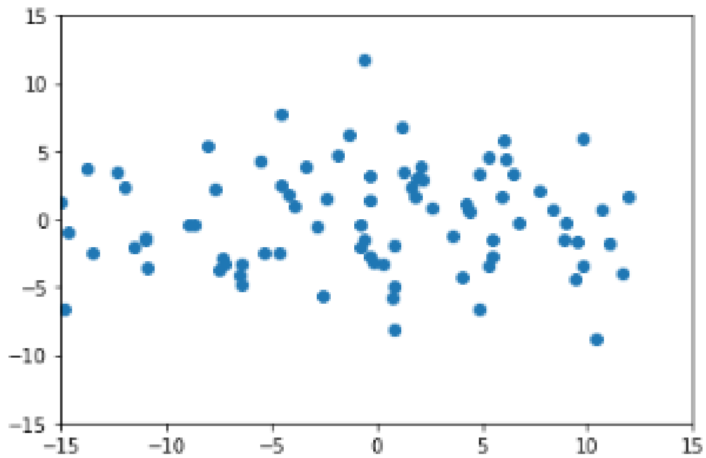


Figure 9: Graph for question 3e.

4 Classification

(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 $P(Y=i|x) = 1 - \frac{\lambda_r}{\lambda_s}$
(2) Choose doubt otherwise.
Solution: On next 3 pages in the images.

Homework 3, Question 4

(a) $R(r(X)=i|X) = \sum_{j=1} L(r(X)=i, Y=j) P(Y=j|X)$

choose class i if $P(Y=i|X) \geq P(Y=j|X)$ for all j and $P(Y=i|X) \geq 1-\frac{\lambda_r}{\lambda_s}$. Choose doubt otherwise.

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

Let $i \in \{1, \dots, c\}$:

Then $R(r(X)=i|X) = \sum_{j=i} 0 \cdot P(Y=i|X) + \sum_{j \neq i} \lambda_s \cdot P(Y=j|X)$

$= \lambda_s \cdot P(Y=j|X) = \lambda_s(1-P(Y=i|X))$

(since j and i are the only 2 class labels Y can take for an X, they are complements)

Let $i = c+1$:

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

$= \lambda_r$

Let $r^*(x)$ be the decision rule that implements the rules/policies stated in the question.

Let $x \in \mathbb{R}^d$ be a point we wish to classify.

Let $f(x)$ be a different decision rule from $r^*(x)$ that behaves differently.

I will show that $r^*(x)$ performs as good or better than $f(x)$.

Let $r^*(x)$ predict i, $i \in \{1, \dots, c\}$:

If $f(x)$ predicts j, $j \neq i$, $j \in \{1, \dots, c\}$:

Case A $\rightarrow$ $R(r^*(x)=i|X) = \lambda_s(1-P(Y=i|X))$

Let $R(f(x)=j|X) = \lambda_s(1-P(Y=j|X))$

Figure 10: Proof for question 4a.

Because $P(Y=i|X) \geq P(Y=j|X)$ for all j:

$R(r^*(X)=i|X) \leq R(f(X)=j|X)$ for all j in case A.

Case B: Let $f(X)$ predict $c+1$:

$R(r^*(X)=i|X) = \lambda_s(1-P(Y=i|X))$

Let $R(f(X)=c+1|X) = \lambda_r$

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

$$R(r^*(X)=i|X) = \lambda_s\left(1-\left(1-\frac{\lambda_r}{\lambda_s}\right)\right) = \lambda_s\left(\frac{\lambda_r}{\lambda_s}\right) = \lambda_r$$

Since $1-\frac{\lambda_r}{\lambda_s}$ is the minimum value $P(Y=i|X)$ can take,

$\Rightarrow R(r^*(X)=i|X) \leq \lambda_r$

Thus, $R(r^*(X)=i|X) \leq R(f(X)=c+1|X)$

Let $r^*(X)$ predict $c+1$:

Case C: Let $f(X)$ predict $c+1$:

$R(r^*(X)=c+1|X) = \lambda_r$

Let $R(f(X)=c+1|X) = \lambda_r$

$R(r^*(X)=c+1|X) = R(f(X)=c+1|X)$

Case D: Let $f(X)$ predict $j \in \{1,\dots,c\}$:

$R(r^*(X)=c+1|X) = \lambda_r$

$R(f(X)=j|X) = \lambda_s(1-P(Y=j|X))$

Figure 11: Proof for question 4a.

Let $P(Y=j|X) = 1 - \frac{\lambda_r}{\lambda_s}$

$$R(f(x)=j|x) = \lambda_s\left(1-\left(1-\frac{\lambda_r}{\lambda_s}\right)\right) = \lambda_s\left(\frac{\lambda_r}{\lambda_s}\right) = \lambda_r$$

However, since $P(Y=i|X) \geq 1-\frac{\lambda_r}{\lambda_s} \geq P(Y=j|x)$,

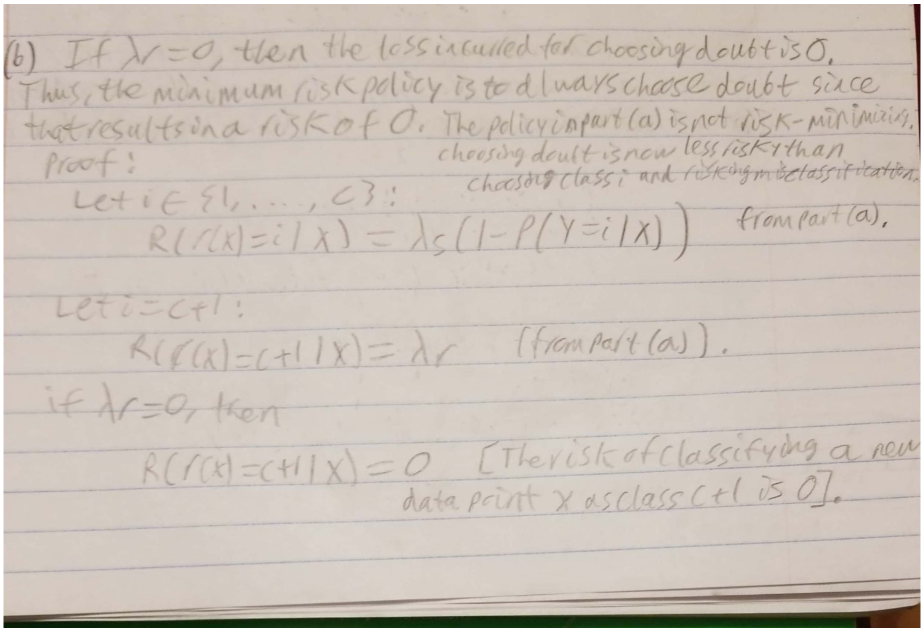
$1-\frac{\lambda_r}{\lambda_s}$ is the maximum value $P(Y=j|X)$ can take,

and $R(f(x)=j|x) \geq \lambda_r$

Thus, in Case D, $R(r^*(x)=c+1|x) \leq R(f(x)=j|x)$.

Thus for all possible cases, r^*, which implements the decision rule policy obtains the minimum risk in comparison to alternate decision rule policies.

Figure 12: Proof for question 4a.



(b) If $\lambda_r = 0$, then the loss incurred for choosing doubt is 0. Thus, the minimum risk policy is to always choose doubt since that results in a risk of 0. The policy in part (a) is not risk-minimizing; choosing doubt is now less risky than choosing class i and risking misclassification.

