Course: Introduction to Machine Learning

Assigment 4

Important NOTE: In order to get full credit, for every question, you need to provide the details of your work on how to get to a solution or the end of a proof.

Instructor: Tan Bui-Thanh

TA: Cole Nockolds

Due day: 11:59 pm, 29 October, Friday

All submission MUST be in pdf format, except codes. All code outputs should be reported in pdf format.

Question 1 (30 points) Probalistic linear regression

Recall: The multivariate Gaussian distribution has the form $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, $\boldsymbol{\mu} \in \mathbb{R}^n, \Sigma \in \mathbb{R}^{n \times n}$

$$\mathcal{N}(\boldsymbol{\mu}, \Sigma) = (2\pi)^{\frac{-n}{2}} \det(\Sigma)^{\frac{-1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu})\right) \propto \underbrace{\exp\left(-\frac{1}{2}(\boldsymbol{x} -}$$

($\propto$ is to ignore the normalization constant).

To find the mean $\boldsymbol{\mu}$, we take first derivative of $-\log(F)$ and solve for $\boldsymbol{x}$, i.e.,

$$\frac{d}{d\boldsymbol{x}}[-\log(F)] = \mathbf{0} \implies \boldsymbol{x} = \boldsymbol{\mu}$$

To find the covariance matrix, Σ, we take the second derivative of $-\log(F)$, which returns Σ^{-1}, then invert the matrix to obtain Σ.

$$\Sigma = \left[\frac{d^2}{d\boldsymbol{x}^2}[-\log(F)]\right]^{-1}$$

By these procedures, we can find the mean $\boldsymbol{\mu}$ and the covariance matrix Σ of original multivariate Gaussian distribution. In the case that the distribution is not directly the form of Gaussian distribution, for example,

$$F = \exp\left(-\frac{1}{2}\left(\boldsymbol{x}^T(A^TA)\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{x}\right)\right)$$

following the same procedure, you would find the mean is $\mathbf{0}$, and the covariance matrix is $(A^TA + I)^{-1}$.

Now, generate 6 data samples for the problem using the polynomial function (use the below given code for create $x^{(i)}$)

$$f(x) = .5 - x - .5x^2 - 2x^3 + 5x^4.$$

Then add noise: $y^{(i)} = f(x^{(i)}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \delta^2)$, and noise level $\delta = 0.2$. In this problem, we assume that the model is 2nd order, i.e., PolynomialFeatures(degree=2) in scikit-learn. \

1. Using a Bayesian approach, find, analytically, the mean and covariance of the posterior distribution $P(y|x, \mathcal{D})$, where, as in lecture 5b, we have

$$P(y|x, \mathcal{D}) = \int_\theta P(y \mid x, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta}.$$

   where

- $P(y \mid x, \boldsymbol{\theta})$ is the likelihood distribution (chosen to be the set of a Gaussian distributions as in lecture 5b)

- $P(\boldsymbol{\theta} \mid \mathcal{D})$ is the posterior distribution given the training data, as in lecture 5a, we have

$$P(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \boldsymbol{\theta}) P(\boldsymbol{\theta})}{P(\mathcal{D})}$$

  in which $P(\boldsymbol{\theta})$ is the Gaussian prior with mean zero and variance σ^2.

Hints:

- Show that the distribution $P(\boldsymbol{\theta} \mid \mathcal{D}) \sim \mathcal{N}(\boldsymbol{\theta}_{\mathrm{MAP}}, \Sigma_{\boldsymbol{\theta}})$ has the form of Gaussian distribution.
- Apply the multivariate Gaussian distribution formula to find $\boldsymbol{\theta}_{\mathrm{MAP}}, \Sigma_{\boldsymbol{\theta}}$.
- Find the distribution $P(y|x, \mathcal{D})$ by marginalizing the $\boldsymbol{\theta}$ random variable.

2. Plot the the uncertainty band with the range of $x \in [0, 1]$.
3. Discuss the benefits of a Bayesian method as opposed to maximum likelihood.

