CS 289A – Spring 2023 – Homework 3

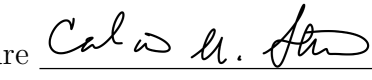
Colin Skinner, SID [redacted]

1 Honor Code

add any collaborations

"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."

Signed Colin Skinner

Signature  Date 2/22/2023

2 Gaussian Classification

2.1

When the posterior probabilities are equal you have

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

Which is the same as

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

But $P(Y = 1) = P(Y = -1)$ so

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

Since $\mu_2 > \mu_1$ we have

$$\int_{-\infty}^{b} f_{x_1}(x)dx = \int_{b}^{\infty} f_{x_2}(x)dx$$

$$\frac{d}{dx}\int_{-\infty}^{b} f_{x_1}(x)dx = \frac{d}{dx}\int_{b}^{\infty} f_{x_2}(x)dx$$

$$f_{x_1}(b) = f_{x_2}(b)$$

by the Fundamental Theorem of Calculus and because the tail probabilities approach zero

$$\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(b-\mu_1)^2}{2\sigma^2}\right) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(b-\mu_2)^2}{2\sigma^2}\right)$$

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

$$b^2 - 2b\mu_1 + \mu_1^2 = b^2 - 2b\mu_2 + \mu_2^2$$

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

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

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

The decision rule is then

$$\boxed{r^*(x) = \begin{cases} C_1 & \text{if } x < \frac{\mu_2+\mu_1}{2} \\ C_2 & \text{otherwise} \end{cases}}$$

2.2

$$P_e(b) = P(r^*(x) = C_2|C_1)P(C_1) + P(r^*(x) = C_1|C_2)P(C_2)$$

$$= \frac{1}{2}(P(r^*(x) = C_2|C_1) + P(r^*(x) = C_1|C_2))$$

$$= \frac{1}{2}\left(\int_{-\infty}^{b} f_{X_2}(x)dx + \int_{b}^{\infty} f_{X_1}(x)dx\right)$$

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

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

Q.E.D.

2.3

Since $P_e(b)$ is convex on $\mu_1 < b < \mu_2$.

$$\frac{dP_e(b^*)}{dx} = 0 \implies b^* \text{ is a minimum.}$$

Therefore

$$\frac{d}{dx}\left[\frac{1}{2\sqrt{2\pi}\sigma}\left(\int_{-\infty}^{b^*}\exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right)dx + \int_{b^*}^{\infty}\exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right)dx\right)\right] = 0$$

$$\frac{d}{dx}\int_{-\infty}^{b^*}\exp\left(-\frac{(x-\mu_2)^2}{2\sigma^2}\right)dx + \frac{d}{dx}\int_{b^*}^{\infty}\exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right)dx = 0$$

As in part 1, use Fundamental Theorem of Calculus; function values are zeros at the tails so we get

$$\exp\left(-\frac{(b^*-\mu_2)^2}{2\sigma^2}\right) + \exp\left(-\frac{(b^*-\mu_1)^2}{2\sigma^2}\right) = 0$$

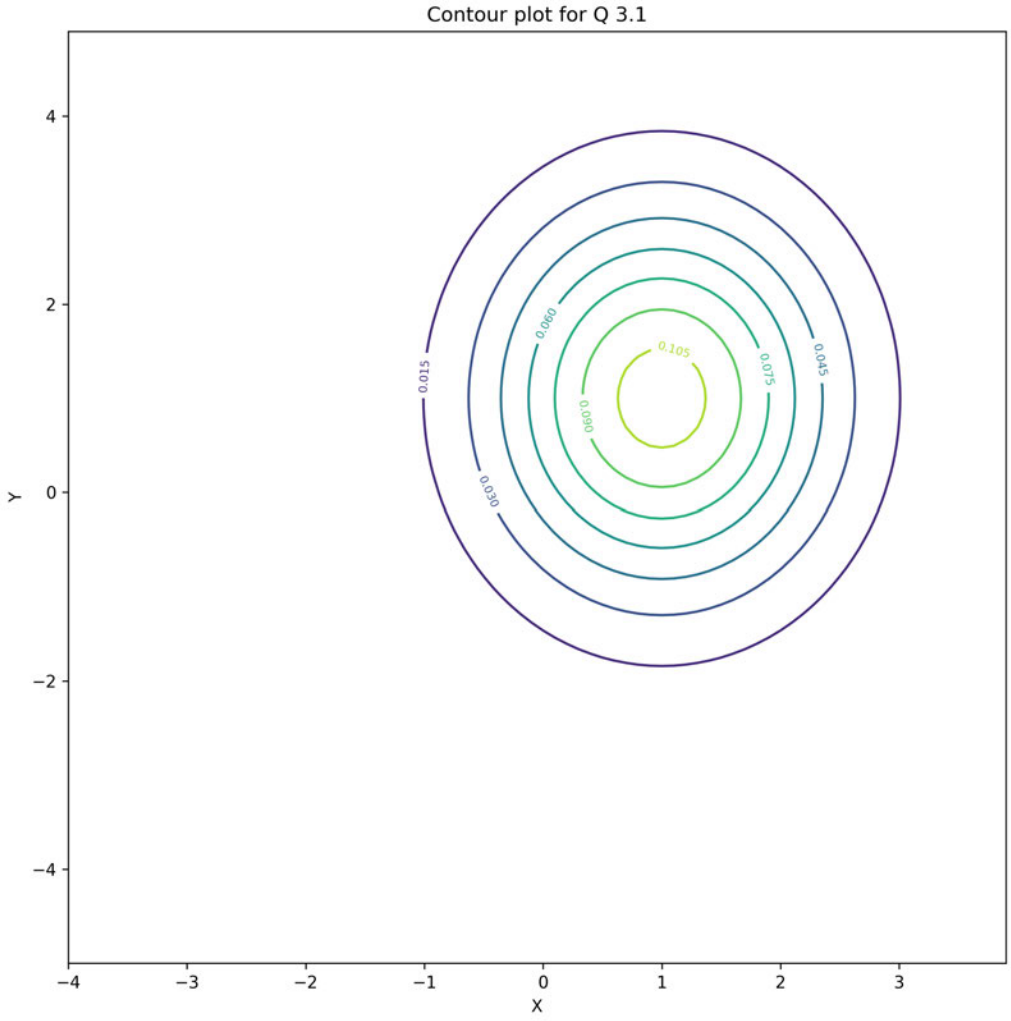
As in part 1, a bit of simple algebra gives

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

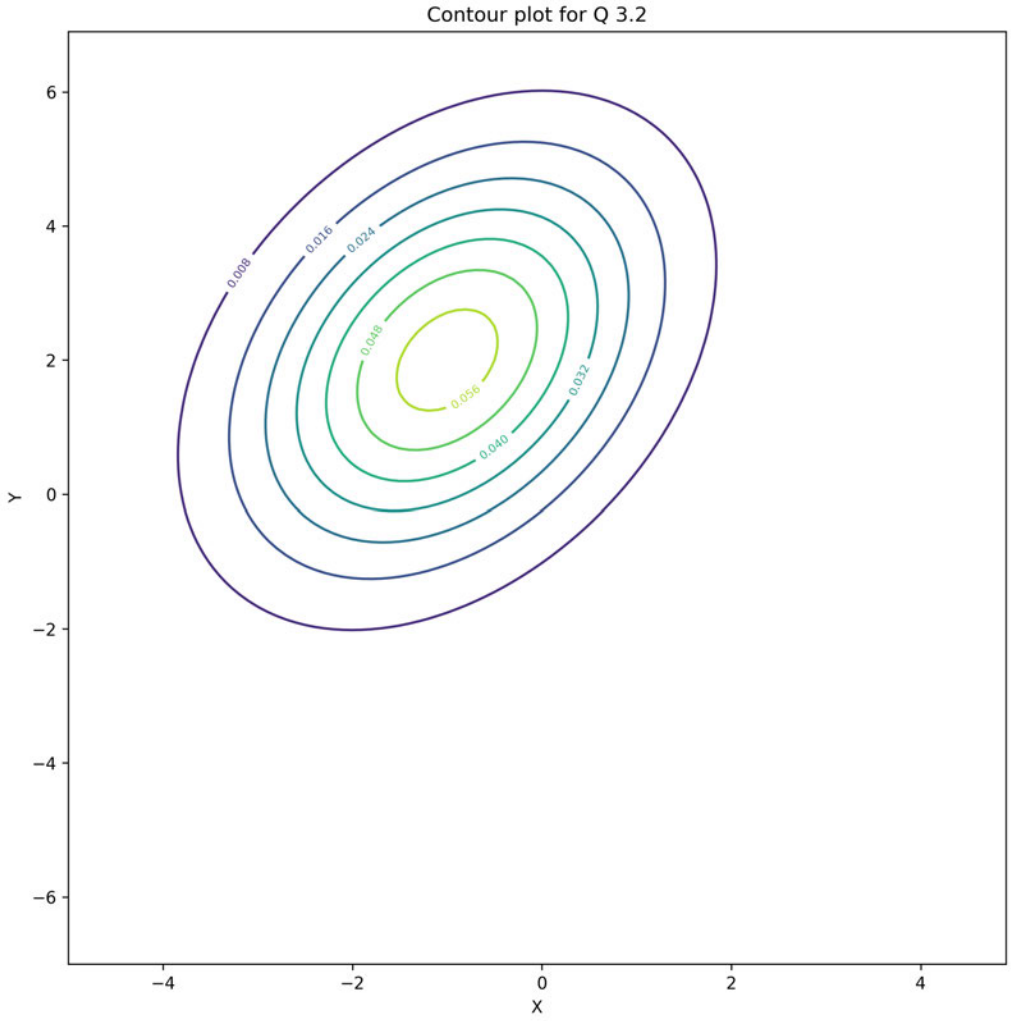
This is the same solution we got for part 1 by using the Baye's decision rule.

3 Isocontours of Normal Distributions

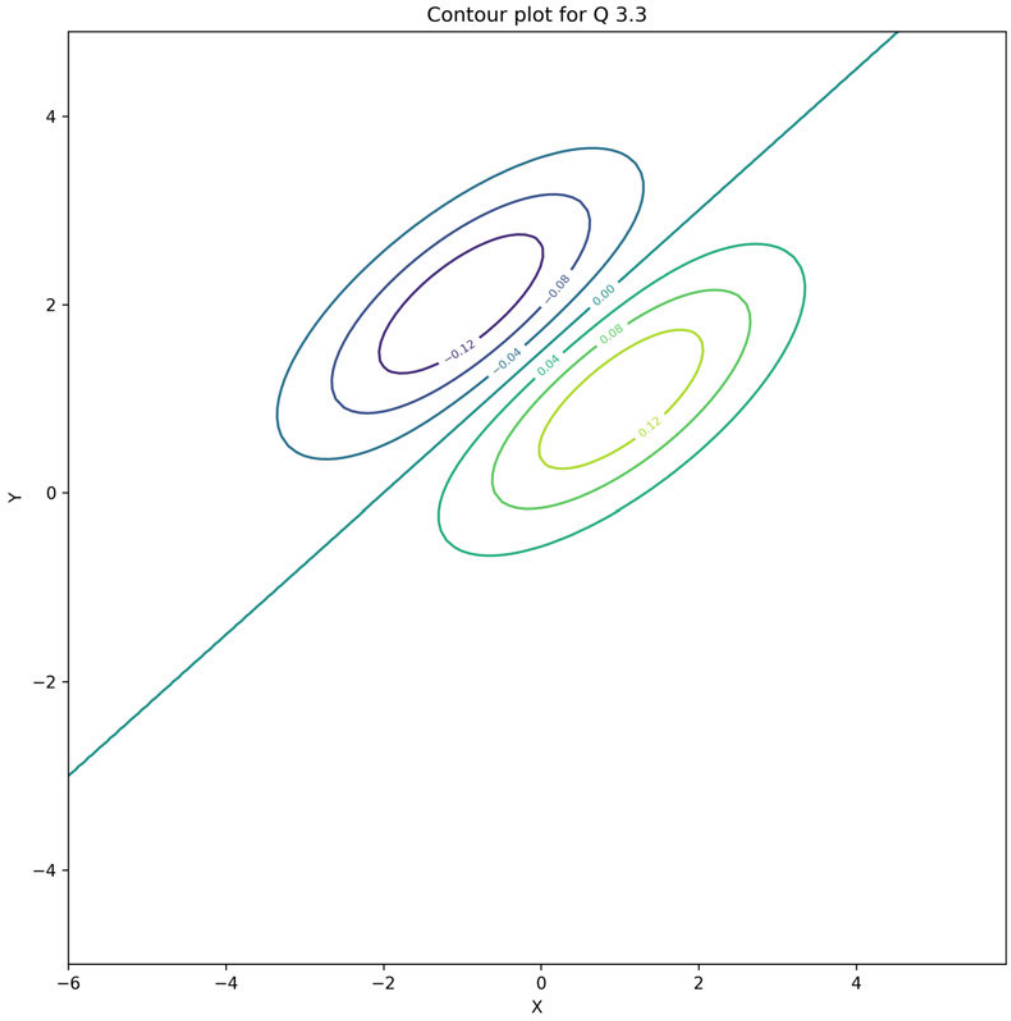
3.1



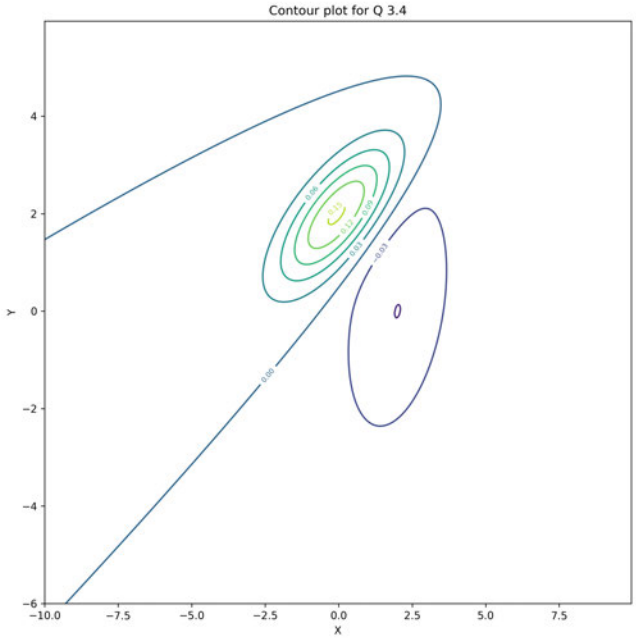
3.2


Contour plot for Q 3.2
0.008
0.016
0.024
0.048
0.056
0.032
0.040
6
4
2
0
−2
−4
−6
Y
−4
−2
0
2
4
X

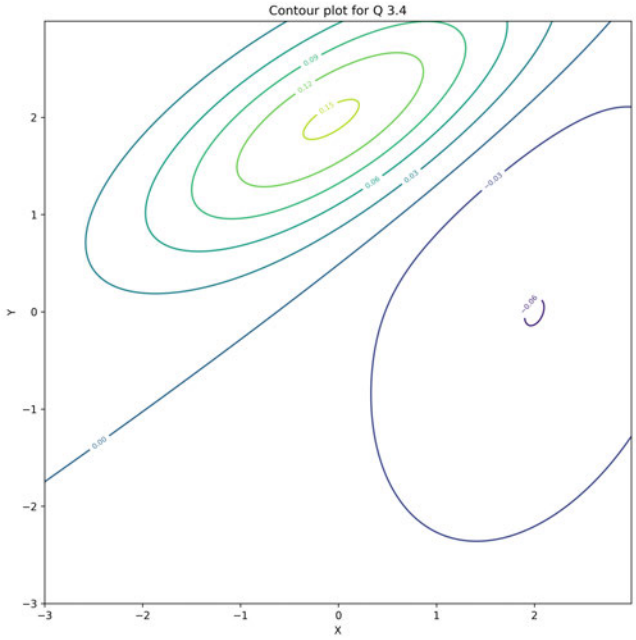
3.3


Contour plot for Q 3.3
−0.12
−0.08
−0.04
0.00
0.04
0.08
0.12
4
2
0
−2
−4
Y
−6
−4
−2
0
2
4
X