Proof:

Let $i \in \{1, \ldots, c\}$:

$$R(r(x) = i \mid x) = \lambda_s (1 - P(Y = i \mid x))$$ from part (a).

Let $i = c+1$:

$$R(r(x) = c+1 \mid x) = \lambda_r$$ (from part (a)).

if $\lambda_r = 0$, then

$$R(r(x) = c+1 \mid x) = 0$$ [The risk of classifying a new data point x as class c+1 is 0].

Figure 13: Proof for question 4b.

(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 is in Figure 6 above and on the next page figure 7.

If $\lambda_r > \lambda_s$, then the policy in part (a) is no longer risk minimizing. It is now better to misclassify a data point rather than choose doubt.

Proof: from part a:

let $i \in \{1, \dots, c\}$:

$$R(r(x)=i|x) = \lambda_s(1-P(Y=i|x))$$

let $i = c+1$:

$$R(r(x)=c+1|x) = \lambda_r$$

since $\lambda_r > \lambda_s$, for all possible values of $P(Y=i|x)$ [$0 \leq P(Y=i|x) \leq 1$],

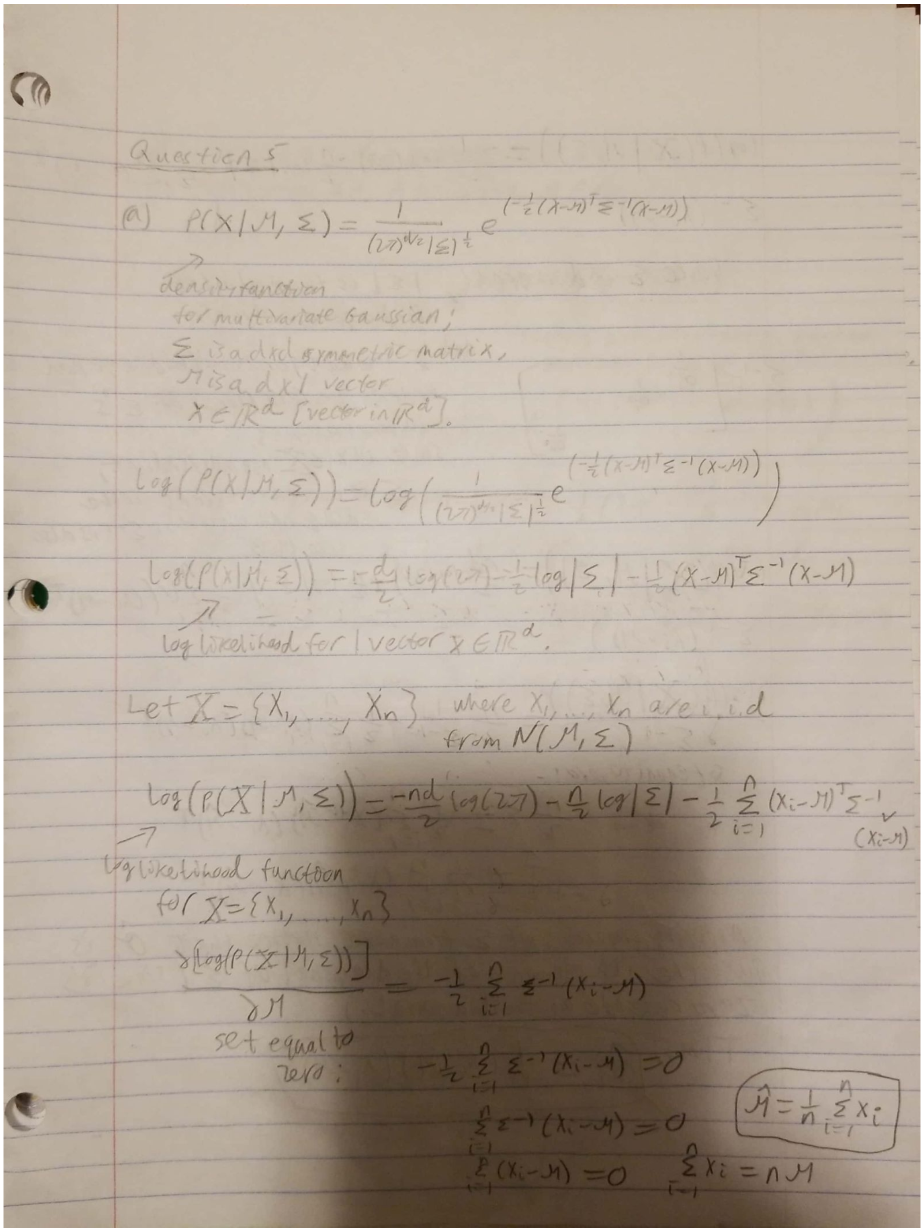
$$R(r(x)=i|x) \leq R(r(x)=c+1|x)$$

Thus, the new risk-minimizing policy intuitively is to always choose class i, since misclassification and correct predictions are less risky than choosing doubt.

Figure 14: Proof for question 4b.

5 Maximum Likelihood Estimation

(a) Derive the maximum likelihood estimates for μ and σ_i.
Solution on next 2 pages in the images.



Question 5

(a) $P(X|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right)}$

density function for multivariate Gaussian; Σ is a $d \times d$ symmetric matrix, μ is a $d \times 1$ vector $X \in \mathbb{R}^d$ [vector in $\mathbb{R}^d$].

$\log(P(X|\mu, \Sigma)) = \log\left(\frac{1}{(2\pi)^{d/2}|\Sigma|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right)}\right)$

$\log(P(X|\mu, \Sigma)) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)$

log likelihood for 1 vector $X \in \mathbb{R}^d$.

Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ where $X_1, \ldots, X_n$ are i.i.d from $N(\mu, \Sigma)$

$\log(P(\mathcal{X}|\mu, \Sigma)) = -\frac{nd}{2}\log(2\pi) - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(X_i-\mu)^T\Sigma^{-1}(X_i-\mu)$

log likelihood function for $\mathcal{X} = \{X_1, \ldots, X_n\}$

$\frac{\partial[\log(P(\mathcal{X}|\mu, \Sigma))]}{\partial\mu} = -\frac{1}{2}\sum_{i=1}^{n}\Sigma^{-1}(X_i-\mu)$