1.1 - derivation of P(y|x, D)

Anushi R. Sadam
Professor Tan Bui Thanh
COE 379L: Introduction to Machine Learning & Data Scienc
29 October 2024

Homework 4

1. Find the mean, covariance of $P(y|x, D)$ w/ Bayesian approach.

$$P(y|x,D) = \int_\theta P(y|x,\theta)\, P(\theta|D)\, d\theta$$

$$P(\theta|D) = \frac{P(D|\theta)\cdot P(\theta)}{P(D)}$$

we assume the prior $P(\theta) = N(0, \sigma^2 I_d)$ (mean 0, variance σ^2), since we have no info about the prior in our test problem. d is the dimension of θ; $\theta \in \mathbb{R}^d$.

$P(D|\theta)$ is the likelihood based on the dataset $D = \{(x^i, y^i)\}_{i=1}^n$ ~ number of samples $= |D|$

$$P(D|\theta) = \prod_{i=1}^{n} P(x^i, y^i|\theta)$$

we know $P(x, y|\theta) = P(y|x,\theta)\cdot P(x|\theta)$

Since $x \perp \theta$ (x is independent of θ), $P(x|\theta) = P(x)$. $P(x)$ is an uniform distribution since we must assume our data collection evenly covers the system.

We assume a gaussian conditional distribution:

$$P(y^i | x^i, \theta) = N(\theta^T x^i, \sigma^2)$$

(x^i: feature vector)

or: (n data samples)

$$P(y | x, \theta) = \prod_{i=1}^{n} N(\theta^T x^i, \sigma^2)$$

(σ^2: variance) assume same noise level for all measurements.

Thus, $P(x, y | \theta) \propto P(y | x, \theta)$

(x: feature matrix, vectors; n: number of data points)

$$\propto \prod_{i=1}^{n} \exp\left(\frac{-1}{2\sigma^2} - (y^i - \theta^T x^i)^2\right)$$

$P(x, y | \theta) = P(y | x, \theta) \cdot P(x | \theta)$

mean of $P(y^i | x^i, \theta) = \theta^T x^i$: proven in HW 2.

Since x is independent of θ, $P(x|\theta)$ is $P(x)$.

So, since $P(x)$ is a uniform distribution,

$P(x, y | \theta) \propto P(y | x, \theta)$

normal distribution assumed by our model.

$$P(D|\theta) = P(x, y | \theta) \propto \prod_{i=1}^{n} \exp\left(\frac{-1}{2\sigma^2} (y^i - \theta^T x^i)^2\right)$$

We assume $P(\theta)$ is a gaussian distribution w/ mean $\vec{0}$, variance of each element λ^2:

Now, we assume a prior of mean 0, variance λ^2 IO (along all dimensions)

in general:

$$P(\vec{\theta}) = \frac{1}{\sqrt{(2\pi)^{|D|} \cdot |\Sigma|^{1/2}}} \exp\left(-\frac{1}{2}(\vec{\theta}-\vec{\mu})^T \Sigma^{-1} (\vec{\theta}-\vec{\mu})\right)$$

where $|D|$ is the number of dimensions for $\vec{\theta}$,

Σ is the covariance matrix,

μ is the mean of $P(\vec{\theta})$.

We know $\mu = \vec{0}$. Now to calculate Σ:

$$\Sigma_{ij} = \text{cov}(\theta_i, \theta_j) \quad \forall i,j \in [1, |D|]$$

$$= \mathbb{E}[(\theta_i - \mathbb{E}[\theta_i])(\theta_j - \mathbb{E}[\theta_j])]$$

$\mathbb{E}[\theta_i] = \mathbb{E}[\theta_j] = 0$ (mean is $\vec{0}$)

$\Rightarrow \Sigma_{ij} = \mathbb{E}[\theta_i \times \theta_j]$

1) $\forall i = j$:

$\Sigma_{ii} = \mathbb{E}[\theta_i^2]$

we know $\text{Var}(\theta_i) = \mathbb{E}[\theta_i^2] - (\underbrace{\mathbb{E}[\theta_i]}_{0})^2 = \lambda^2$