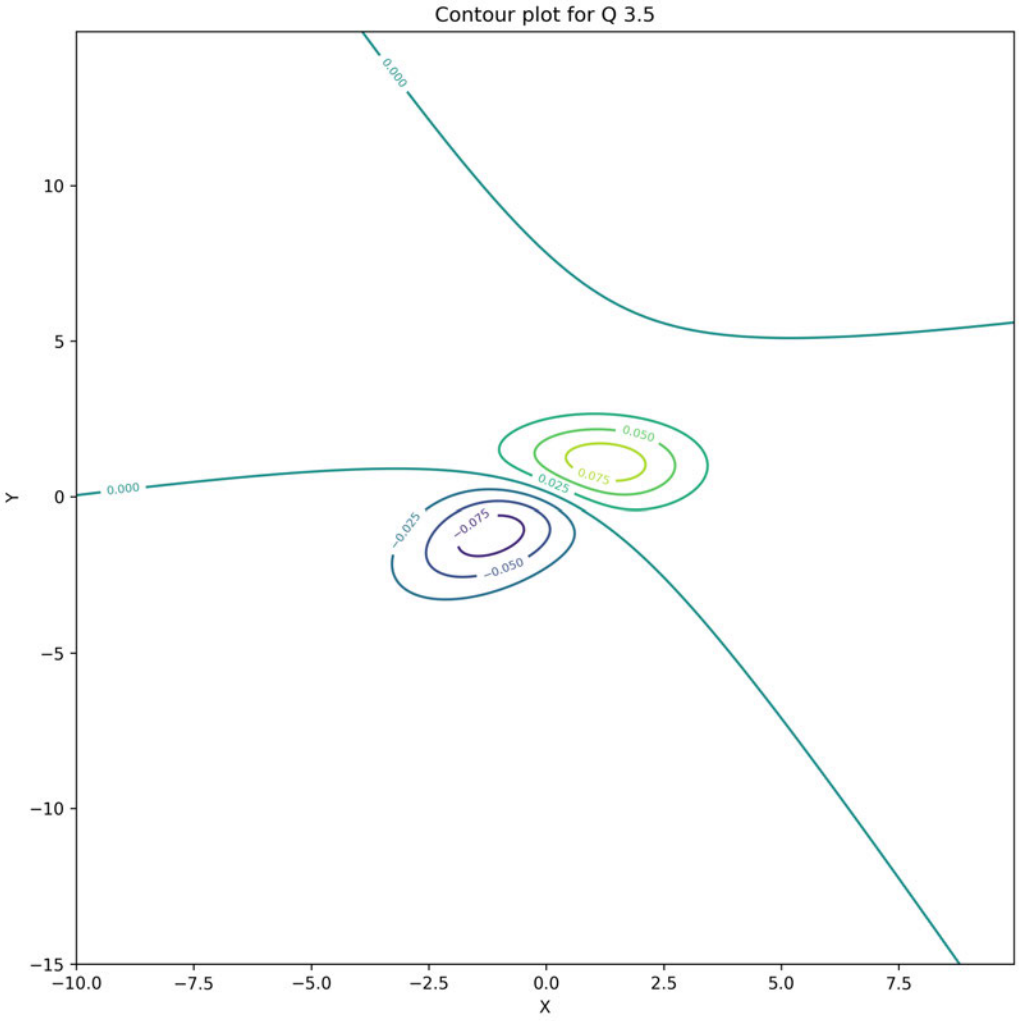
3.4



*The deep purple unlabeled contour has isovalue $z = -0.06$



3.5


Contour plot for Q 3.5
0.000
0.050
0.075
0.025
0.000
−0.025
−0.075
−0.050
10
5
0
−5
−10
−15
Y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
X

4 Eigenvectors of the Gaussian Covariance Matrix

For all of question 4 I used the np.random.normal() function to generate the random Gaussian values. I used np.random.seed() to set a seed for the random number generator, with 42 as the seed.

4.1

With my random seed and for n=100, in $\mathbb{R}^2$ the mean $\left(\overline{X_1}, \overline{X_2}\right)$ (rounded to four decimal places) is

$$\boxed{\left(\overline{X_1}, \overline{X_2}\right) = (2.6533, 5.3947)}$$

4.2

The (rounded) covariance matrix Σ_S of the sample is

$$\boxed{\Sigma_S = \begin{bmatrix} 6.5995 & 3.4656 \\ 3.4656 & 5.8066 \end{bmatrix}}$$

4.3

The eigenvectors u, v are

$$u = \begin{bmatrix} 0.7462 \\ 0.6657 \end{bmatrix}$$

and

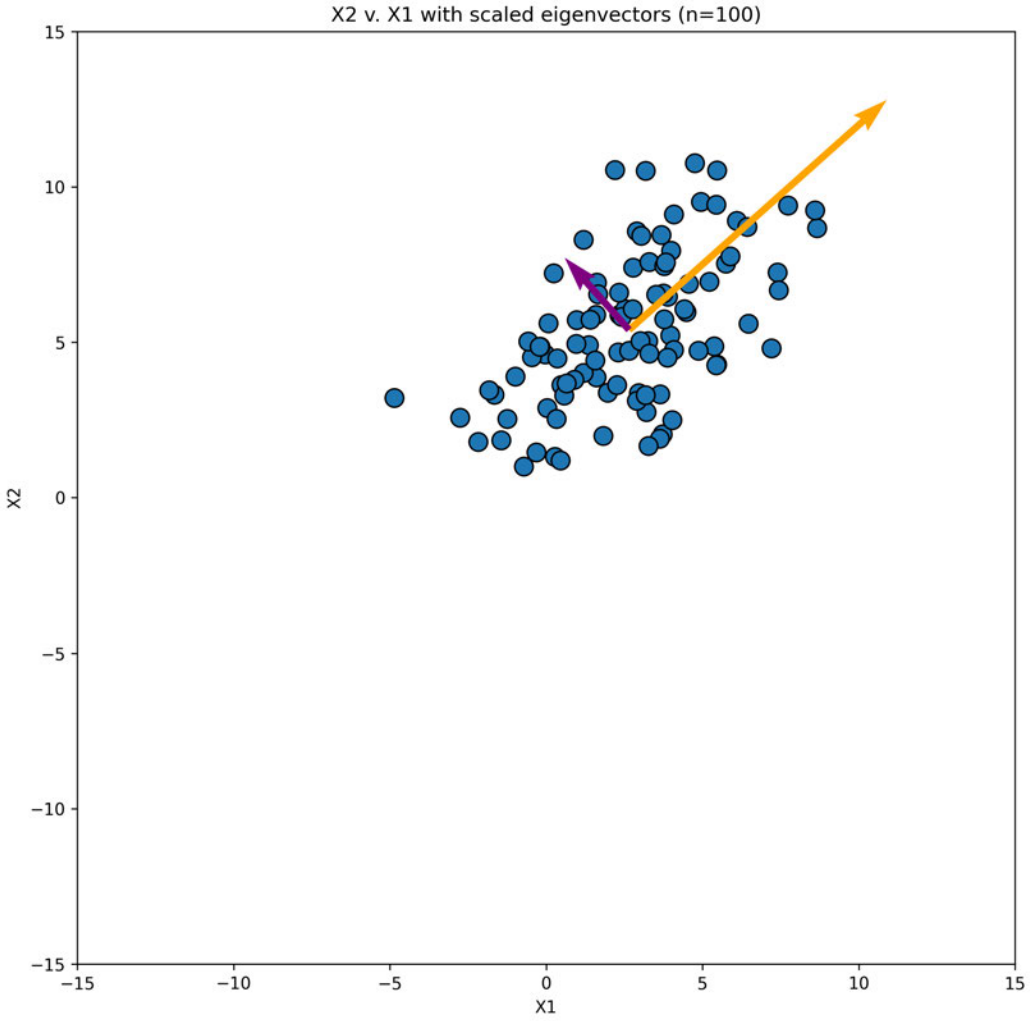
$$v = \begin{bmatrix} -0.6657 \\ 0.7462 \end{bmatrix}$$

With corresponding eigenvalues λ_1, λ_2

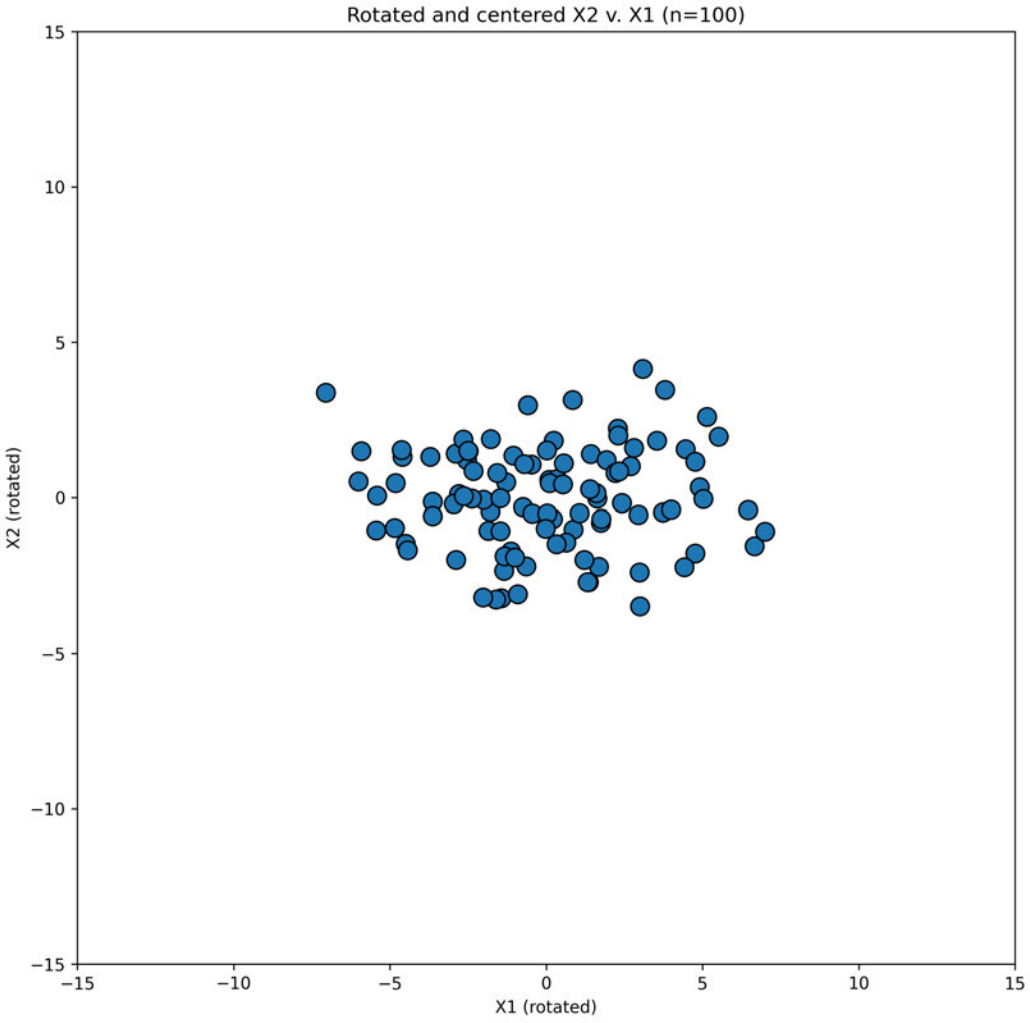
$$\lambda_1 = 9.6912$$

$$\lambda_2 = 2.7148$$

4.4


X2 v. X1 with scaled eigenvectors (n=100)
X2
X1
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15

4.5


Rotated and centered X2 v. X1 (n=100)
X2 (rotated)
X1 (rotated)
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15

5 Classification and Risk

5.1

First, check the risk for choosing a class $i \in \{1, ..., c\}$:

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

$$= L(r(x) = i, y = 1)P(y = 1|x) + ... + L(r(x) = i, y = i)P(y = i|x) + ... + L(r(x) = i, y = c)P(y = c|x)$$

$$= \lambda_s P(y = 1|x) + ... + 0 * P(y = i|x) + ... + \lambda_s P(y = c|x)$$

$$= \lambda_s \sum_{j \neq i}^{c} P(y = j|x)$$

Because the sum of all probabilities is 1

$$= \lambda_s (1 - P(y = i|x))$$

Next, we consider the risk of choosing "doubt"

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

$$= \sum_{j=1}^{c} \lambda_r P(y = j|x)$$

$$= \lambda_r \sum_{j=1}^{c} P(y = j|x)$$

$$= \lambda_r$$

Assuming $\lambda_r \leq \lambda_s$, to minimize the risk of classifying a data point, we can either always choose doubt if $\lambda_r \leq \lambda_s(1 - P(y = i|x))$, or we can find a rule for choosing a class $i \in \{1, ..., c\}$ such that it is guaranteed that

$$\lambda_s(1 - P(y = i|x)) \leq \lambda_r$$

$$1 - P(y = i|x) \leq \frac{\lambda_r}{\lambda_s}$$

$$P(y = i|x) \geq 1 - \frac{\lambda_r}{\lambda_s}$$

With this condition, we see that if we classify a point as class i when $P(y = i|x) \geq 1 - \frac{\lambda_r}{\lambda_s}$

$$\begin{aligned}\lambda_s(1 - P(y = i|x)) &\leq \lambda_s \left(1 - \left(1 - \frac{\lambda_r}{\lambda_s}\right)\right) \\ &= \lambda_s \left(\frac{\lambda_r}{\lambda_s}\right) \\ &= \lambda_r\end{aligned}$$

Meaning the risk for choosing that class is at most the same as the risk for choosing "doubt". Since it is preferable for a classifier to make classifications over not making classifications, we would choose to classify the point as being in class i in this scenario.

In the case where class i has the highest posterior, but $P(y = i|x) < 1 - \frac{\lambda_r}{\lambda_s}$

$$\begin{aligned}\lambda_s(1 - P(y = i|x)) &> \lambda_s \left(1 - \left(1 - \frac{\lambda_r}{\lambda_s}\right)\right) \\ &= \lambda_s \left(\frac{\lambda_r}{\lambda_s}\right) \\ &= \lambda_r\end{aligned}$$

and we would have lower risk choosing "doubt" instead of classifying the point as class i. Therefore, the decision rule that minimizes risk is the one provided in homework problem.

5.2

If $\lambda_r = 0$ then there would be no risk in choosing "doubt". Additionally, according to the condition

$$P(y = i|x) \geq 1 - \frac{\lambda_r}{\lambda_s}$$

in order to classify a data point as being in class i, we would need $P(y = i|x) = 1$, or in other words, we would need to have perfect certainty about our data point's class. If the goal is to minimize risk, and there is no risk associated with just not choosing a class unless you have perfect certainty, then in any situation where there is randomness (i.e. a situation in which you would want a statistical classifier) you would just fail to classify any points. This would be a classifier with zero risk, but it would be pretty much useless since it does no actual classification.

In the case where $\lambda_r > \lambda_s$

$$1 - \frac{\lambda_r}{\lambda_s} < 0$$

Since all probabilities are non-negative, this would render insignificant the condition that

$$P(y = i|x) \geq 1 - \frac{\lambda_r}{\lambda_s}$$

as it would invariably be true. Also, in this case it would be strictly true that

$$\lambda_s(1 - P(y = i|x)) < \lambda_r$$

and so we would always have lower risk just classifying a data point as whatever class has the greatest posterior probability, never choosing "doubt".

Intuitively, these two results make sense: if the goal is a minimal risk classifier and there is no risk at all in choosing to just not classify a point, you would do so for all points (and have a useless classifier). If instead it was more risky to not choose a class, then you would choose a class in all cases, rendering the option to "doubt" pretty much useless. The scenario where a "doubt" option becomes useful is when $\lambda_r \leq \lambda_s$ (such that $\lambda_r > 0$).

6 Maximum Likelihood Estimation and Bias

6.1

Source: https://en.wikipedia.org/wiki/Maximum_likelihood_estimation

For a sample X_i

$$f_X(X_i|\mu,\sigma) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma_i^2}\right)$$

Since the samples are independent, we can say

$$f_X(X_1,...,X_n|\mu,\sigma) = f_X(X_1|\mu,\sigma)f_X(X_2|\mu,\sigma)...f_X(X_n|\mu,\sigma)$$

$$= \mathcal{L}_X(\mu,\sigma;X_1,...,X_n)$$

We see that

$$\mathcal{L}(\mu,\sigma;X_1,...,X_n) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma_i^2}\right)$$

Because

$$\arg\max_{\mu,\sigma} \mathcal{L}(\mu,\sigma;X_1,...,X_n) = \arg\max_{\mu,\sigma} \log\left(\mathcal{L}(\mu,\sigma;X_1,...,X_n)\right)$$

Let the estimator

$$\hat{\mu} = \arg\max_{\mu} \log\left(\mathcal{L}(\mu;\sigma,X_1,...,X_n)\right)$$

Note that

$$\log\left(\mathcal{L}(\mu;\sigma,X_1,...,X_n)\right) = \log\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x_i-\mu)^2}{2\sigma_i^2}\right)$$

$$= \log\prod_{i=1}^{n} \left(\frac{i}{2\pi\sigma^2}\right)^{\frac{1}{2}} \exp\left(-\frac{i(x_i-\mu)^2}{2\sigma^2}\right)$$

Because $\sigma_i = \frac{\sigma}{\sqrt{i}}$

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

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

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

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

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

So we get

$$\frac{\partial}{\partial \mu} \log \left(\mathcal{L}(\mu; \sigma, X_1, ..., X_n) \right) = \frac{\partial}{\partial \mu} \left(\frac{1}{2} \sum_{i=1}^{n} \log(i) - \frac{n}{2} \log(2\pi) - n\log(\sigma) - \frac{1}{2} \sum_{i=1}^{n} \frac{i(x_i - \mu)^2}{\sigma^2} \right)$$

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

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

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

Set equal to zero to get $\hat{\mu}$

$$\frac{1}{\sigma^2} \left(\sum_{i=1}^{n} i x_i - \sum_{i=1}^{n} i\hat{\mu} \right) = 0$$

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

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

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

Next we find

$$\frac{\partial}{\partial\sigma}\log\left(\mathcal{L}(\sigma;\hat{\mu},X_1,...,X_n)\right) = \frac{\partial}{\partial\sigma}\left(\frac{1}{2}\sum_{i=1}^{n}\log(i) - \frac{n}{2}\log(2\pi) - n\log(\sigma) - \frac{1}{2}\sum_{i=1}^{n}\frac{i(x_i-\hat{\mu})^2}{\sigma^2}\right)$$

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

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

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

Next, set equal to zero to get $\hat{\sigma}^2$

$$-\frac{n}{\hat{\sigma}} + \frac{1}{\hat{\sigma}^3}\sum_{i=1}^{n} i(x_i-\hat{\mu})^2 = 0$$

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

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

6.2

$\hat{\mu}$ is an unbiased estimator. Proof:

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

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

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

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

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

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

$$= \mu$$

Therefore

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

$$= \mu - \mu$$

$$= 0$$

Q.E.D.