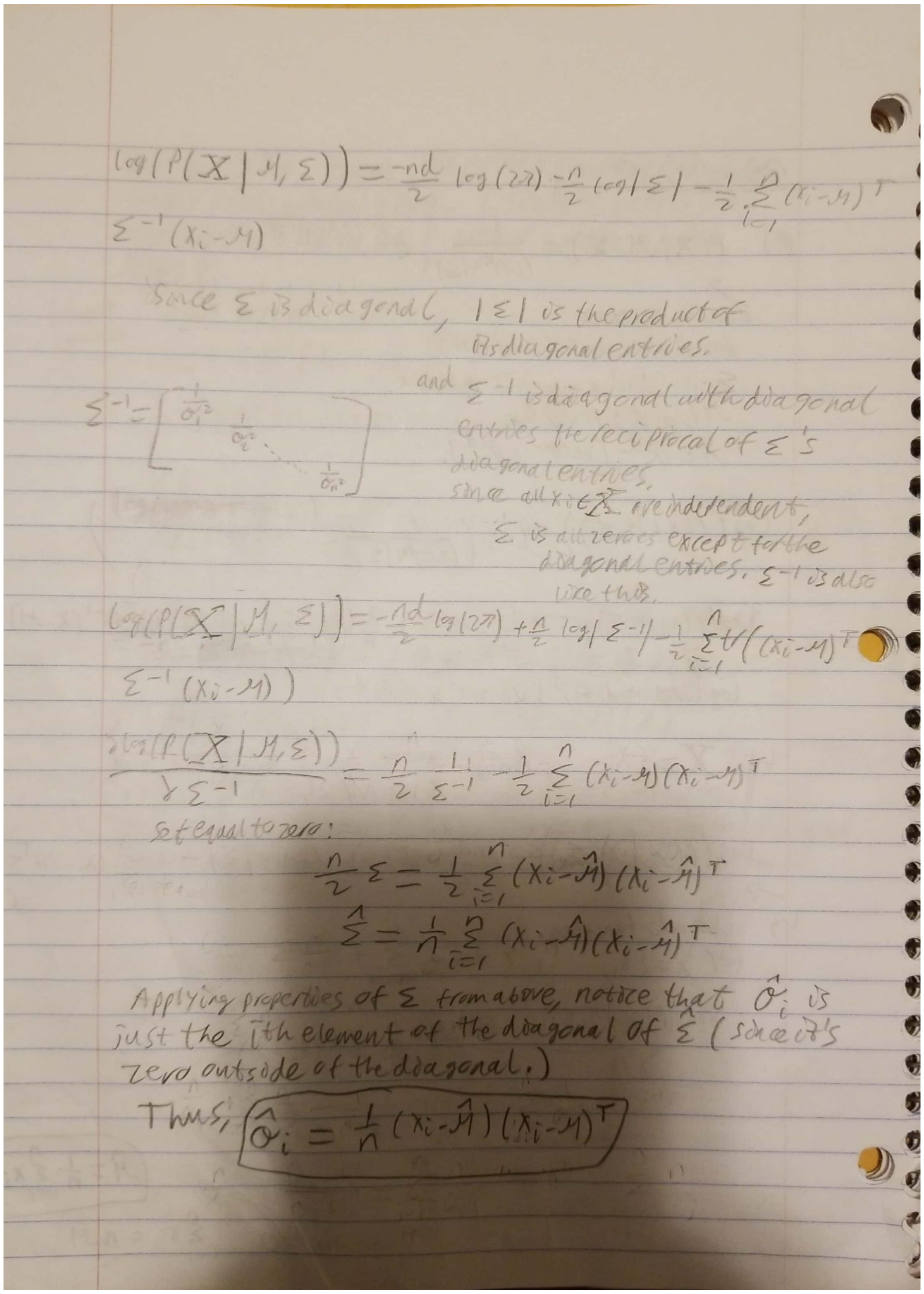
set equal to zero: $-\frac{1}{2}\sum_{i=1}^{n}\Sigma^{-1}(X_i-\mu) = 0$

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

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

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

Figure 15: Proof for question 5a.



$\log(P(X \mid \mu, \Sigma)) = -\frac{nd}{2}\log(2\pi) - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T \Sigma^{-1}(x_i-\mu)$

Since Σ is diagonal, $|\Sigma|$ is the product of its diagonal entries.

$$\Sigma^{-1} = \begin{bmatrix} \frac{1}{\sigma_1^2} & & \\ & \frac{1}{\sigma_2^2} & \\ & & \ddots & \\ & & & \frac{1}{\sigma_n^2} \end{bmatrix}$$

and Σ^{-1} is diagonal with diagonal entries the reciprocal of Σ's diagonal entries.

Since all $x_i \in \Sigma$ are independent, Σ is all zeroes except for the diagonal entries. Σ^{-1} is also like this.

$\log(P(X \mid \mu, \Sigma)) = -\frac{nd}{2}\log(2\pi) + \frac{n}{2}\log|\Sigma^{-1}| - \frac{1}{2}\sum_{i=1}^{n} tr((x_i-\mu)^T \Sigma^{-1}(x_i-\mu))$

$$\frac{\partial \log(P(X \mid \mu, \Sigma))}{\partial \Sigma^{-1}} = \frac{n}{2}\frac{1}{\Sigma^{-1}} - \frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T$$

Set equal to zero:

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

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(x_i-\hat{\mu})(x_i-\hat{\mu})^T$$

Applying properties of Σ from above, notice that $\hat{\sigma}_i$ is just the ith element of the diagonal of $\hat{\Sigma}$ (since it's zero outside of the diagonal.)

Thus, $\boxed{\hat{\sigma}_i = \frac{1}{n}(x_i-\hat{\mu})(x_i-\mu)^T}$

Figure 16: Proof for question 5a.

(b) Derive the maximum likelihood estimate for μ.
Solution is on the next page in the image.

(b) Known Σ, Σ is positive definite, $\Sigma \in M_{d \times d}(F)$

unknown mean $A\mu$, A is invertible, $A \in M_{d \times d}(F)$

$N(A\mu, \Sigma)$

$$P(x \mid A\mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(x - A\mu)^T \Sigma^{-1} (x - A\mu)\right)}$$

$x \in \mathbb{R}^d$

since Σ is positive definite, Σ has no zero eigenvalues, and thus Σ is invertible.

$$\log(P(x \mid A\mu, \Sigma)) = \log\left(\frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} e^{\left(-\frac{1}{2}(x - A\mu)^T \Sigma^{-1} (x - A\mu)\right)}\right)$$

$$\log(P(x \mid A\mu, \Sigma)) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(x - A\mu)^T \Sigma^{-1} (x - A\mu)$$

let $\underline{X} = \{x_1, \ldots, x_n\}$ $x_i \in \mathbb{R}^d$, $x_1, \ldots, x_n$ are iid

$$\log(P(\underline{X} \mid A\mu, \Sigma)) = -\frac{nd}{2}\log(2\pi) - \frac{n}{2}\log|\Sigma| - \frac{1}{2}\sum_{i=1}^{n}(x_i - A\mu)^T \Sigma^{-1} (x_i - A\mu)$$

$$\frac{\partial(\log P(\underline{X} \mid A\mu, \Sigma))}{\partial(A\mu)} = -\frac{1}{2}\sum_{i=1}^{n}\Sigma^{-1}(x_i - A\mu)$$

set equal to zero:

$$-\frac{1}{2}\sum_{i=1}^{n}\Sigma^{-1}(x_i - A\mu) = 0$$

$$\sum_{i=1}^{n}\Sigma^{-1}(x_i - A\mu) = 0$$

$$\sum_{i=1}^{n}\Sigma^{-1}x_i = \sum_{i=1}^{n}\Sigma^{-1}A\mu$$

since Σ is known, Σ^{-1} is known and both are fixed and constant.