6.3

$\hat{\sigma}^2$ is a *biased* estimator. Proof:

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

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

$$= \frac{1}{n}\mathbb{E}\left[\sum_{i=1}^{n} ix_i^2 - 2\sum_{i=1}^{n} ix_i\hat{\mu} + \sum_{i=1}^{n} i\hat{\mu}^2)\right]$$

$$= \frac{1}{n}\mathbb{E}\left[\sum_{i=1}^{n} ix_i^2 - 2\hat{\mu}\sum_{i=1}^{n} ix_i + \hat{\mu}^2\sum_{i=1}^{n} i\right]$$

Let $\sum_{i=1}^{n} i = \alpha$. Note then that $\hat{\mu} = \frac{1}{\alpha}\sum_{i=1}^{n} ix_i$

$$= \frac{1}{n}\left(\sum_{i=1}^{n} i\mathbb{E}[x_i^2] - 2\mathbb{E}\left[\hat{\mu}\sum_{i=1}^{n} ix_i\right] + \alpha\mathbb{E}[\hat{\mu}^2]\right)$$

$$= \frac{1}{n}\left(\sum_{i=1}^{n} i\left(\mathrm{Var}(x_i) + (\mathbb{E}[x_i])^2\right) - 2\mathbb{E}\left[\frac{1}{\alpha}\sum_{i=1}^{n} ix_i\sum_{i=1}^{n} ix_i\right] + \alpha\mathbb{E}[\hat{\mu}^2]\right)$$

$$= \frac{1}{n}\left(\sum_{i=1}^{n} i\left(\frac{\sigma^2}{i} + \mu^2\right) - \frac{2}{\alpha}\mathbb{E}\left[\left(\sum_{i=1}^{n} ix_i\right)^2\right] + \alpha\mathbb{E}[\hat{\mu}^2]\right)$$

$$= \frac{1}{n}\left(\sum_{i=1}^{n}\sigma^2 + \sum_{i=1}^{n} i\mu^2 - \frac{2}{\alpha}\left(\mathrm{Var}\left(\sum_{i=1}^{n} ix_i\right) + \left(\mathbb{E}\left[\sum_{i=1}^{n} ix_i\right]\right)^2\right) + \alpha\mathbb{E}[\hat{\mu}^2]\right)$$

$$= \frac{1}{n}\left(n\sigma^2 + \alpha\mu^2 - \frac{2}{\alpha}\left(\mathrm{Var}\left(\sum_{i=1}^{n} ix_i\right) + (\alpha\mu)^2\right) + \alpha\mathbb{E}[\hat{\mu}^2]\right)$$

$$= \frac{1}{n}\left(n\sigma^2 + \alpha\mu^2 - \frac{2}{\alpha}\text{Var}\left(\sum_{i=1}^{n} ix_i\right) - 2\alpha\mu^2 + \alpha\mathbb{E}[\hat{\mu}^2]\right)$$

$$= \frac{1}{n}\left(n\sigma^2 - \alpha\mu^2 - \frac{2}{\alpha}\text{Var}\left(\sum_{i=1}^{n} ix_i\right) + \alpha\left(\text{Var}\left(\hat{\mu}\right) + \left(\mathbb{E}[\hat{\mu}]\right)^2\right)\right)$$

$$= \frac{1}{n}\left(n\sigma^2 - \alpha\mu^2 - \frac{2}{\alpha}\text{Var}\left(\sum_{i=1}^{n} ix_i\right) + \alpha\left(\text{Var}\left(\frac{1}{\alpha}\sum_{i=1}^{n} ix_i\right) + \mu^2\right)\right)$$

$$= \frac{1}{n}\left(n\sigma^2 - \alpha\mu^2 - \frac{2}{\alpha}\text{Var}\left(\sum_{i=1}^{n} ix_i\right) + \frac{1}{\alpha}\text{Var}\left(\sum_{i=1}^{n} ix_i\right) + \alpha\mu^2\right)$$

$$= \frac{1}{n}\left(n\sigma^2 - \frac{1}{\alpha}\text{Var}\left(\sum_{i=1}^{n} ix_i\right)\right)$$

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

Because the x_i's are independent

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

$$= \frac{1}{n}\left(n\sigma^2 - \frac{\sigma^2}{\alpha}\alpha\right)$$

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

Therefore, the bias is

$$bias(\hat{\sigma}^2) = \frac{(n-1)}{n}\sigma^2 - \sigma^2$$

$$= \frac{-\sigma^2}{n}$$

$$\neq 0$$

Q.E.D.

7 Covariance Matrices and Decompositions

7.1

A matrix is singular when its determinant is zero. A zero determinant is indicative of a matrix not having full rank, i.e. the dimension of its column space is less than either the number of rows or columns. Geometrically, this would correspond to all linear combinations of vectors in $\mathbb{R}^n$ spanning a space which has dimension less than n, For example, vectors in $\mathbb{R}^3$ only being able to space a plane, line or point (i.e. the zero vector) within $\mathbb{R}^3$.

Note that the sample covariance matrix is symmetric because it is a sum of symmetric matrices of the form AA^T making it at least positive semi-definite. This means, for all of its eigenvalues σ_i

$$\sigma_i \geq 0$$

Which means some of its eigenvalues could be zero. This is important because

$$\det(\hat{\Sigma}) = \prod_i \sigma_i$$

and thus, if any of its eigenvalues are zero, it will have a zero determinant and therefore be singular. Zero eigenvalues indicate that the linear transformation "squashes" vectors in some n-dimensional space, into a subspace of dimension less than n (e.g. vectors which span $\mathbb{R}^3$ all being mapped to a plane or line inside of $\mathbb{R}^3$.

With data, this can occur when a feature vector contains features which are linear combinations of other features. A singular sample covariance matrix can also occur when there is high multicollinearity between features as this could pose the same problem as features being scalar multiples of other features. A feature vector with zero variance (i.e. all features are invariant) would have a singular covariance matrix because it would have zero eigenvalues.

7.2

Source: doi: 10.1109/TIT.2011.2162175.

If some sample covariance matrix $\hat{\Sigma}$ is singular, that would mean it has at least one zero eigenvalue. We want to "fix" it by making it positive definite, while at the same time keeping it as close as possible to the original estimator. We can do this by adding some small positive constant to the diagonal of $\hat{\Sigma}$. This is known as diagonal loading. Let the new estimator $\hat{\Sigma}'$ be one such that

$$\hat{\Sigma}' - \hat{\Sigma} = \epsilon I_p$$

where ϵ is the loading constant. We want $0 < \epsilon << 1$. A simple, a priori kludge factor could be

$$\epsilon = \frac{1}{n^4}$$

Thus, the new matrix could be

$$\hat{\Sigma}' = \frac{1}{n^4} I_p + \hat{\Sigma}$$

Which will tend towards the true covariance matrix Σ as the sample number increases.

7.3

Sources: The Matrix Cookbook &
https://www.youtube.com/watch?v=6oZT72-nnyI

Let

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

and

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

We can use the condition that $\|x\| = 1$ and the scalar value λ with the method of Lagrange multipliers to get

$$\nabla_x \log f = \lambda\nabla_x((x^Tx) - 1)$$

$$\frac{\partial}{\partial x}\log((2\pi)^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}}) - \frac{1}{2}\frac{\partial}{\partial x}(x-\mu)^T\Sigma^{-1}(x-\mu) = \lambda 2x$$

$$-\frac{1}{2}\left(\Sigma^{-1}(x-\mu) + (\Sigma^{-1})^T(x-\mu)\right) = 2\lambda x$$

$$-\frac{1}{2}\left(2\Sigma^{-1}(x-\mu)\right) = 2\lambda x$$

$$-\Sigma^{-1}(x-\mu) = 2\lambda x$$

In this case, $\mu = 0$ so

$$-\Sigma^{-1}x = 2\lambda x$$

$$\Sigma^{-1}x = -2\lambda x$$

But -2λ is just another constant so

$$\Sigma^{-1}x = \alpha x$$

Here, α represents an eigenvalue of Σ^{-1} and therefore x is an eigenvector Σ^{-1}. We see that for the ith eigenvalue/eigenvector pair

$$\|\alpha_i x_i\| = |\alpha_i|\|x_i\|$$

$$= \alpha_i$$

Therefore, the x vectors with magnitude 1 that minimize and maximize f, are the eigenvectors which correspond to $\alpha_{\min}$ and $\alpha_{\max}$

7.4

Source: Math StackExchange

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

$$= \mathbb{E}(y^T X (y^T X)^T) - (\mathbb{E}(y^T X))^2$$

$$= \mathbb{E}(y^T X X^T y) - (y^T \mathbb{E}(X))^2$$

$$= y^T \mathbb{E}(X X^T) y$$

Because y is not a random vector

$$= y^T \mathbb{E} \begin{bmatrix} X_1^2 & X_1 X_2 & . & . & . & X_1 X_n \\ . & . & & & & . \\ . & & . & & & . \\ . & & & & . & . \\ X_n X_1 & X_n X_2 & . & . & . & X_n^2 \end{bmatrix} y$$

$$= y^T \begin{bmatrix} \mathbb{E}[X_1^2] & \mathbb{E}[X_1 X_2] & . & . & . & \mathbb{E}[X_1 X_n] \\ . & . & & & & . \\ . & & . & & & . \\ . & & & & . & . \\ \mathbb{E}[X_n X_1] & \mathbb{E}[X_n X_2] & . & . & . & \mathbb{E}[X_n^2] \end{bmatrix} y$$

$$= y^T \begin{bmatrix} \text{Var}(X_1) + (E[X_1])^2 & \text{Cov}(X_1, X_2) + \mathbb{E}[X_1]\mathbb{E}[X_2] & . \ . \ . & \text{Cov}(X_1, X_n) + \mathbb{E}[X_1]\mathbb{E}[X_n] \\ . & . & & . \\ . & & . & . \\ . & & & . \\ \text{Cov}(X_n, X_1) + \mathbb{E}[X_n]\mathbb{E}[X_1] & \text{Cov}(X_n, X_2) + \mathbb{E}[X_n]\mathbb{E}[X_2] & . \ . \ . & \text{Var}(X_n) + (E[X_n])^2 \end{bmatrix} y$$

$$= y^T \begin{bmatrix} \text{Var}(X_1) & \text{Cov}(X_1, X_2) & . \ . \ . & \text{Cov}(X_1, X_n) \\ . & . & & . \\ . & & . & . \\ . & & & . \\ \text{Cov}(X_n, X_1) & \text{Cov}(X_n, X_2) & . \ . \ . & \text{Var}(X_n) \end{bmatrix} y$$

Because $\mathbb{E}[X_i] = 0, \quad \forall i \in \{1, ..., n\}$

$$\boxed{= y^T \Sigma y}$$

Let y be the unit eigenvector of the covariance matrix corresponding to the largest eigenvalue, $\lambda_{\max}(\Sigma)$. Then

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

$$= y^T \lambda_{\max}(\Sigma) y$$

$$= \lambda_{\max}(\Sigma) \|y\|$$

$$= \lambda_{\max}(\Sigma)$$

This tells us that variances of dot products between unit vectors and Gaussian random vectors are bounded by the greatest eigenvalue of the covariance matrix or

$$\boxed{\text{Var}(y^T X) \leq \lambda_{\max}(\Sigma), \quad \forall y \in \mathbb{R}^n, \quad \|y\| = 1}$$

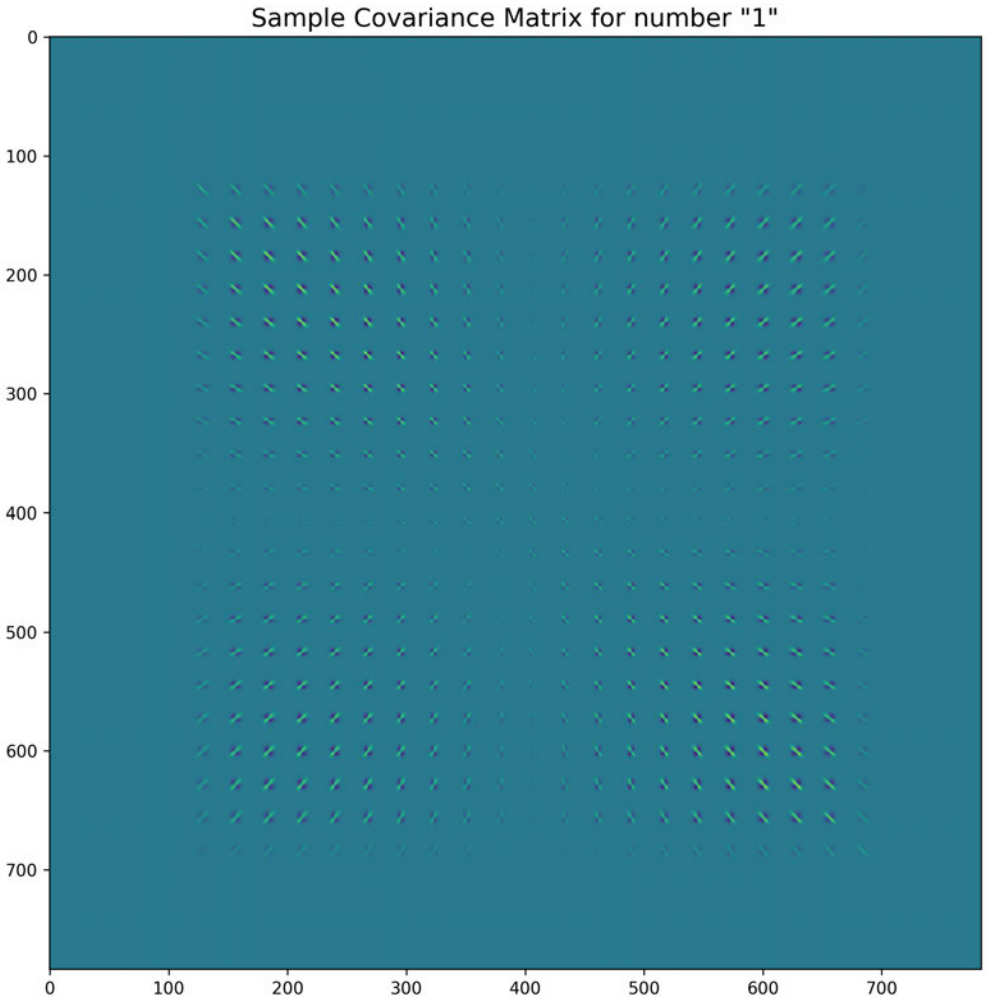
8 Gaussian Classifiers for Digits and Spam

8.1

For this problem, I took the MNIST training data set and then separated it into classes defined by their labels. I then created two separate dictionaries: one containing the sample mean vectors (each feature mean was simply calculated with df['<col. name>'].mean()), and one containing the covariance matrices (calculating with numpy.cov()) for each class. See code in the appendix.

8.2 (a)

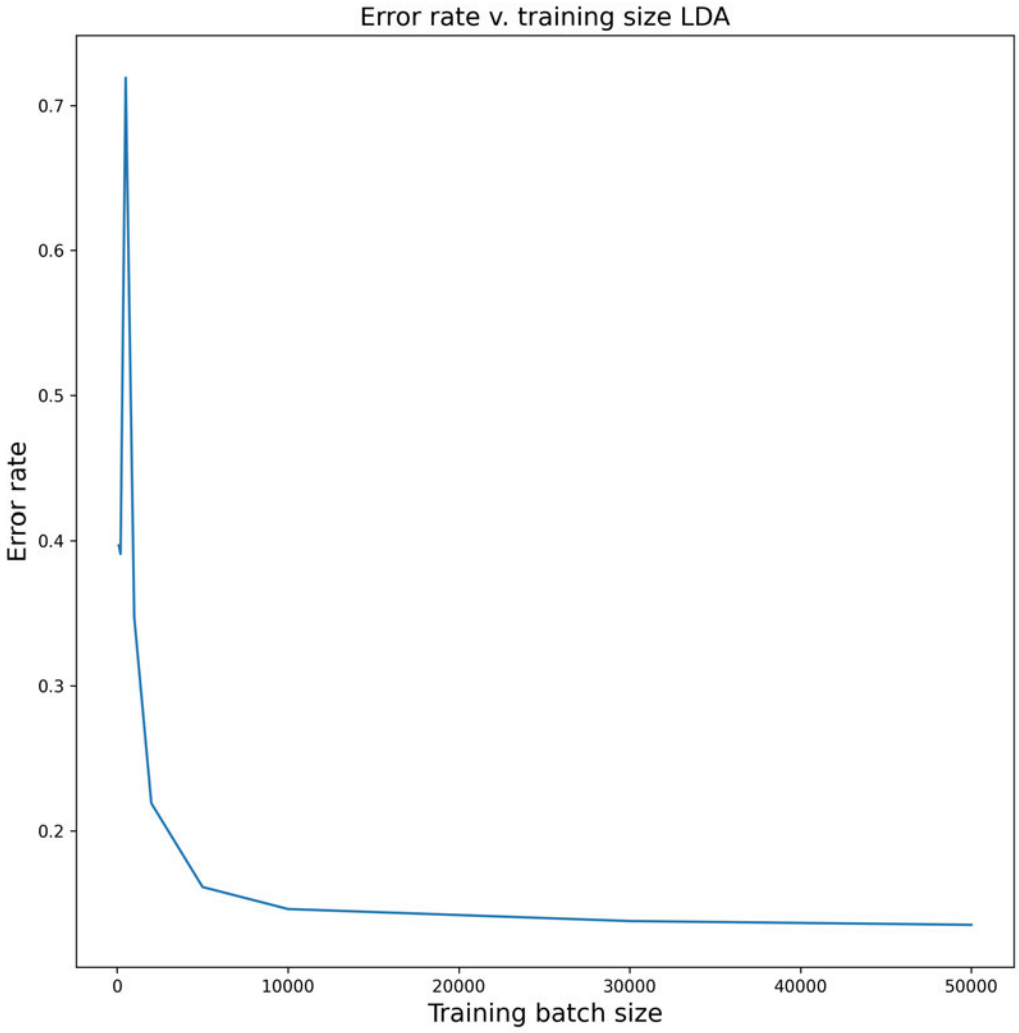
I plotted the sample covariance matrix for the class "1" as an image with pixel brightness representing the variances/covariances. This is an interesting covariance matrix as the the variances are low, while the surrounding covariances are relatively high. This should be due to the fact that one is a simpe number with primarily a single vertical stroke. Therefore, there should not be much individual pixel variance, except for those pixels near the center of the image where the digit is most likely to be. There should also be high positive covariances between the different pixels along the the vertical line in the center of the image, which is exactly what the covariance matrix shows.



8.3

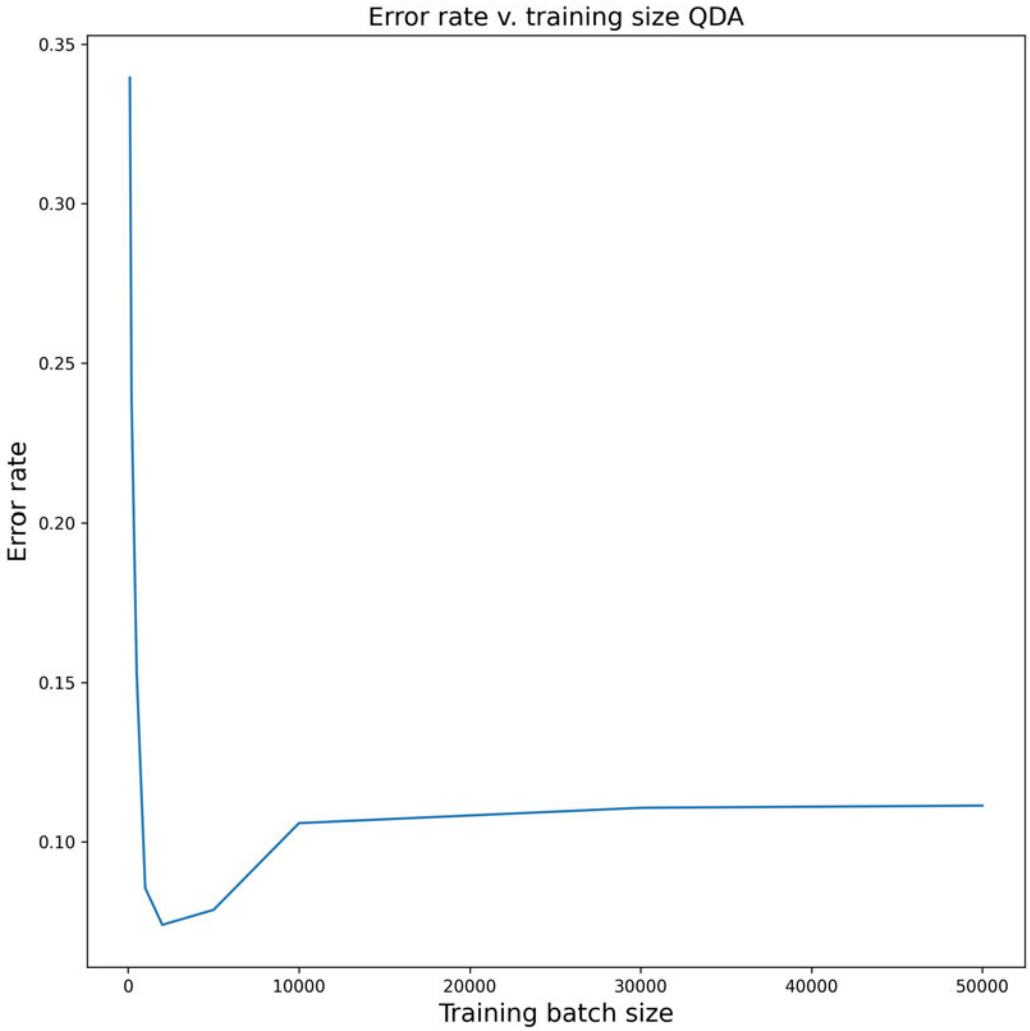
8.3.1 (a)

For this LDA model, I used the kludge of diagonal loading I described in 7.4 to get a slightly transformed version of the pooled sample covariance matrix that is positive definite. It appears to have worked fairly well as I achieved a lowest error rate of about 13.5% when training on 50,000 samples.



8.3.2 (b)

For QDA, I had to change my kludge method, because the method of adding a factor of $1/n^4$ that I used for LDA was causing the model to increase in error with increasing sample size. Instead, I added $0.001\lambda_{\max}(\hat{\Sigma})$. I chose the max, rather than min, eigenvalue, because there were apparently some covariance matrices with min eigenvalues of zero. Still, QDA appears to outperform LDA.

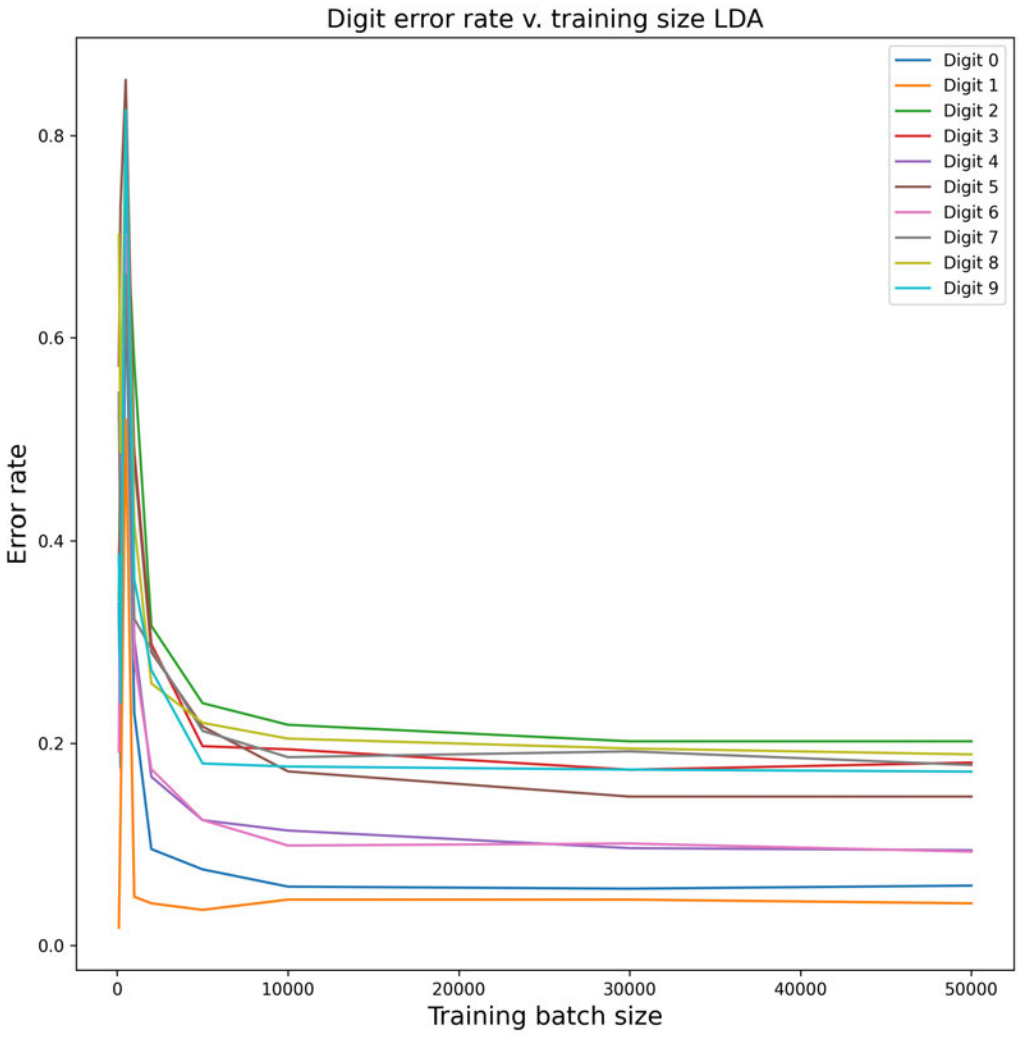


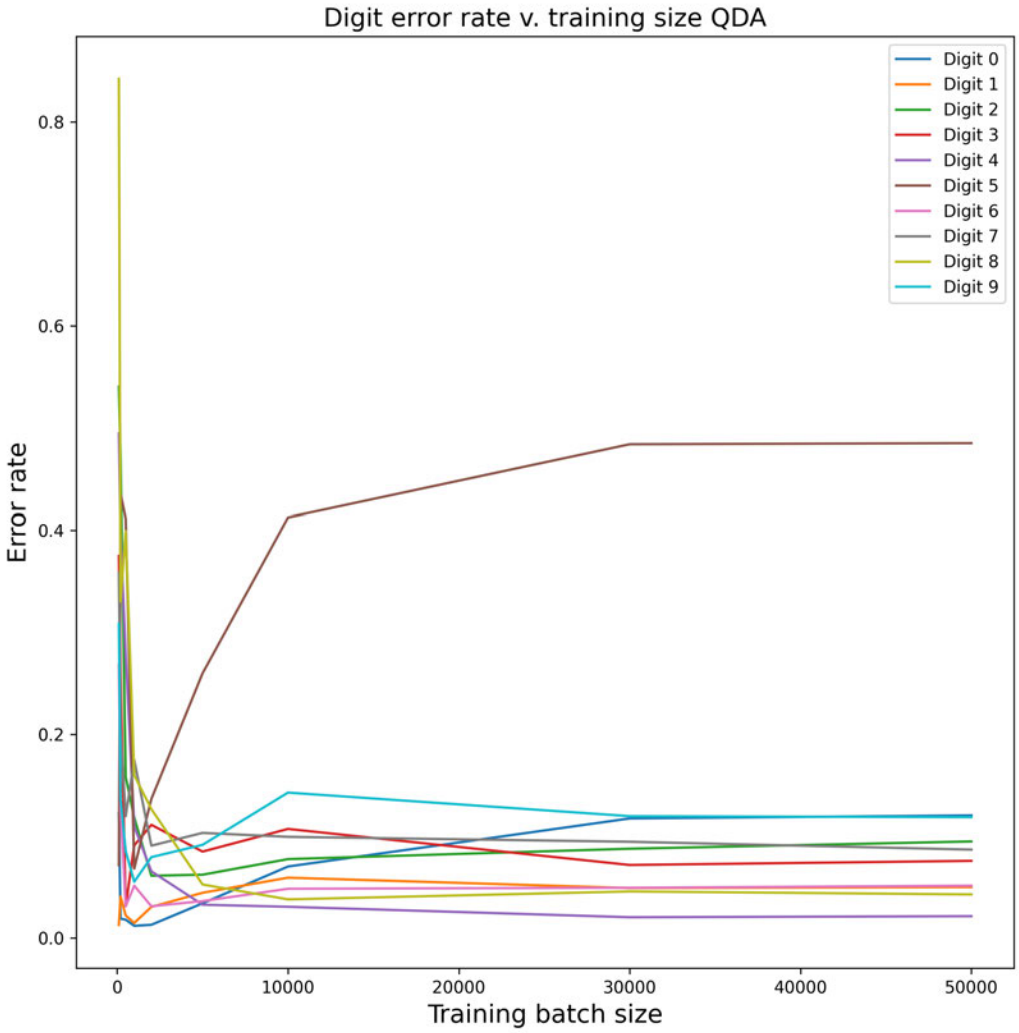
8.3.3 (c)

QDA appears to perform slightly better than LDA. This is probably due to LDA treating the classes as though they have equal variances, which is highly unlikely with digits. For example, we may expect a wider variation for the way people write the number "4", and less for a number like "0".

8.3.4 (d)

LDA and QDA did best with different digits. LDA performed best with "1" and this makes sense to make because it has only a very distinct vertical line that sets it apart from all the other numbers (though "7" can look similar in some cases). QDA did best with "4", but I do not exactly know why. I assume "4" could have a high variance because it has both an "open" and "closed" form which are used by people probably equally. Perhaps QDA does better with features that have clearly different variances. However, QDA really struggled with "5" for some reason, and I do not expect "5" to have a particularly low variance.




Digit error rate v. training size QDA
Digit 0
Digit 1
Digit 2
Digit 3
Digit 4
Digit 5
Digit 6
Digit 7
Digit 8
Digit 9
Error rate
0.0
0.2
0.4
0.6
0.8
0
10000
20000
30000
40000
50000
Training batch size

8.4

Kaggle username:Colin Skinner
Prediction rate: 0.80500

8.5

In [1]:
```
from math import sqrt,pi
import numpy as np
import pandas as pd
from scipy.stats import multivariate_normal as mvnorm
import matplotlib.pyplot as plt

%matplotlib inline
```

Q3

In [2]:
```
def grid_gen(xlim,ylim,step):
    return np.mgrid[-xlim:xlim:step, -ylim:ylim:step]
```

In [3]:
```
def f_eval(grid,mu,cov):
    pos = np.dstack((grid[0],grid[1]))
    return mvnorm.pdf(pos,mu,cov)
```