$$\sum_{i=1}^{n}x_i = \sum_{i=1}^{n}A\mu$$

$$\sum_{i=1}^{n}x_i = nA\mu$$

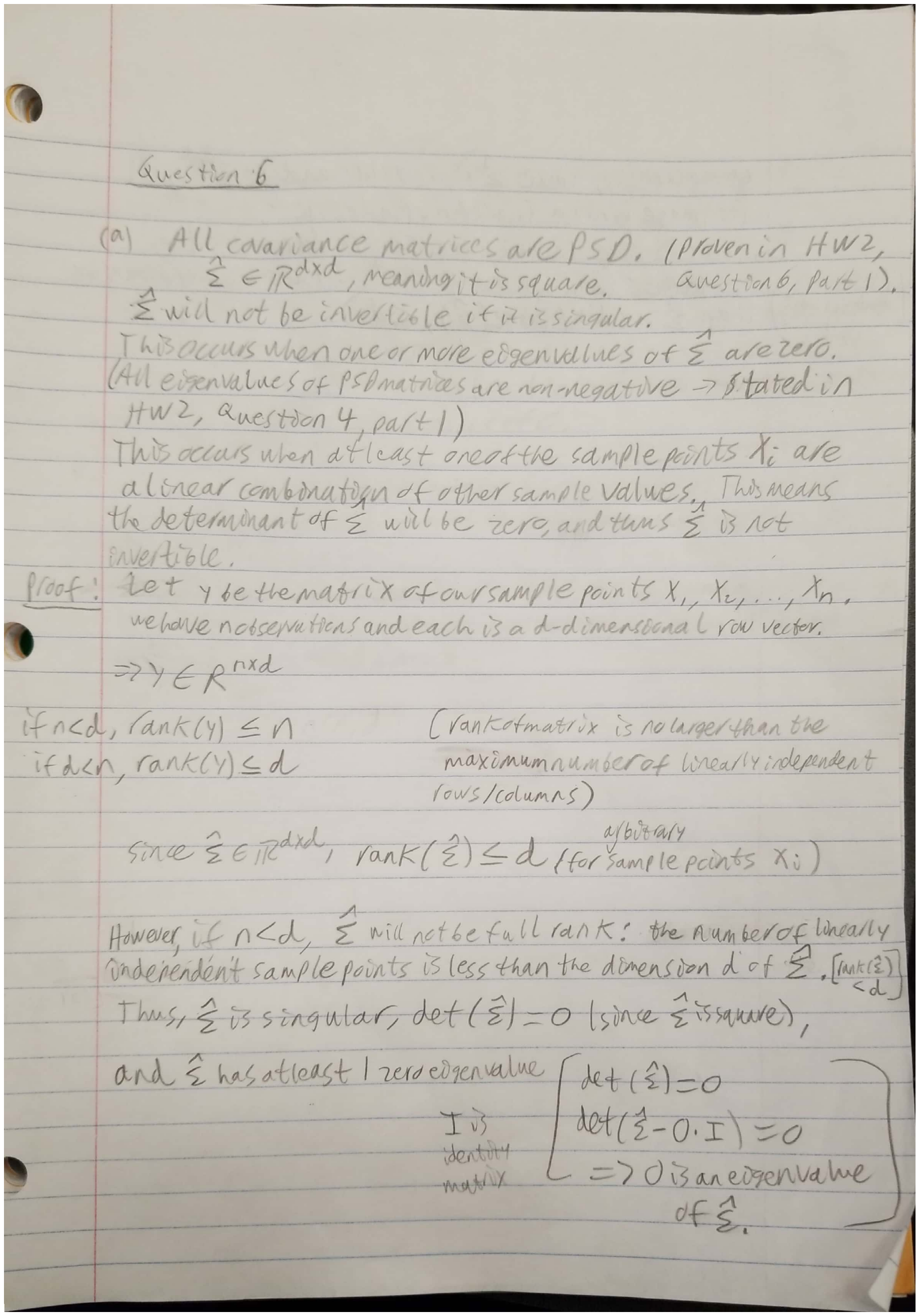
since A is invertible,

$$\boxed{\hat{\mu} = \frac{A^{-1}\sum_{i=1}^{n}x_i}{n}}$$

Figure 17: Proof for question 5b.

6 Covariance Matrices and Decompositions

(a) Under what circumstances is $\hat{\sum}$ not invertible?
Solution: On next 2 pages in the images.



Question 6

(a) All covariance matrices are PSD. (Proven in HW2, Question 6, part 1).
$\hat{\Sigma} \in \mathbb{R}^{d \times d}$, meaning it is square.
$\hat{\Sigma}$ will not be invertible if it is singular.
This occurs when one or more eigenvalues of $\hat{\Sigma}$ are zero.
(All eigenvalues of PSD matrices are non-negative → stated in HW2, Question 4, part 1)
This occurs when at least one of the sample points X_i are a linear combination of other sample values. This means the determinant of $\hat{\Sigma}$ will be zero, and thus $\hat{\Sigma}$ is not invertible.

Proof: Let Y be the matrix of our sample points $X_1, X_2, \dots, X_n$. We have n observations and each is a d-dimensional row vector.

$\Rightarrow Y \in R^{n \times d}$

if $n<d$, $\text{rank}(Y) \leq n$
if $d<n$, $\text{rank}(Y) \leq d$
(rank of matrix is no larger than the maximum number of linearly independent rows/columns)

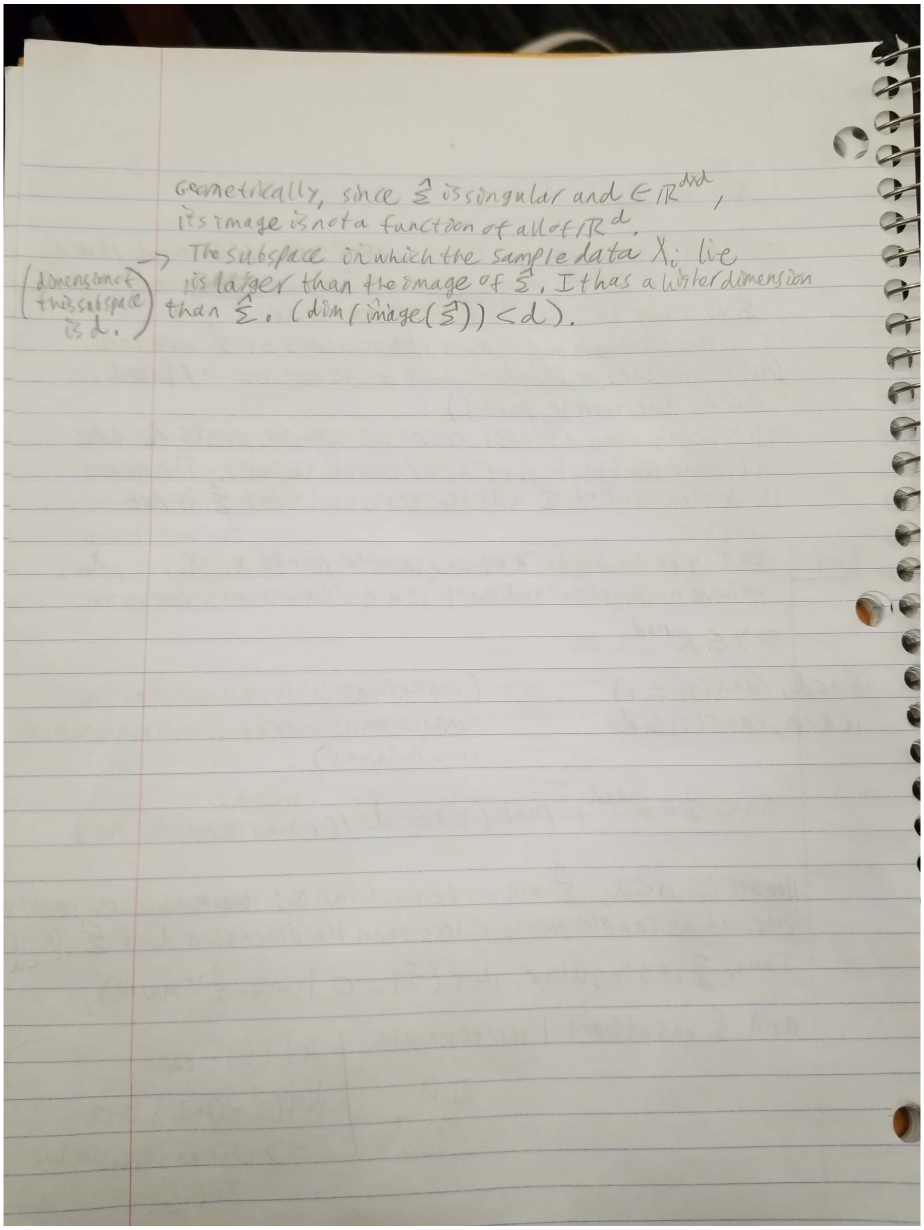
since $\hat{\Sigma} \in \mathbb{R}^{d \times d}$, $\text{rank}(\hat{\Sigma}) \leq d$ (for arbitrary sample points X_i)