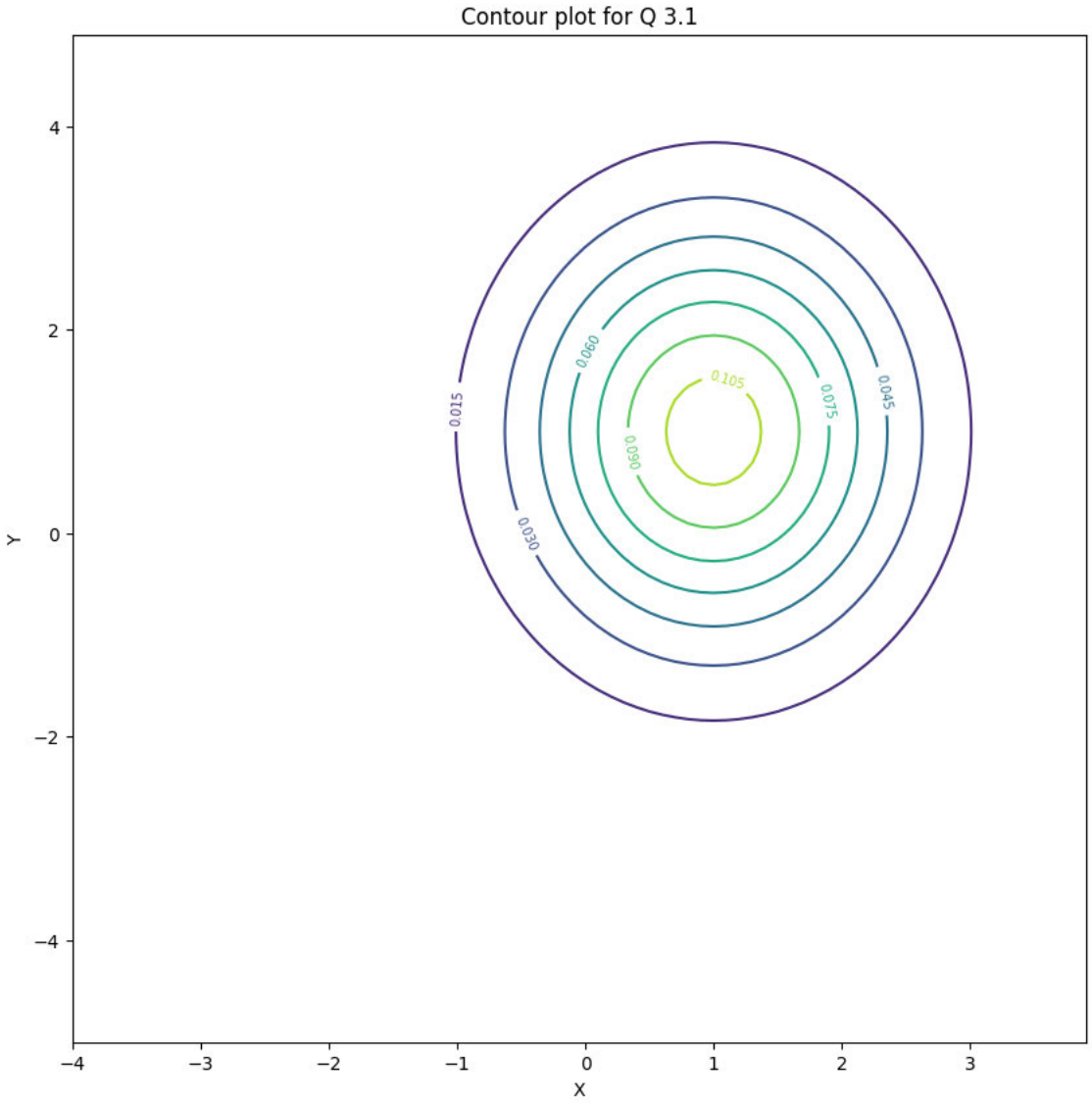
In [4]:
```
def contour(prob,mu=None,cov=None,xlim=None,ylim=None,step=None,f=None,grid=None,flag=True):

    if flag==True:
        grid = grid_gen(xlim,ylim,step)
        z = f_eval(grid,mu,cov)

    else:
        z=f

    fig = plt.figure(figsize=(10,10))
    cs = plt.contour(grid[0],grid[1],z)
    plt.clabel(cs,inline_spacing=5, fontsize=7)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Contour plot for {}'.format(prob))
#     plt.savefig('{}_plot.png'.format(prob), dpi=300)
```

3.1

In [149]:
```
mu1 = np.array([1,1])
```

In [150]:
```
cov1 = np.array([[1, 0],[0, 2]])
```

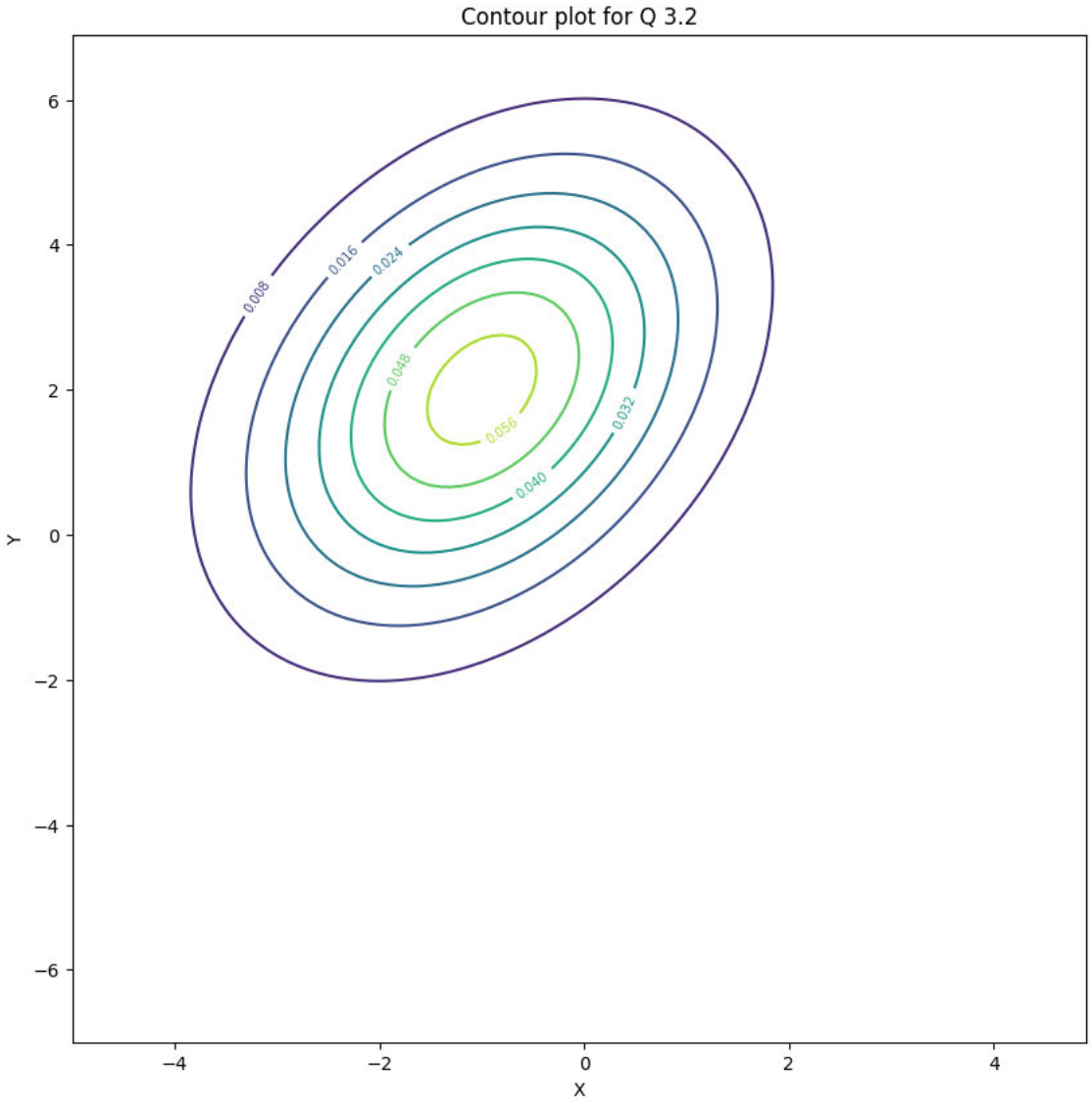
In [151]:
```
contour('Q 3.1',mu1,cov1,4,5,0.1)
```



3.2

In [152]:
```
mu2 = np.array([-1,2])
cov2 = np.array([[2, 1],[1, 4]])
```

In [153]:
```
contour('Q 3.2',mu2,cov2,5,7,0.1)
```



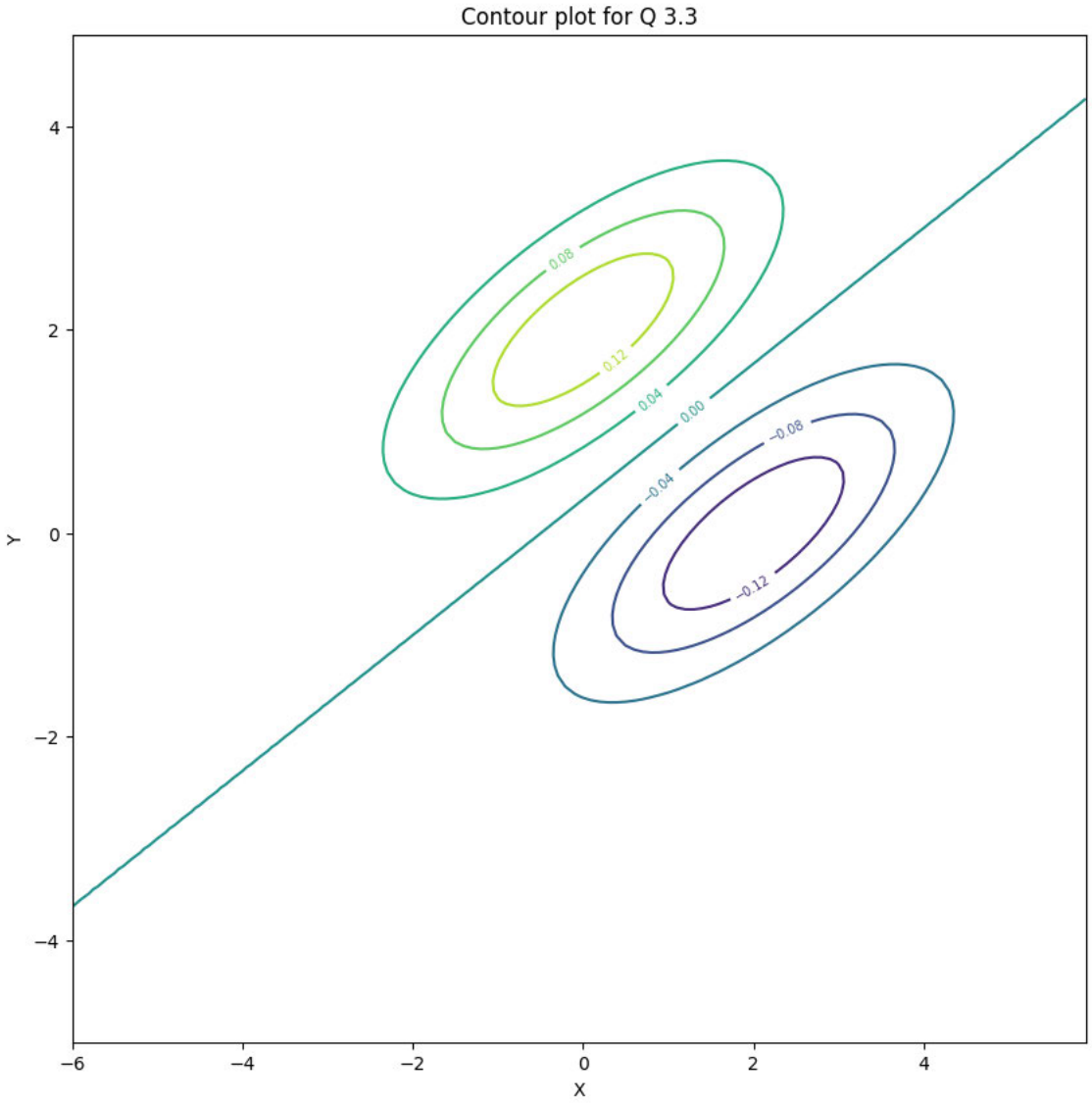
3.3

In [154]:
```
mu1 = np.array([0,2])
mu2 = np.array([2,0])
cov = np.array([[2, 1],[1, 1]])
```

In [155]:
```
mgrid = grid_gen(6,5,0.1)
```

In [156]:
```
z = f_eval(mgrid,mu1,cov)-f_eval(mgrid,mu2,cov)
```

In [157]:
```
contour('Q 3.3',f=z,grid=mgrid,flag=False)
```

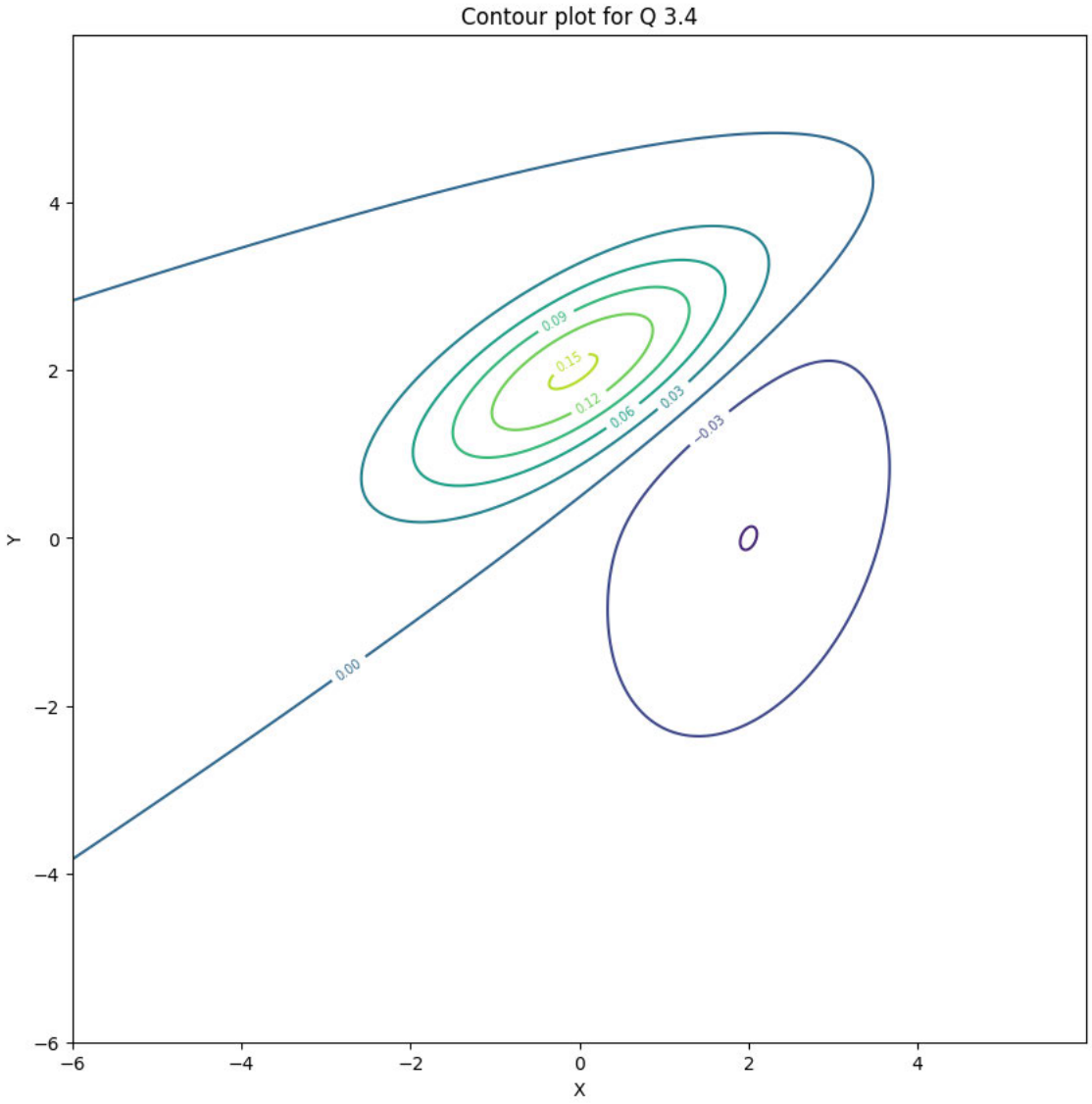


3.4

In [183]:
```
mu1 = np.array([0,2])
mu2 = np.array([2,0])
cov1 = np.array([[2, 1],[1, 1]])
cov2 = np.array([[2, 1],[1, 4]])
```

In [181]:
```
mgrid = grid_gen(6,6,0.01)
z = f_eval(mgrid,mu1,cov1)-f_eval(mgrid,mu2,cov2)
```

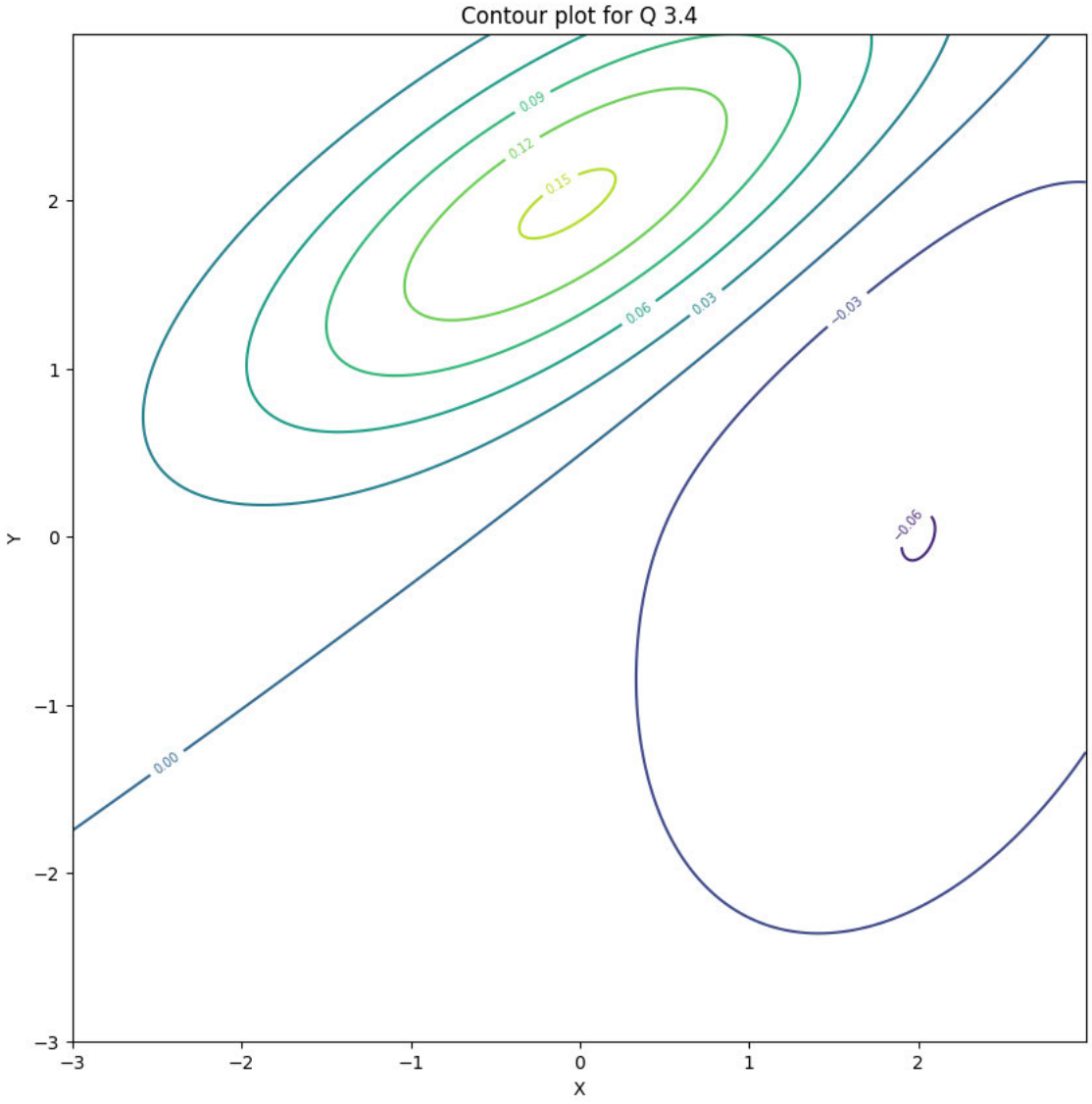
In [182]:
```
contour('Q 3.4',f=z,grid=mgrid,flag=False)
```



Zoom in to get the value for the one unlabeled contour

In [192]:
```
mgrid = grid_gen(3,3,0.01)
z = f_eval(mgrid,mu1,cov1)-f_eval(mgrid,mu2,cov2)
```

In [194]:
```
fig = plt.figure(figsize=(10,10))
cs = plt.contour(mgrid[0],mgrid[1],z)
plt.clabel(cs,inline_spacing=5, fontsize=7)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Contour plot for Q 3.4')
# plt.savefig('Q 3.4_plot_zoom.png', dpi=300)
```

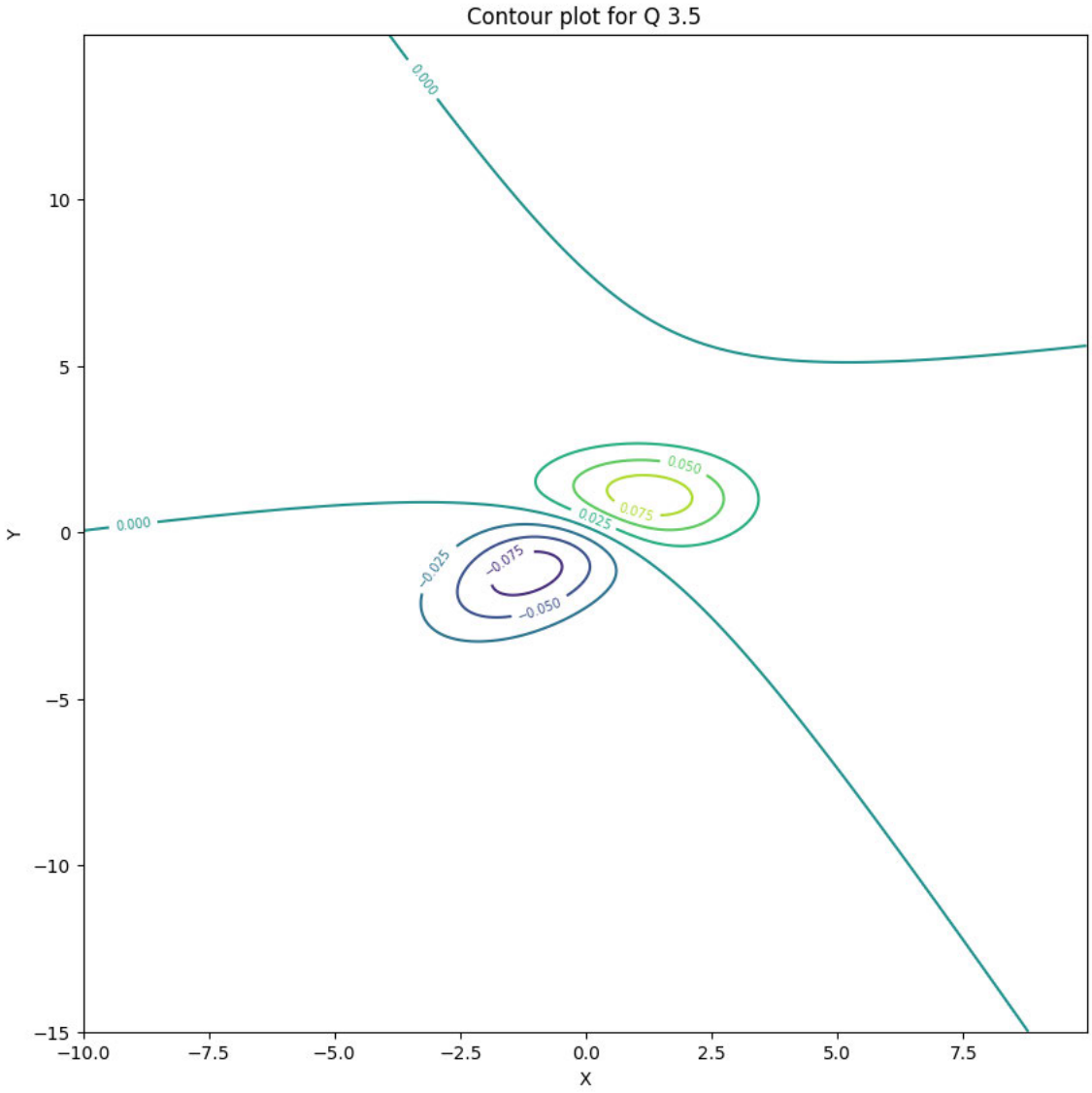


3.5

In [161]:
```
mu1 = np.array([1,1])
mu2 = np.array([-1,-1])
cov1 = np.array([[2, 0],[0, 1]])
cov2 = np.array([[2, 1],[1, 2]])
```

In [162]:
```
mgrid = grid_gen(10,15,0.05)
z = f_eval(mgrid,mu1,cov1)-f_eval(mgrid,mu2,cov2)
```

In [163]:
```
contour('Q 3.5',f=z,grid=mgrid,flag=False)
```



Q4

In [5]:
```
class tupl(object):
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return tupl(self.x+other.x,self.y+other.y)

    def __sub__(self, other):
        return tupl(self.x - other.x, self.y - other.y)

    def __rmul__(self, other):
        if isinstance(other, int) or isinstance(other, float):
            return tupl(other*self.x, other*self.y)

    def __repr__(self):
        return f"({self.x},{self.y})"
```

In [6]:
```
np.random.seed(42)
```

4.1

In [7]:
```
points = []

for i in range(100):
    x1=np.random.normal(3,3)
    points+=[tupl(x1 ,0.5*x1+np.random.normal(4,2))]
```

In [8]:
```
temp_sum=tupl(0,0)

for point in points:
    temp_sum+=point

points_mean = (1/len(points))*temp_sum
```

In [9]:
```
print('The mean in R2 is: ', points_mean)
```

```
The mean in R2 is:  (2.6533072356348306,5.394698266705852)
```

4.2

In [10]:
```
x1=[]
x2=[]

for point in points:
    x1+=[point.x]
    x2+=[point.y]

temp_matrix = np.array([x1,x2])
```

In [11]:
```
sigma = np.cov(temp_matrix)
```

In [12]:
```
print("The covariance matrix based off the samples is: \n\n ", sigma)
```

```
The covariance matrix based off the samples is:

  [[6.59945084 3.46560266]
 [3.46560266 5.80661362]]
```

4.3

In [633]:
```
eig_vals, eig_vecs = np.linalg.eig(sigma)
```

In [634]:
```
eig_vecs
```

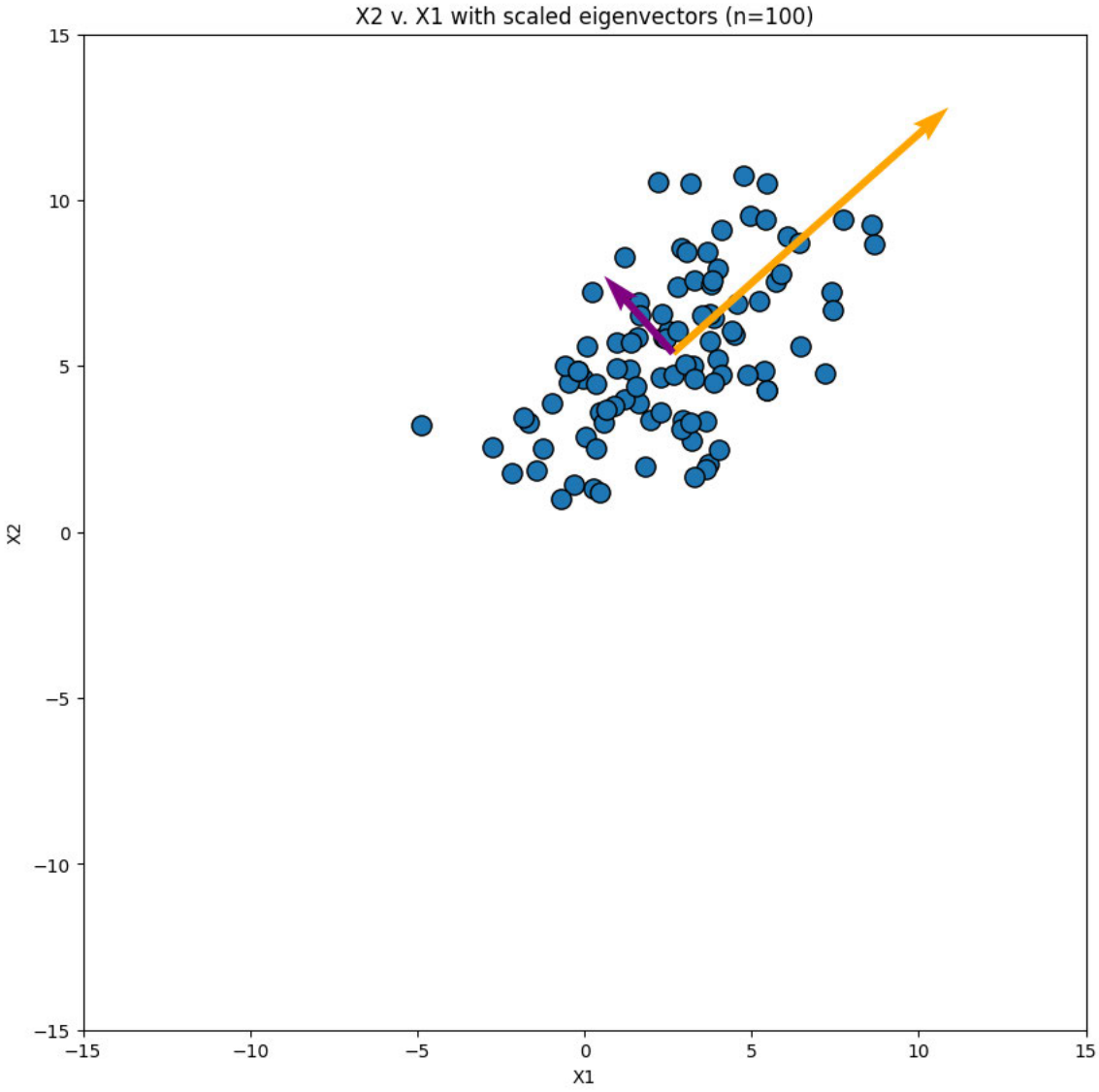
Out[634]:
```
array([[ 0.74620559, -0.66571557],
       [ 0.66571557,  0.74620559]])
```

In [638]:
```
eig_vals
```

Out[638]:
```
array([9.6912337 , 2.71483076])
```

4.4

In [694]:
```
plt.figure(figsize=(10,10))
xlim = [-15,15]
ylim = [-15,15]

plt.scatter(x1,x2,s=120, edgecolor='k')
plt.quiver(points_mean.x,points_mean.y, eig_vecs[0][0],eig_vecs[1][0],color='orange',scale=eig_vals[1])
plt.quiver(points_mean.x,points_mean.y, eig_vecs[0][1],eig_vecs[1][1],color='purple',scale=eig_vals[0])
plt.xlim(xlim)
plt.ylim(ylim)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('X2 v. X1 with scaled eigenvectors (n=100)')
# plt.savefig('Q4.4_plot.png', dpi=300)
```

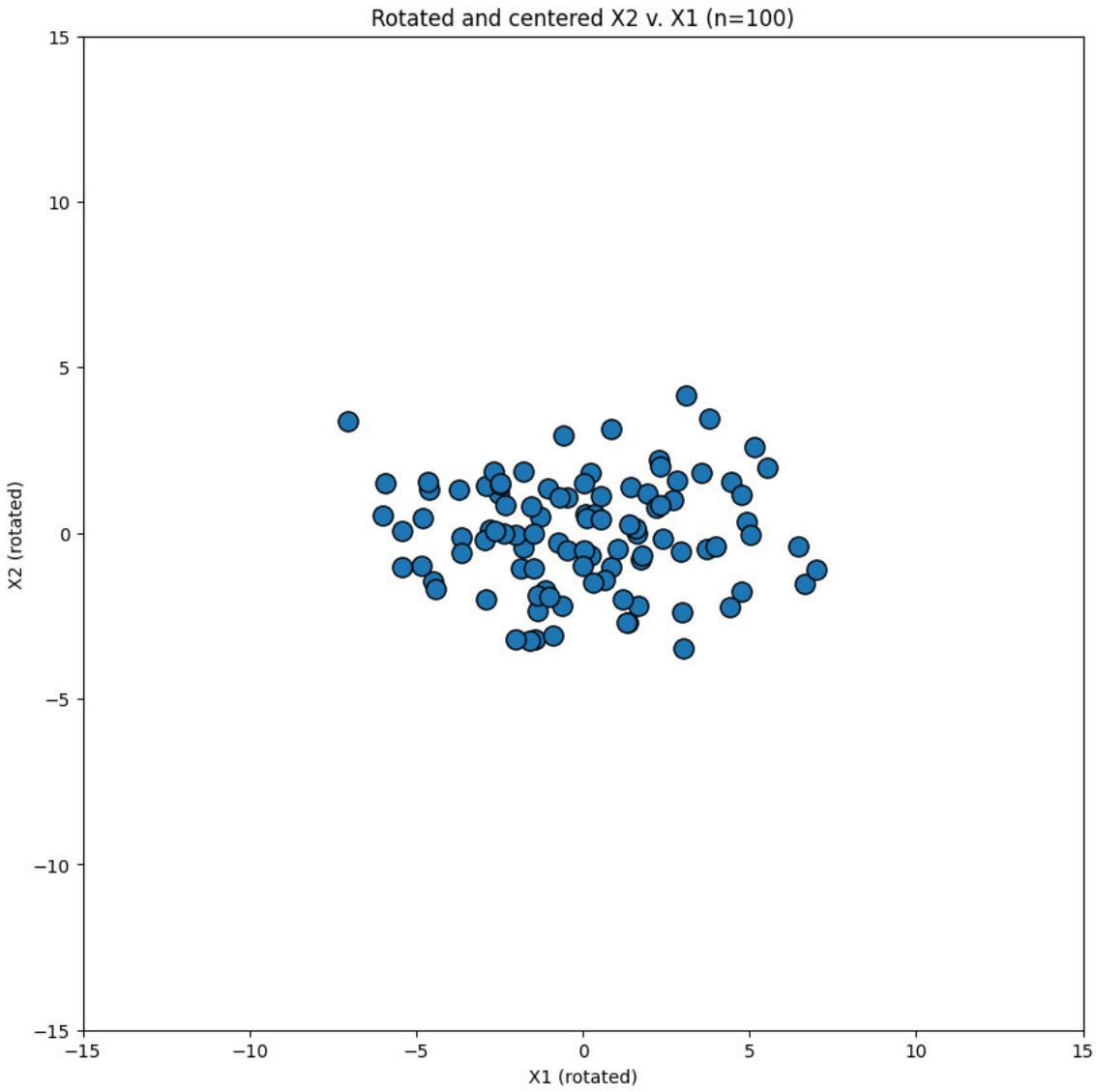


4.5

In [659]:
```
x_temp=[]

for point in points:

    x_temp+=[point-points_mean]
```