However, if $n<d$, $\hat{\Sigma}$ will not be full rank: the number of linearly independent sample points is less than the dimension d of $\hat{\Sigma}$. [$\text{rank}(\hat{\Sigma}) < d$]
Thus, $\hat{\Sigma}$ is singular, $\det(\hat{\Sigma}) = 0$ (since $\hat{\Sigma}$ is square),
and $\hat{\Sigma}$ has at least 1 zero eigenvalue

I is identity matrix

$$\left[\begin{array}{l} \det(\hat{\Sigma}) = 0 \\ \det(\hat{\Sigma} - 0 \cdot I) = 0 \\ \Rightarrow 0 \text{ is an eigenvalue of } \hat{\Sigma}. \end{array}\right]$$

Figure 18: Proof for question 6a.



Geometrically, since $\hat{\Sigma}$ is singular and $\in \mathbb{R}^{d \times d}$, its image is not a function of all of $\mathbb{R}^d$.

→ The subspace in which the sample data X_i live is larger than the image of $\hat{\Sigma}$. It has a higher dimension than $\hat{\Sigma}$. ($\dim(\text{image}(\hat{\Sigma})) < d$).

(dimension of this subspace is d.)

Figure 19: Proof for question 6a continued.

(b) Suggest a way to fix a singular covariance matrix estimator $\hat{\sum}$ by replacing it with a similar but invertible matrix.
Solution: On the next page in the image.

(b) $\hat{\Sigma}$ is PSD and has non-negative eigenvalues (proven in HW2).

If $\hat{\Sigma}$ is singular, we can add a small value c to the diagonal elements of $\hat{\Sigma}$. Since $\hat{\Sigma}$ is PSD, this operation ensures that $\hat{\Sigma}$ no longer has zero eigenvalues, as all eigenvalues of the transformed matrix are adjusted upwards by c. The optimal value of c can be obtained through cross-validation using $\hat{\Sigma}$ and different values of c.

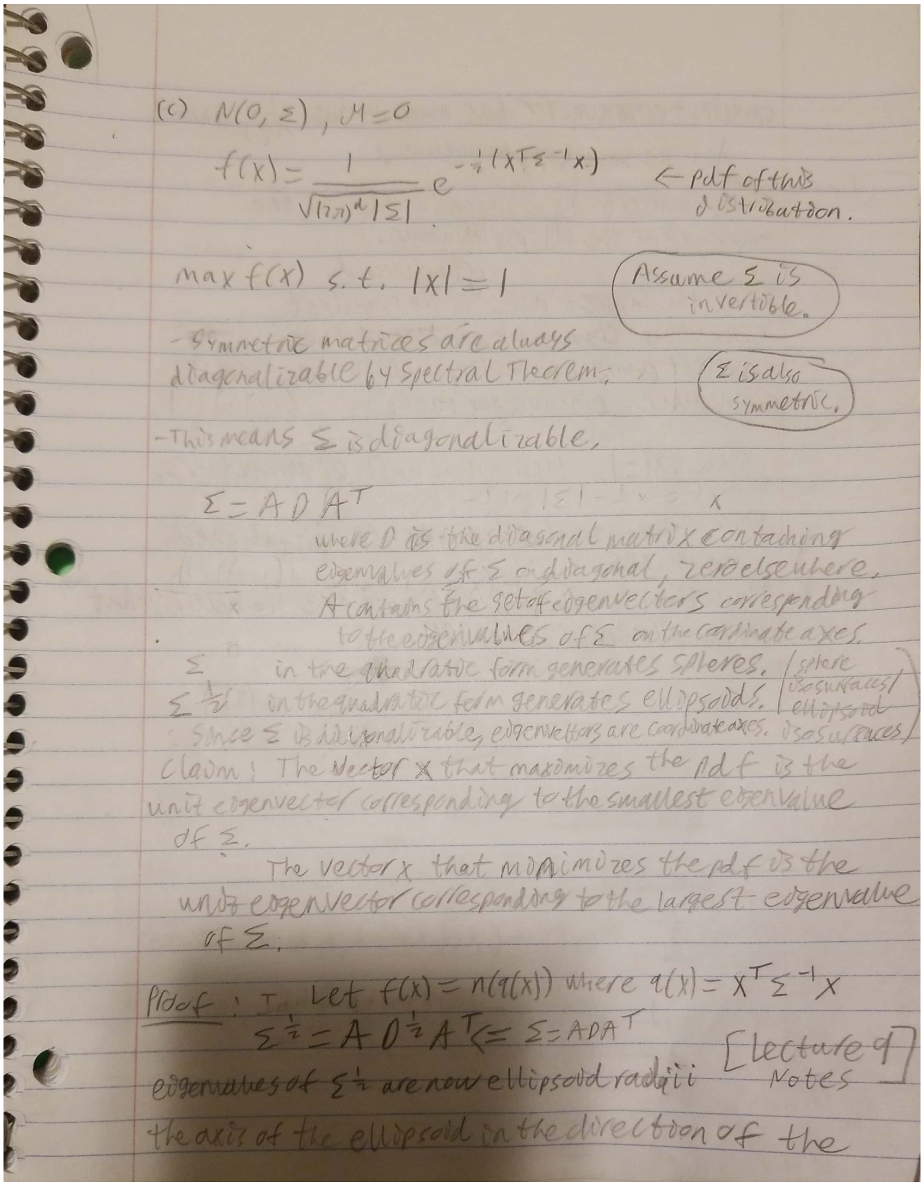
We do not want c to be too large because then the inverse of the transformed $\hat{\Sigma}$ will be further away from the true inverse/solution.

We don't want c to be too small either, as this could cause the matrix inversion to become unstable, resulting in weird values in the inverted matrix.

Figure 20: Proof for question 6b.

(c) Which vector(s) x of length 1 maximizes the PDF f(x)? Which vector(s) x of length 1 minimizes f(x)?

Solution: On the next 2 pages in the images.



(c) $N(0, \Sigma)$, $\mu = 0$

$$f(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(x^T \Sigma^{-1} x)}$$ ← pdf of this distribution.

$\max f(x)$ s.t. $|x| = 1$

Assume Σ is invertible.

- Symmetric matrices are always diagonalizable by Spectral Theorem.

Σ is also symmetric.

- This means Σ is diagonalizable.

$\Sigma = A D A^T$

where D is the diagonal matrix containing eigenvalues of Σ on diagonal, zero elsewhere. A contains the set of eigenvectors corresponding to the eigenvalues of Σ on the coordinate axes.

Σ in the quadratic form generates spheres. (sphere isosurfaces/ ellipsoid isosurfaces)

$\Sigma^{\frac{1}{2}}$ in the quadratic form generates ellipsoids.