In [685]:
```
x_rot_x =[]
x_rot_y =[]

for i in range(len(x_temp)):
    x_rot_x+=[np.matmul(eig_vecs.T,np.array([x_temp[i].x,x_temp[i].y]))[0]]
    x_rot_y+=[np.matmul(eig_vecs.T,np.array([x_temp[i].x,x_temp[i].y]))[1]]
```

In [695]:
```
plt.figure(figsize=(10,10))
xlim = [-15,15]
ylim = [-15,15]

plt.scatter(x_rot_x,x_rot_y,s=120, edgecolor='k')
plt.xlim(xlim)
plt.ylim(ylim)
plt.xlabel('X1 (rotated)')
plt.ylabel('X2 (rotated)')
plt.title('Rotated and centered X2 v. X1 (n=100)')
# plt.savefig('Q4.5_plot.png', dpi=300)
```



Q8

8.1

In [144]:
```
mnist = np.load('data/mnist-data-hw3.npz')
```

In [145]:
```
mnist.files
```

Out[145]:
```
['training_data', 'training_labels', 'test_data']
```

In [146]:
```
mnist['training_data'].shape
```

Out[146]:
```
(60000, 1, 28, 28)
```

In [287]:
```
#Need to reshape the data to have the images be row vectors
mnist_data=mnist['training_data'].reshape(60000,784)
```

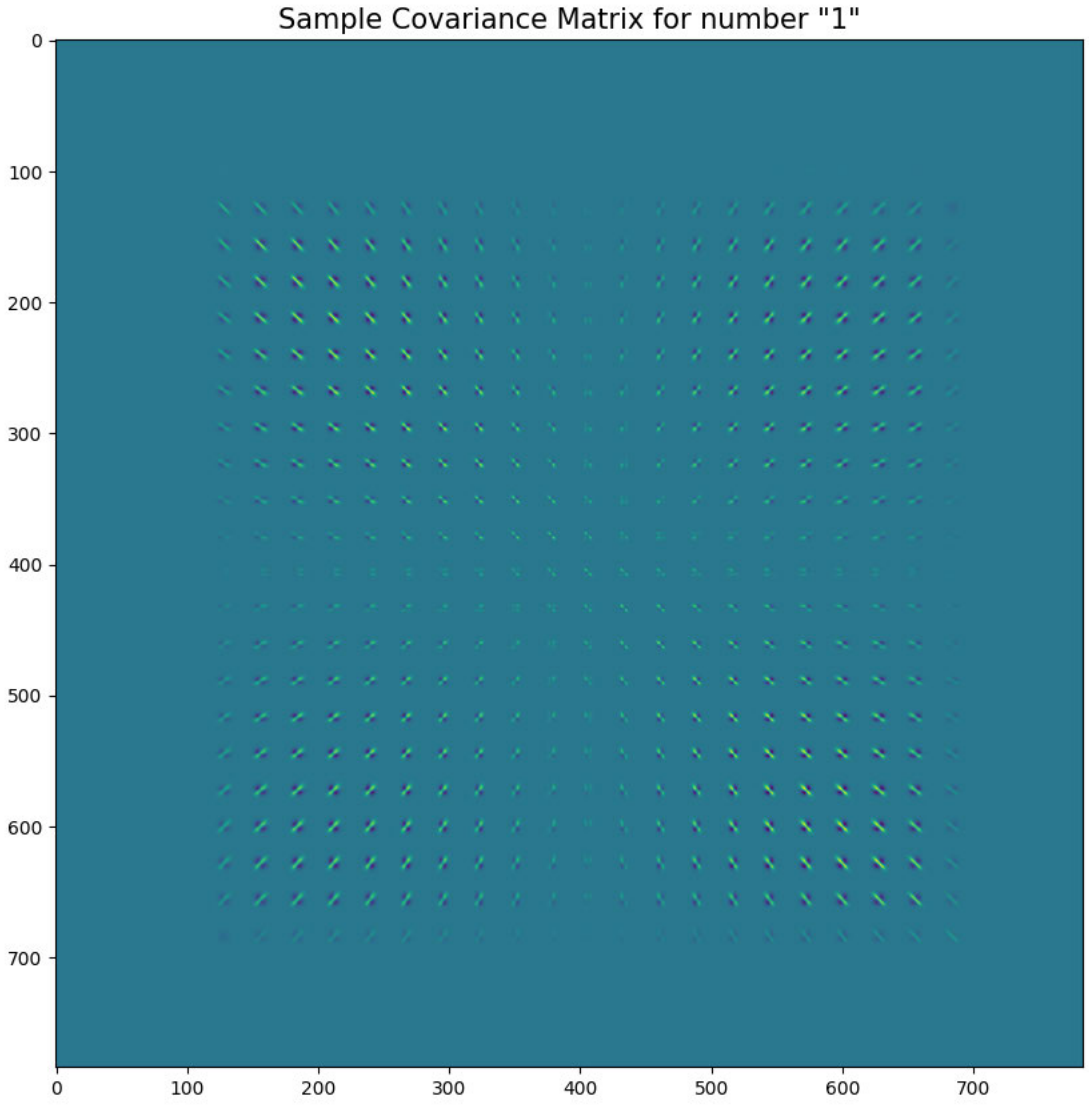
In [148]:
```
# contrast-normalize by the 2-norm of the pixels
for i in range(mnist_data.shape[0]):

    mnist_data[i] = 1/(np.linalg.norm(mnist_data[i])) * mnist_data[i]
```

In [299]:
```
#'data' needs to be an array with sample rows and column features
def fit_gaussians(data, labels):

    #Initialize dictionaries to hold the mean vectors and the covariance matrices for the different number classes
    mus = {}
    covs = {}

    #Step through each class
    for i in range(10):

        #create a temporary array with just the samples for class i; each sample is a row vector of pixels
        temp = data[labels==i,:]

        #get the sample covariance matrix for class i
        covs[i] = np.cov(temp, rowvar=False)

        #Initialize a list for the class's mean vector
        means = []

        #step through each column/feature, take its mean and add it to the list; the final list is the mean vector for the class
        for j in range(temp.shape[1]):
            means += [temp[:,j].mean()]

        #Add the class mean vector to the dictionary for the class means
        mus[i] = means

    return mus, covs
```

In [301]:
```
mus, covs = fit_gaussians(mnist_data,mnist['training_labels'])
```

8.2

In [303]:
```
plt.figure(figsize=(10,10))
plt.imshow(covs[1])
plt.title('Sample Covariance Matrix for number "1"', fontsize=15)
# plt.savefig('Q8.2_plot.png', dpi=300)
```



8.3

(a)

In [162]:
```
from sklearn.model_selection import train_test_split
from math import log,e
import random
```

In [452]:
```
def get_arrays(data,labels):

    # get the class means and the class covariance matrices
    mus, covs = fit_gaussians(data,labels)

    #Make a list of class priors calculate by dividing the number of class instances in the labels by the # of samples
    priors = []
    for i in range(10):

        priors += [sum(labels==i)/len(labels)]

    return mus, covs, priors
```

In [459]:
```
def get_pooled_inv(covs,labels):

    #calculate the class-pooled covariance matrix
    temp_matrix = np.zeros(covs[1].shape,dtype=float)
    for i in covs.keys():
        temp_matrix += covs[i]

    pooled_cov = 1/len(labels)*temp_matrix

    #Diagonally load pooled_cov to make it pd
    approx_cov = (1/len(labels)**4)*np.identity(covs[1].shape[0]) + pooled_cov

    #Get the inverse pooled pd covariance matrix
    inv_cov = np.linalg.inv(approx_cov)

    return inv_cov
```

In [460]:
```
def lda_model(training_data, training_labels, validation):

    mus, covs, priors = get_arrays(training_data, training_labels)

    inv_cov = get_pooled_inv(covs,training_labels)

    preds = []

    for i in range(validation.shape[0]):

        class_scores = []

        for j in range(len(priors)):

            proj = np.matmul(np.array(mus[j]),inv_cov)

            class_scores += [np.matmul(proj,validation[i])-0.5*np.matmul(proj,mus[j]) + log(priors[j])]

        preds+=[class_scores.index(max(class_scores))]

    return preds
```

In [548]:
```
def get_error(predictions, labels):
    score = 0

    dig_scores = dict.fromkeys(np.linspace(0,9,10,dtype=int),0)

    for i in range(len(predictions)):
        if predictions[i] == labels[i]:
            score+=1
            dig_scores[labels[i]]+=1

    dig_errors = {}
    for key in dig_scores.keys():

        dig_errors[key] = 1-dig_scores[key]/sum(labels==key)

    return (1 - (score/len(predictions))), dig_errors
```

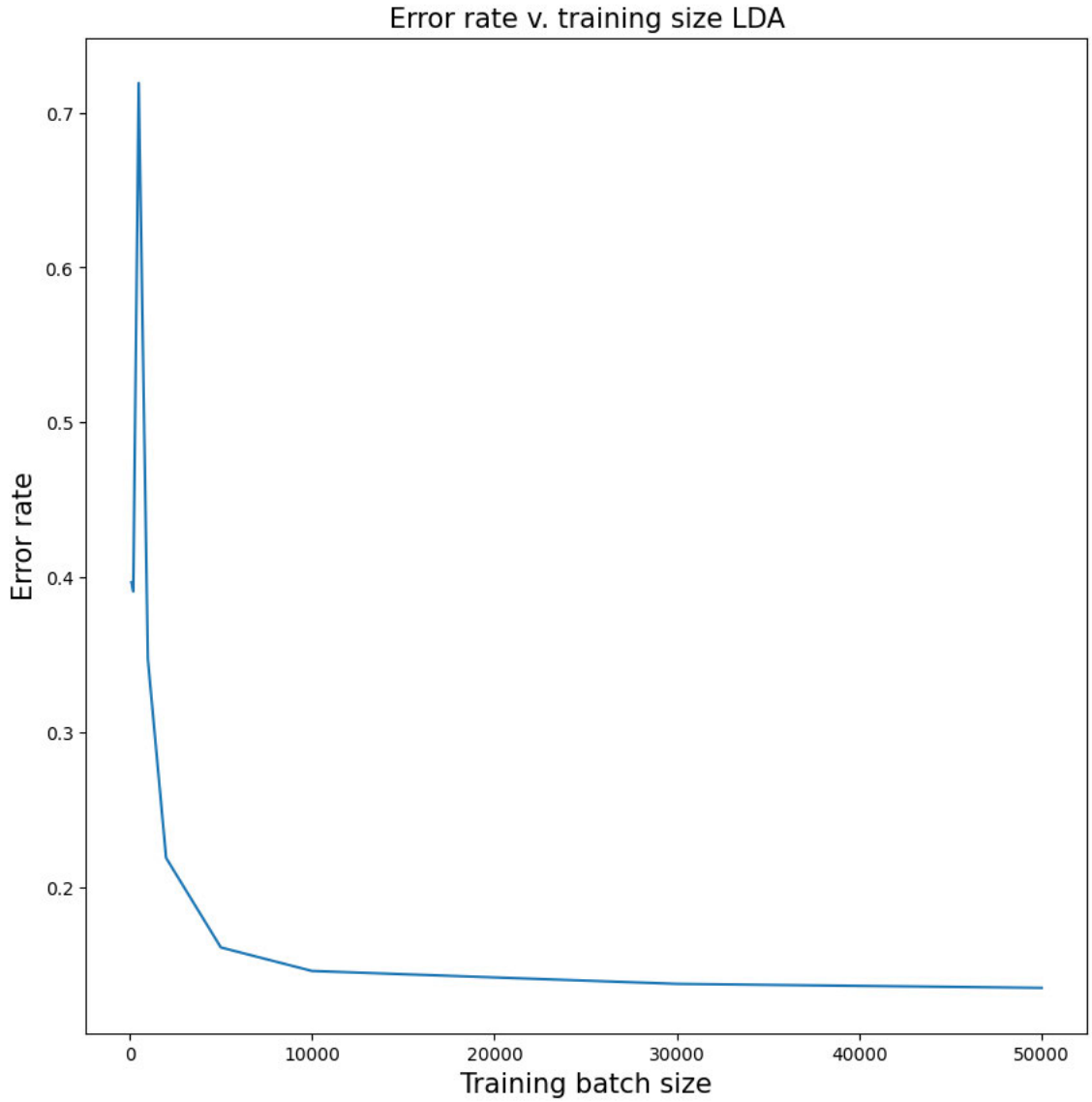
In [379]:
```
m_train, m_val, ml_train, ml_val = train_test_split(mnist_data,
                                                    mnist['training_labels'], test_size=1/6, random_state=42)
```

In [461]:
```
test = lda_model(m_train[:100],ml_train[:100], m_val)
```

In [416]:
```
training_n = [100,200,500,1000,2000,5000,10000,30000,50000]
```

In [549]:
```
# reset the random seed
random.seed(42)

# train on the different training set sizes
errors = []
dig_errors = []

for n in training_n:
    rlist=random.sample(range(0, len(m_train)), n)

    temp_train = m_train[rlist]
    temp_labels = ml_train[rlist]

    preds = lda_model(temp_train, temp_labels, m_val)

    model_error, dig_error = get_error(preds, ml_val)

    errors +=[model_error]
    dig_errors +=[dig_error]
```

In [550]:
```
plt.figure(figsize=(10,10))
plt.plot(training_n, errors)
plt.title('Error rate v. training size LDA', fontsize=15)
plt.xlabel('Training batch size', fontsize=15)
plt.ylabel('Error rate', fontsize=15)
# plt.savefig('Q8.3_plot.png', dpi=300)
```



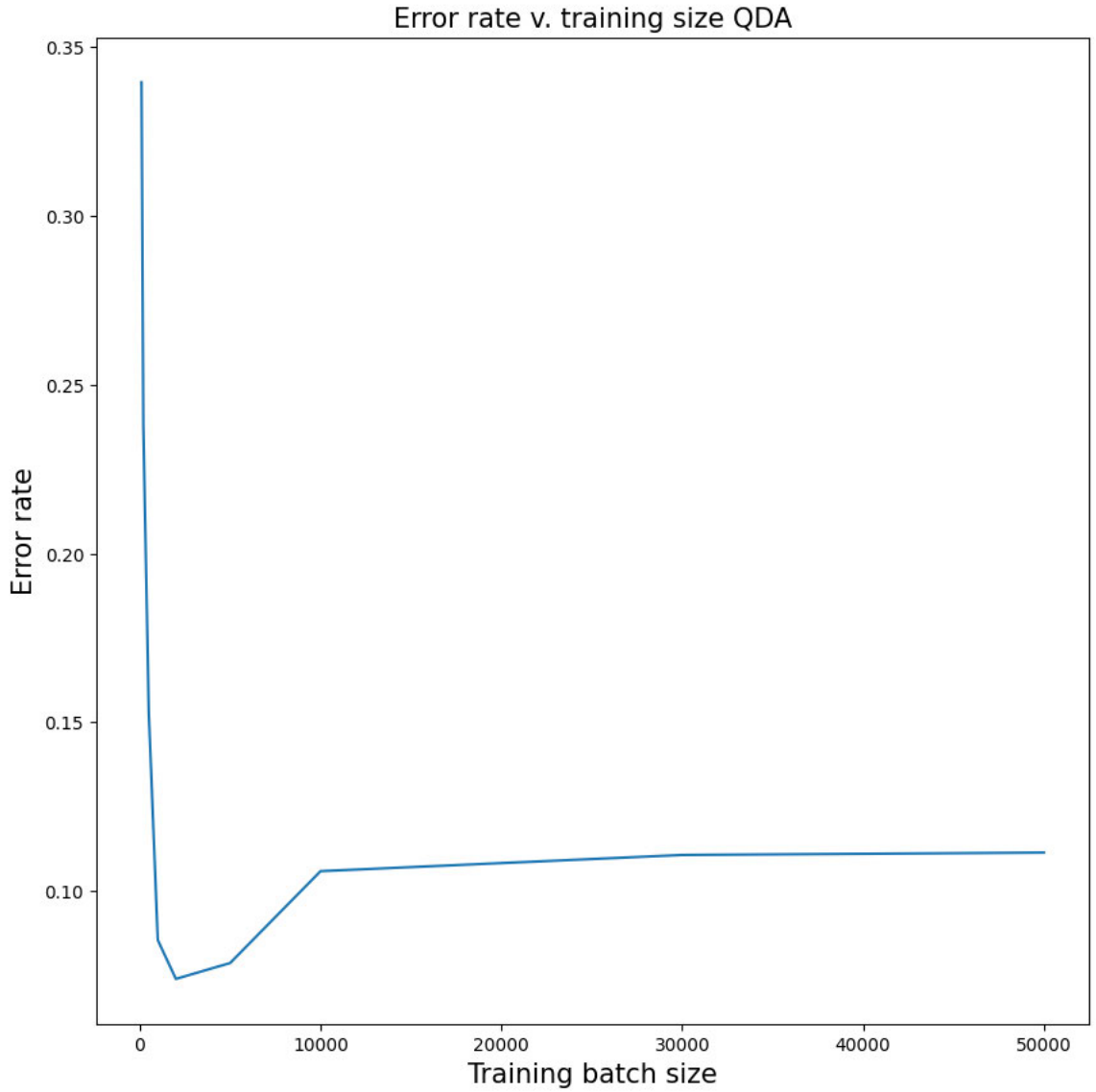
(b)

In [512]:
```
def diag_loading(covs,labels):

    #initialize libraries for the 'kludged' covariance matrices and their inverses for the different classes
    k_covs={}
    inv_k_covs={}

    for i in range(len(covs)):

        approx_cov = (1e-3*np.linalg.eig(covs[i])[0][0])*np.identity(covs[1].shape[0]) + covs[i]

        k_covs[i] = approx_cov

        inv_k_covs[i] = np.linalg.inv(approx_cov)

    return k_covs, inv_k_covs
```

In [496]:
```
def qda_model(data,labels,validation):

    mus, covs, priors = get_arrays(data, labels)

    covs, inv_covs = diag_loading(covs, labels)

    # Make a dictionary of the logs of the determinants of the covariance matrices so the calculations are done once
    dets = {}
    for i in range(len(covs)):
        dets[i] = np.linalg.slogdet(covs[i])

    preds = []

    for i in range(validation.shape[0]):

        class_scores = []

        for j in range(len(priors)):

            diff = validation[i] - mus[j]

            class_scores += [-0.5*np.matmul(np.matmul(diff,inv_covs[j]),diff) -0.5*dets[j][1] + log(priors[j])]

        preds+=[class_scores.index(max(class_scores))]

    return preds
```

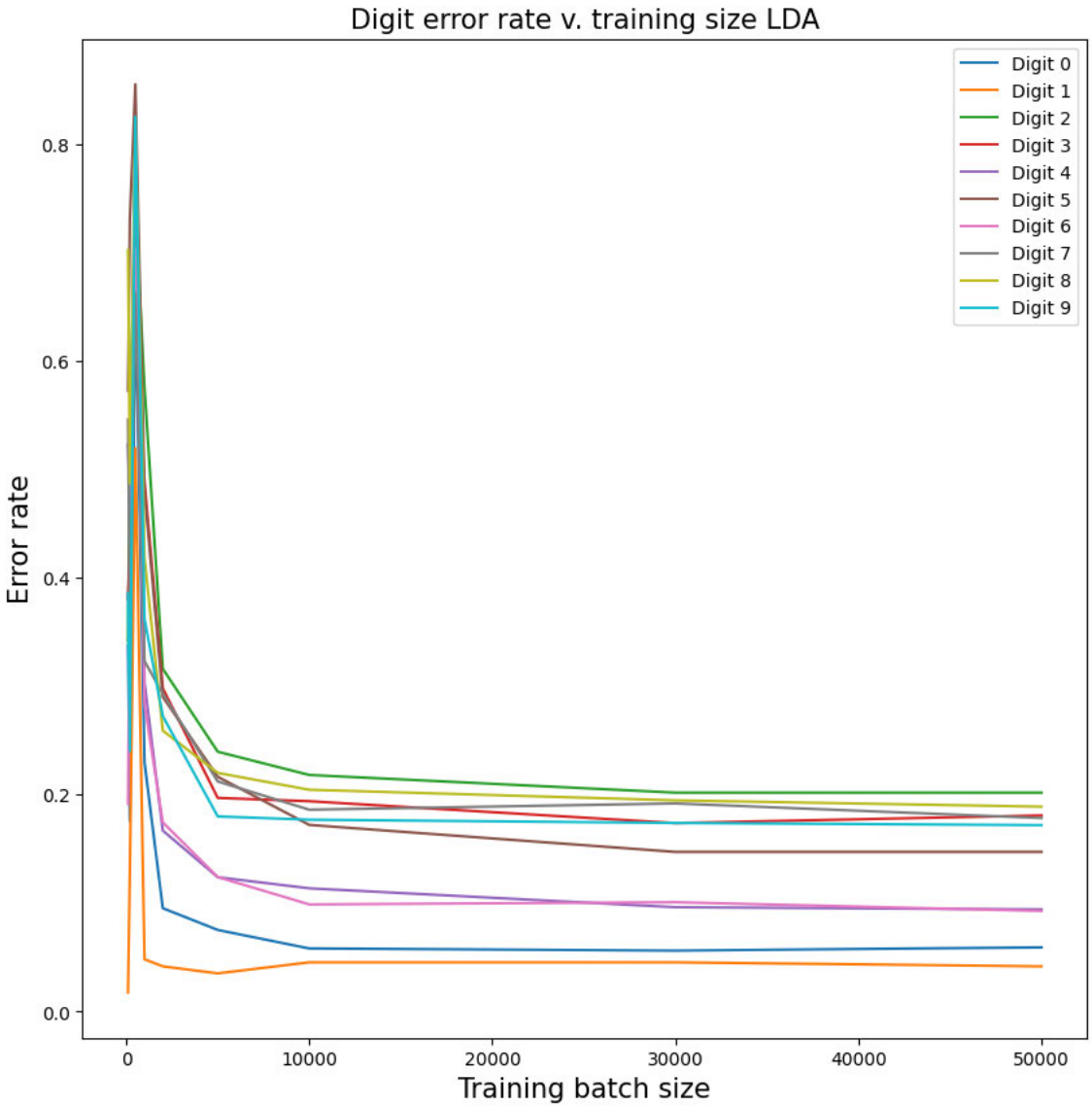
In [572]:
```
# reset the random seed
random.seed(42)

# train on the different training set sizes
qda_errors = []
qda_dig_errors = []

for n in training_n:
    rlist=random.sample(range(0, len(m_train)), n)

    temp_train = m_train[rlist]
    temp_labels = ml_train[rlist]

    preds = qda_model(temp_train, temp_labels, m_val)

    model_error, dig_error = get_error(preds, ml_val)

    qda_errors +=[model_error]
    qda_dig_errors +=[dig_error]
```

In [514]:
```
plt.figure(figsize=(10,10))
plt.plot(training_n, qda_errors)
plt.title('Error rate v. training size QDA', fontsize=15)
plt.xlabel('Training batch size', fontsize=15)
plt.ylabel('Error rate', fontsize=15)
# plt.savefig('Q8.3b_plot.png', dpi=300)
```

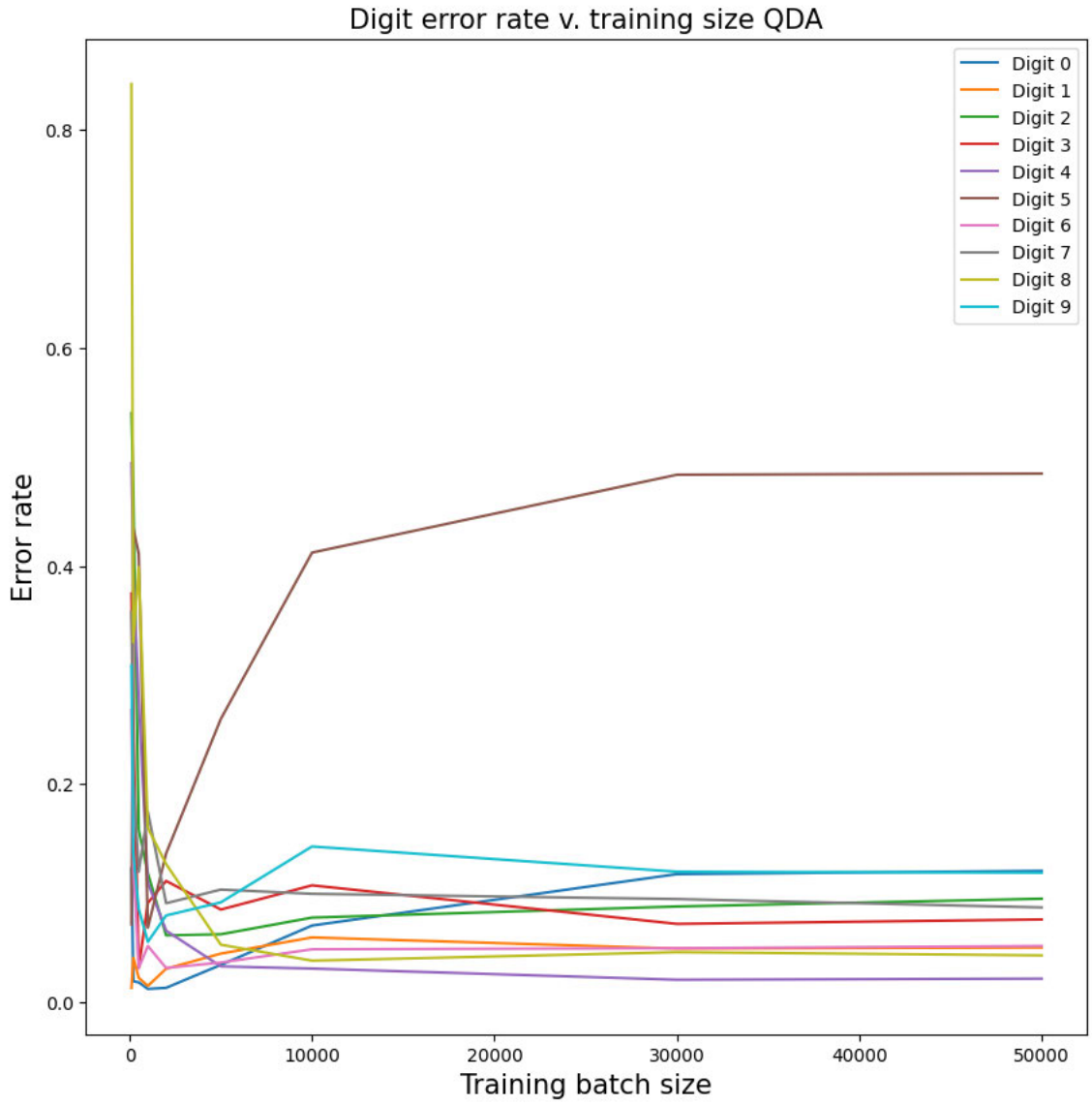


(d)

In [571]:
```
plt.figure(figsize=(10,10))
plt.plot(training_n, [dig_errors[i][0] for i in range(len(dig_errors))], label='Digit 0')
plt.plot(training_n, [dig_errors[i][1] for i in range(len(dig_errors))], label='Digit 1')
plt.plot(training_n, [dig_errors[i][2] for i in range(len(dig_errors))], label='Digit 2')
plt.plot(training_n, [dig_errors[i][3] for i in range(len(dig_errors))], label='Digit 3')
plt.plot(training_n, [dig_errors[i][4] for i in range(len(dig_errors))], label='Digit 4')
plt.plot(training_n, [dig_errors[i][5] for i in range(len(dig_errors))], label='Digit 5')
plt.plot(training_n, [dig_errors[i][6] for i in range(len(dig_errors))], label='Digit 6')
plt.plot(training_n, [dig_errors[i][7] for i in range(len(dig_errors))], label='Digit 7')
plt.plot(training_n, [dig_errors[i][8] for i in range(len(dig_errors))], label='Digit 8')
plt.plot(training_n, [dig_errors[i][9] for i in range(len(dig_errors))], label='Digit 9')
plt.title('Digit error rate v. training size LDA', fontsize=15)
plt.xlabel('Training batch size', fontsize=15)
plt.ylabel('Error rate', fontsize=15)
plt.legend()
# plt.savefig('Q8.3d_plot_LDA.png', dpi=300)
```



In [573]:
```
plt.figure(figsize=(10,10))
plt.plot(training_n, [qda_dig_errors[i][0] for i in range(len(qda_dig_errors))], label='Digit 0')
plt.plot(training_n, [qda_dig_errors[i][1] for i in range(len(qda_dig_errors))], label='Digit 1')
plt.plot(training_n, [qda_dig_errors[i][2] for i in range(len(qda_dig_errors))], label='Digit 2')
plt.plot(training_n, [qda_dig_errors[i][3] for i in range(len(qda_dig_errors))], label='Digit 3')
plt.plot(training_n, [qda_dig_errors[i][4] for i in range(len(qda_dig_errors))], label='Digit 4')
plt.plot(training_n, [qda_dig_errors[i][5] for i in range(len(qda_dig_errors))], label='Digit 5')
plt.plot(training_n, [qda_dig_errors[i][6] for i in range(len(qda_dig_errors))], label='Digit 6')
plt.plot(training_n, [qda_dig_errors[i][7] for i in range(len(qda_dig_errors))], label='Digit 7')
plt.plot(training_n, [qda_dig_errors[i][8] for i in range(len(qda_dig_errors))], label='Digit 8')
plt.plot(training_n, [qda_dig_errors[i][9] for i in range(len(qda_dig_errors))], label='Digit 9')
plt.title('Digit error rate v. training size QDA', fontsize=15)
plt.xlabel('Training batch size', fontsize=15)
plt.ylabel('Error rate', fontsize=15)
plt.legend()
# plt.savefig('Q8.3d_plot_QDA.png', dpi=300)
```



8.4

In [579]:
```
#Need to reshape the data to have the images be row vectors
test=mnist['test_data'].reshape(10000,784)
```

In [580]:
```
rlist=random.sample(range(0, len(m_train)), 200)
temp_train = m_train[rlist]
temp_labels = ml_train[rlist]
```

In [582]:
```
kaggle_preds = qda_model(temp_train, temp_labels, test)
```

In [585]:
```
df = pd.DataFrame({'Id': np.linspace(1,10000,10000,dtype=int), 'Category': kaggle_preds},dtype=np.int64)
df.to_csv('mnist_preds.csv',index=False)
```

8.5

In [588]:
```
spam = np.load('data/spam-data-hw3.npz')
```

In [589]:
```
spam.files[]
```

Out[589]:
```
['training_data', 'training_labels', 'test_data']
```

In [590]:
```
import os
```

In [591]:
```
def get_freqs(path):

    files = os.listdir(path)
    word_freqs = {}

    for file in files:
        with open('{}/{}'.format(path,file), encoding='utf8', errors='ignore') as f:
            try:
                lines = f.readlines() # Read in text from file
            except Exception as e:
                    # skip files we have trouble reading.
                continue

        for i in range(len(lines)):
            words=lines[i].split()

            for word in words:
                if word not in word_freqs.keys():
                    word_freqs[word] = 1
                elif word in word_freqs.keys():
                    word_freqs[word] += 1

    word_freqs = sorted(word_freqs.items(), key=lambda x:x[1],reverse=True)
    return word_freqs
```

In [597]:
```
spam_freqs=get_freqs('data/spam')
```

In [598]:
```
ham_freqs=get_freqs('data/ham')
```

In [599]:
```
spam_words=[spam_freqs[:200][i][0] for i in range(200)]
spam_freqs=[spam_freqs[:200][i][1] for i in range(200)]
```

In [600]:
```
ham_words=[ham_freqs[:200][i][0] for i in range(200)]
ham_freqs=[ham_freqs[:200][i][1] for i in range(200)]
```

In [601]:
```
# add these all as features
additions=list(set(spam_words)^set(ham_words))
```

In [602]:
```
additions
```

```
'20',
'more',
'change',
'free',
'today',
'11',
'statements',
'software',
'without',
'over',
'office',
'>',
'daily',
'o',
'texas',
'69',
'april',
'products',
'cc',
'robert',
```

In [611]:
```
for word in additions:
    print('def freq_{}_feature(text, freq):\n    return float(freq[\'{}\'])'.format(word,word))
```

```
    return float(freq['hpl'])
def freq_style_feature(text, freq):
    return float(freq['style'])
def freq_nd_feature(text, freq):
    return float(freq['nd'])
def freq_energy_feature(text, freq):
    return float(freq['energy'])
def freq_volume_feature(text, freq):
    return float(freq['volume'])
def freq_than_feature(text, freq):
    return float(freq['than'])
def freq_b_feature(text, freq):
    return float(freq['b'])
def freq_http_feature(text, freq):
    return float(freq['http'])
def freq_prices_feature(text, freq):
    return float(freq['prices'])
def freq_adobe_feature(text, freq):
    return float(freq['adobe'])
```

In [612]:
```
for word in additions:
    print('feature.append(freq_{}_feature(text, freq))'.format(word))
```

```
feature.append(freq_microsoft_feature(text, freq))
feature.append(freq_50_feature(text, freq))
feature.append(freq_size_feature(text, freq))
feature.append(freq_forward_feature(text, freq))
feature.append(freq_2001_feature(text, freq))
feature.append(freq_best_feature(text, freq))
feature.append(freq_align_feature(text, freq))
feature.append(freq_only_feature(text, freq))
feature.append(freq_forwarded_feature(text, freq))
feature.append(freq_contract_feature(text, freq))
feature.append(freq_31_feature(text, freq))
feature.append(freq_ticket_feature(text, freq))
feature.append(freq_its_feature(text, freq))
feature.append(freq_g_feature(text, freq))
feature.append(freq_xp_feature(text, freq))
feature.append(freq_let_feature(text, freq))
feature.append(freq_go_feature(text, freq))
feature.append(freq_xls_feature(text, freq))
feature.append(freq_other_feature(text, freq))
feature.append(freq_www_feature(text, freq))
```