Since Σ is diagonalizable, eigenvectors are coordinate axes.

Claim: The vector x that maximizes the pdf is the unit eigenvector corresponding to the smallest eigenvalue of Σ.

The vector x that minimizes the pdf is the unit eigenvector corresponding to the largest eigenvalue of Σ.

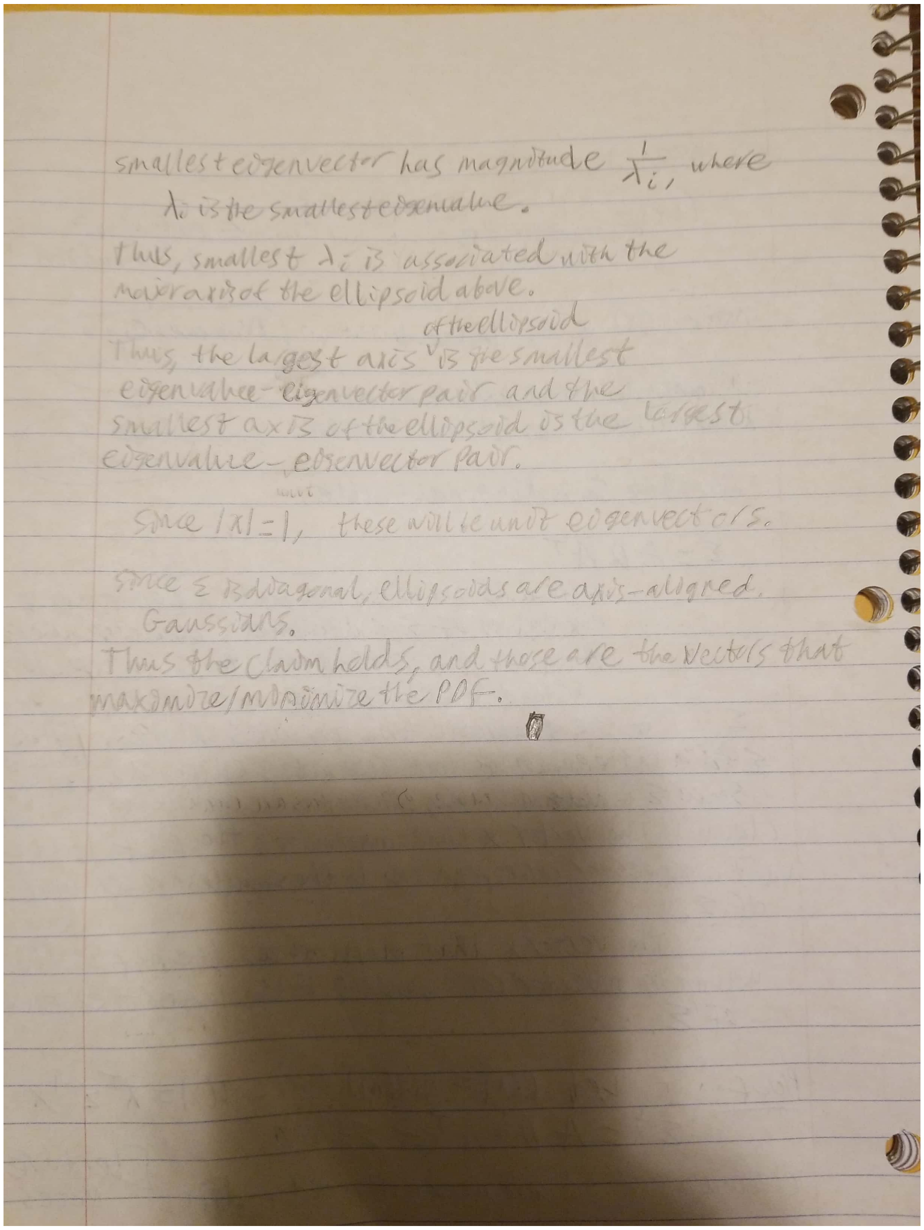
Proof: Let $f(x) = n(q(x))$ where $q(x) = x^T \Sigma^{-1} x$

$\Sigma^{\frac{1}{2}} = A D^{\frac{1}{2}} A^T \Leftarrow \Sigma = ADA^T$ [Lecture 9 Notes]

eigenvalues of $\Sigma^{\frac{1}{2}}$ are now ellipsoid radii

the axis of the ellipsoid in the direction of the

Figure 21: Proof for question 6c.



smallest eigenvector has magnitude $\frac{1}{\lambda_i}$, where λ_i is the smallest eigenvalue.

Thus, smallest λ_i is associated with the major axis of the ellipsoid above.

Thus, the largest axis of the ellipsoid is the smallest eigenvalue-eigenvector pair and the smallest axis of the ellipsoid is the largest eigenvalue-eigenvector pair.

Since $|x|=1$, these will be unit eigenvectors.

Since Σ is diagonal, ellipsoids are axis-aligned Gaussians.

Thus the claim holds, and those are the vectors that maximize/minimize the PDF.

Figure 22: Proof for question 6c continued.

7 Gaussian Classifiers for Digits and Spam

(a) Taking pixel values as features (no new features yet, please), fit a Gaussian distribution to each digit class using maximum likelihood estimation.
Solution:

```
mtrain = mnist_data['training_data']
#part (a) asks us to fit a Gaussian distribution to each digit class.
maverage = np.mean(mtrain)
mtrain = mtrain - maverage
contrast = np.sqrt(10 + np.mean(mtrain**2))
mtrain = mtrain / max(contrast, 0.000000001)
#attempt at contrast normalization before using pixel values.
#external source used:
#https://datascience.stackexchange.com/questions/15110/how-to-implement-global-c
#we will compute a mean and covariance matrix for each digit class
uniques = np.unique(mnist_data['training_labels'])
indexlst = []
for i in uniques:
    templst = [x for x in range(len(mnist_data['training_labels'])) if mnist_dat
    indexlst.append(templst)
#building a list of lists of indices to map the training points to their corresp
from scipy.stats import multivariate_normal
matrix_of_cov_matrices = []
matrix_of_means = []
gaussians = []
for i in indexlst:
    mcov = np.cov(mtrain[i], bias = True, rowvar = False)
    for j in range(len(mcov)):
        mcov[j][j] += 0.1
    #computes the covariance matrix for each digit class.
    #uses the trick from question 6b to solve the singular covariance matrix pro
    m_mean = np.mean(mtrain[i], axis = 0)
    matrix_of_cov_matrices.append(mcov)
    matrix_of_means.append(m_mean)
    #computes the mean for each digit class.
    digit_gaussian = multivariate_normal(m_mean, mcov)
    gaussians.append(digit_gaussian)
    #fitting the gaussian distribution to each digit class.
```

(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: It appears that the correlation coefficient matrix of the covariance matrix for the class 0 shows that the diagonal terms are equal to each other and that the diagonal terms are larger than the off-diagonal terms. I conclude from this that the diagonal entries are the correlation of a given feature with

itself (hence why it's 1), and that the correlation between different features is generally strong and negative, but appears to increase as you move down and right in the matrix.

(c) (1) Linear discriminant analysis (LDA). Model the class conditional probabilities as Gaussians $N(\mu_C, \sum)$ with different means μ_C (for class C) and the same covariance matrix $\sum$, which you compute by averaging the 10 covariance matrices from the 10 classes.

Solution: After implementing the trick in 6(b) to avoid singular covariance matrices, and implementing contrast-normalization (both in 7a), my approach to this problem was to utilize scipy's multivariatenormal logpdf function to classify each point using the gaussian for each class with a class mean and the average covariance matrix. As the following figure shows, the strategy did not work well for the validation set, but worked surprisingly well on the training set, as the error rates are markedly lower for the training sets as compared to the validation set.

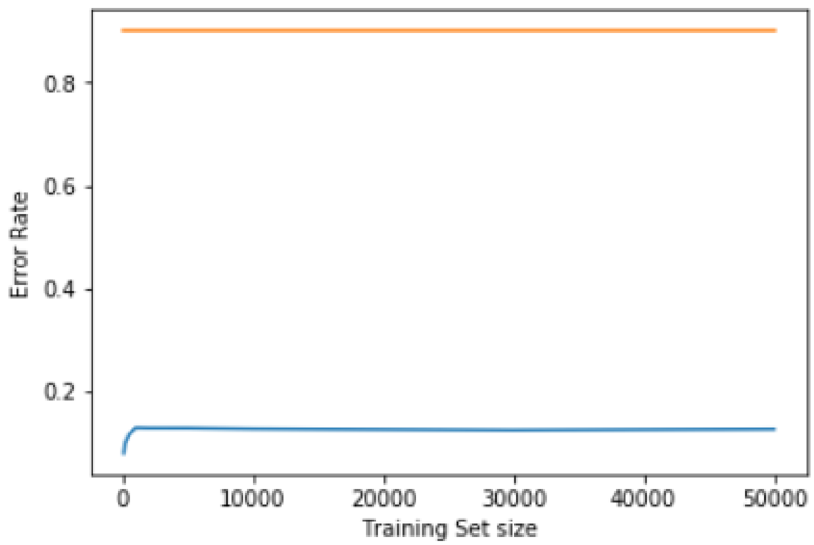


Figure 23: Graph for LDA for MNIST. Orange is validation error, and blue is training error.

```
avg_cov = np.zeros((784, 784))
for i in range(len(matrix_of_cov_matrices)):
    avg_cov = np.add(avg_cov, matrix_of_cov_matrices[i])
#finding the average covariance matrix across the classes.
avg_cov = avg_cov/10
indices_to_choose = np.arange(len(mtrain))
valid_indices = np.random.choice(indices_to_choose, 10000, replace = False)
mvalid = mtrain[valid_indices]
#creating the validation set of 10,000 randomly selected points
t_indices = np.arange(60000)
def diff(first, second):
    second = set(second)
    return [item for item in first if item not in second]
valid_indices = diff(t_indices, valid_indices)
#creating a list of valid indices in the training set (that are not in the valid
def loss_fn(labels1, labels2):
    total_loss = 0
    for i in range(len(labels1)):
        if labels1[i] != labels2[i]:
            total_loss += 1
    return total_loss
avg_digit_gaussians = []
```

```
for mean in matrix_of_means:
    avg_digit_gaussians.append(multivariate_normal(mean, avg_cov))
#for lda, we keep covariance the same across classes——we thus compute new mul
sizes = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
training_losses = []
valid_losses = []
#training set sizes
for size in sizes:
    training_indices = np.random.choice(valid_indices, size, replace = False)
    temp_training_set = mtrain[training_indices]
    #getting the training set of size size.
    predicted_labels = []
    for point in temp_training_set:
        temp_class_densities = []
        for gaussian in avg_digit_gaussians:
            temp_class_densities.append(gaussian.logpdf(point))
        maximum = max(temp_class_densities)
        predicted_labels.append(temp_class_densities.index(maximum))
        #the assigned label is the class associated with the maximum density
    true_labels = mnist_data['training_labels'][training_indices]
    loss = loss_fn(predicted_labels, true_labels)
    training_losses.append(loss)
    predicted_vlabels = []
    for point1 in mvalid:
        temp_valid_class_densities = []
        for gaussian in avg_digit_gaussians:
            temp_valid_class_densities.append(gaussian.logpdf(point1))
        maximum = max(temp_valid_class_densities)
        predicted_vlabels.append(temp_valid_class_densities.index(maximum))
    true_vlabels = mnist_data['training_labels'][valid_indices]
    vloss = loss_fn(predicted_vlabels, true_vlabels)
    valid_losses.append(vloss)
#computing the validation set errors and
#computing the training set errors.
terror_rates = [training_losses[i]/sizes[i] for i in range(len(training_losses))
verror_rates = [valid_losses[i]/10000 for i in range(len(valid_losses))]
plt.plot(sizes, terror_rates)
plt.plot(sizes, verror_rates)
plt.ylabel('Error Rate')
plt.xlabel('Training Set size')
#plotting the error rates against training_set size.
```

(2) Quadratic discriminant analysis (QDA). Model the class conditionals as Gaussians $N(\mu_C, \sum_C)$, where $\sum_C$ 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.
Solution: For QDA, I allowed different covariance matrices for the different classes. I followed roughly the same procedures otherwise as I did with LDA. In this case, training errors were lower for all training sizes, but validation errors were roughly the same.

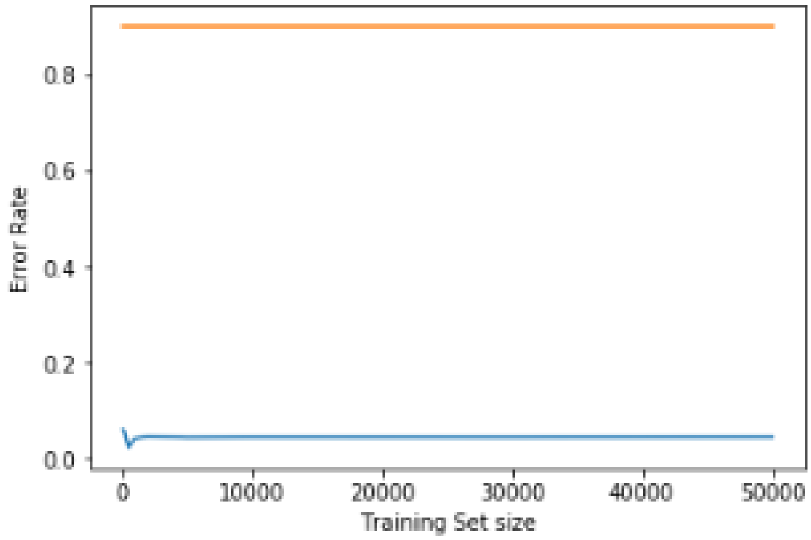


Figure 24: Graph for QDA for MNIST. Orange is validation error, and blue is training error.

```
sizes = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000]
qtraining_losses = []
qvalid_losses = []
#training set sizes
for size in sizes:
    training_indices = np.random.choice(valid_indices, size, replace = False)
    temp_training_set = mtrain[training_indices]
    #getting the training set of size size.
    predicted_labels = []
    for point in temp_training_set:
        temp_class_densities = []
        for gaussian in gaussians:
            temp_class_densities.append(gaussian.logpdf(point))
        maximum = max(temp_class_densities)
        predicted_labels.append(temp_class_densities.index(maximum))
        #the assigned label is the class associated with the maximum density
    true_labels = mnist_data['training_labels'][training_indices]
    loss = loss_fn(predicted_labels, true_labels)
    qtraining_losses.append(loss)
    predicted_vlabels = []
    for point1 in mvalid:
        temp_valid_class_densities = []
```

```
        for gaussian in gaussians:
            temp_valid_class_densities.append(gaussian.logpdf(point1))
        maximum = max(temp_valid_class_densities)
        predicted_vlabels.append(temp_valid_class_densities.index(maximum))
    true_vlabels = mnist_data['training_labels'][valid_indices]
    vloss = loss_fn(predicted_vlabels, true_vlabels)
    qvalid_losses.append(vloss)
#computing the validation set errors and
#computing the training set errors.
qterror_rates = [qtraining_losses[i]/sizes[i] for i in range(len(qtraining_losse
qverror_rates = [qvalid_losses[i]/10000 for i in range(len(qvalid_losses))]
plt.plot(sizes, qterror_rates)
plt.plot(sizes, qverror_rates)
plt.ylabel('Error Rate')
plt.xlabel('Training Set size')
#plotting the error rates against training_set size.
```

(3) (Written answer.) Which of LDA and QDA performed better? Why?
Solution: QDA performed better because it allows for different covariance matrices across the different classes. Thus, it captures more of the variation between the classes and in the data and has greater explanatory power in solving the multi-class classification problem.

(4) Kaggle Submission:
I did not use any new features in my Kaggle submission.
My screen name on Kaggle is: Vinay Maruri (email: vmaruri1@berkeley.edu)
My score was: 0.89400 (position 229)

```
#part(4) kaggle submission
#choosing to use QDA since lower error rates in training.
from save_csv import results_to_csv
mtest = mnist_data['test_data']
kaggle_labels = []
for point in mtest:
    kaggle_temp_class_densities = []
    for gaussian in gaussians:
        kaggle_temp_class_densities.append(gaussian.logpdf(point))
    maximum = max(kaggle_temp_class_densities)
    kaggle_labels.append(kaggle_temp_class_densities.index(maximum))
results_to_csv(np.array(kaggle_labels))
```

(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.
Solution: I used LDA for spam given that it is a 2-class classification problem and a linear decision boundary would seem to work better in this case. I re-used the same approach to LDA as I did in part (c) (1) of this question. I did not omit/add any features. My training/validation errors seemed appropriate.
My Kaggle screen-name: Vinay Maruri
My Kaggle score for this competition was: 0.78884 (position 345).

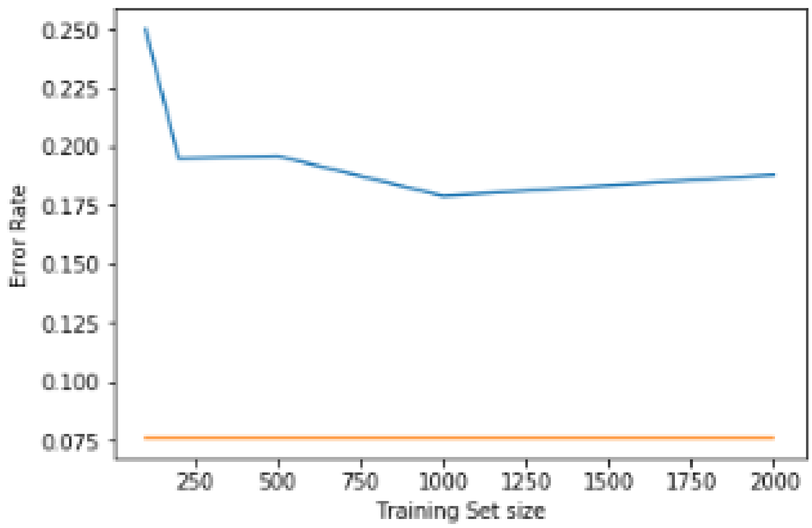


Figure 25: Graph for LDA for Spam. Orange is validation error, and blue is training error.

```
#part (d): I will use LDA for spam.
spam_train = spam_data['training_data']
spam_labels = spam_data['training_labels']
spam_test = spam_data['test_data']

uniques = np.unique(spam_labels)
indexlst = []
for i in uniques:
    templst = [x for x in range(len(spam_labels)) if spam_labels[x] == i]
    indexlst.append(templst)
#building a list of lists of indices to map the training points to their corresp
smatrix_of_cov_matrices = []
smatrix_of_means = []
sgaussians = []

for i in indexlst:
    scov = np.cov(spam_train[i], bias = True, rowvar = False)
    for j in range(len(scov)):
        scov[j][j] += 0.1
    #computes the covariance matrix for each digit class.
    #uses the trick from question 6b to solve the singular covariance matrix pro
    s_mean = np.mean(spam_train[i], axis = 0)
    smatrix_of_cov_matrices.append(scov)
```

```
    smatrix_of_means.append(s_mean)
    #computes the mean for each digit class.
    digit_gaussian = multivariate_normal(s_mean, scov)
    sgaussians.append(digit_gaussian)
    #fitting the gaussian distribution to each digit class.
avg_cov = np.zeros((32, 32))
for i in range(len(smatrix_of_cov_matrices)):
    avg_cov = np.add(avg_cov, smatrix_of_cov_matrices[i])
#finding the average covariance matrix across the classes.
avg_cov = avg_cov/2
spam_gaussians = []
for mean in smatrix_of_means:
    spam_gaussians.append(multivariate_normal(mean, avg_cov))
#for lda, we keep covariance the same across classes---we thus compute new mul
sgaussians = spam_gaussians
sizes = [100, 200, 500, 1000, 2000]
qtraining_losses = []
qvalid_losses = []
indices_to_choose = np.arange(len(spam_train))
valid_indices = np.random.choice(indices_to_choose, 1000, replace = False)
spam_valid = spam_train[valid_indices]
#creating the validation set of 10,000 randomly selected points
t_indices = np.arange(5172)
valid_indices = diff(t_indices, valid_indices)
#training set sizes
for size in sizes:
    training_indices = np.random.choice(valid_indices, size, replace = False)
    temp_training_set = spam_train[training_indices]
    #getting the training set of size size.
    predicted_labels = []
    for point in temp_training_set:
        temp_class_densities = []
        for gaussian in sgaussians:
            temp_class_densities.append(gaussian.logpdf(point))
        maximum = max(temp_class_densities)
        predicted_labels.append(temp_class_densities.index(maximum))
        #the assigned label is the class associated with the maximum density
    true_labels = spam_labels[training_indices]
    loss = loss_fn(predicted_labels, true_labels)
    qtraining_losses.append(loss)
    predicted_vlabels = []
    for point1 in spam_valid:
        temp_valid_class_densities = []
        for gaussian in sgaussians:
            temp_valid_class_densities.append(gaussian.logpdf(point1))
        maximum = max(temp_valid_class_densities)
```

```
            predicted_vlabels.append(temp_valid_class_densities.index(maximum))
        true_vlabels = spam_labels[valid_indices]
        vloss = loss_fn(predicted_vlabels, true_vlabels)
        qvalid_losses.append(vloss)

    qterror_rates = [qtraining_losses[i]/sizes[i] for i in range(len(qtraining_losse
    qverror_rates = [qvalid_losses[i]/10000 for i in range(len(qvalid_losses))]
    plt.plot(sizes, qterror_rates)
    plt.plot(sizes, qverror_rates)
    plt.ylabel('Error Rate')
    plt.xlabel('Training Set size')
    #plotting the error rates against training_set size.
    kaggle_labels = []
    for point in spam_test:
        kaggle_temp_class_densities = []
        for gaussian in sgaussians:
            kaggle_temp_class_densities.append(gaussian.logpdf(point))
        maximum = max(kaggle_temp_class_densities)
        kaggle_labels.append(kaggle_temp_class_densities.index(maximum))
    results_to_csv(np.array(kaggle_labels))
```