$\Rightarrow \text{Var}(\theta_i) = \mathbb{E}[\theta_i^2] = \lambda^2$

for each θ_i, variance is a constant, λ^2

Thus, $\Sigma_{ii} = \lambda^2$

2) $\forall i \neq j$

$\Sigma_{ij} = \mathbb{E}[\theta_i \theta_j]$

we assume independent variables for the coefficients θ_i (random)

so, $\Sigma_{ij} = \mathbb{E}[\theta_i] \cdot \mathbb{E}[\theta_j]$

$= \mu_i \cdot \mu_j$

(μ_i: mean of θ_i distribution; μ_j: mean of θ_j distribution)

since we assume random normal distributions of mean 0 ($\mu_i = 0$, $\mu_j = 0$),

$\Sigma_{ij} = 0 \cdot 0 = 0$.

Thus, $\Sigma = \begin{bmatrix} \lambda^2 & & 0 \\ & \lambda^2 & \\ 0 & & \ddots \lambda^2 \end{bmatrix} = \lambda^2 I_{|D|}$

($I_{|D|}$: identity of dimension $|D| \times |D|$)

$\det(\Sigma) = (\lambda^2)^{|D|}$

$\Sigma^{-1} = (\lambda^2 I_{|D|})^{-1} = \frac{1}{\lambda^2} I_{|D|}$

Note $P(\theta|D) = \frac{P(D|\theta) \cdot P(\theta)}{P(D)}$

$P(D)$ is simply a normalization value used so that the total probability $\int_D P(\theta|D)\, dD = 1$.

($\Rightarrow P(\theta|D)$ is a bona fide probability distribution)

Then,

$$P(\theta|D) \propto P(D|\theta) \cdot P(\theta)$$

$$= \prod_{i=1}^{n} \exp\left(\frac{-1}{2\delta^2} (y^i - \theta^T x^i)^2\right) \cdot$$

$$\exp\left(\frac{-1}{2\lambda^2} \|\theta\|_2^2\right)$$

$$= \exp\left(-\frac{1}{2}\left(\sum_{i=1}^{n} \left[\frac{1}{\delta^2} (y^i - \theta^T x^i)^2\right] + \frac{1}{\lambda^2} \|\theta\|_2^2\right)\right)$$

$P(\theta|D)$.

we seek to maximize ^ (feature matrix)

$$P(\theta|D) \propto \exp\left(-\frac{1}{2}\left[(y-x\theta)^2 \cdot \frac{1}{\delta^2} + \frac{1}{\lambda^2}\theta^T\theta\right]\right)$$

$$\propto \exp\left(-\frac{1}{2}\left[\frac{1}{\delta^2}\left(y^Ty - 2y^Tx\theta + \theta^Tx^Tx\theta\right) + \frac{1}{\lambda^2}(\theta^T\theta)\right]\right)$$

$$\propto \exp\left(-\frac{1}{2}\left(\theta^T\underbrace{\left(\frac{x^Tx}{\delta^2} + \frac{I}{\lambda^2}\right)}_{A}\theta - 2\underbrace{\frac{y^Tx}{\delta^2}}_{b^T}\theta + \underbrace{\frac{y^Ty}{\delta^2}}_{c}\right)\right)$$

let $A = \left[\frac{x^Tx}{\delta^2} + \frac{I}{\lambda^2}\right]$ & $b: \frac{x^Ty}{\delta^2}$ & $C = \frac{y^Ty}{\delta^2}$

then,

$$\frac{-1}{2}\left(\theta^TA\theta - 2b^T\theta + C\right) = \frac{-1}{2}\left(\theta^TA\theta - 2b^T\theta + b^TA^{-1}b - b^TA^{-1}b + C\right)$$

$$= \frac{-1}{2}\left((\theta - A^{-1}b)^TA(\theta - A^{-1}b) - b^TA^{-1}b + C\right)$$

F Aside:

$$\theta^TA\theta - 2b^T\theta = \left(\theta^TA\theta - 2b^T\theta + b^TA^{-1}b\right) - b^TA^{-1}b - 2b\theta$$

$$\theta^TA\theta - 2b^T\theta + b^TA^{-1}b = (\theta - A^{-1}b)^TA(\theta - A^{-1}b)$$

since:

LHS expansion:

$$(\theta^T)\ A\theta - (\theta^T)\ A\cdot A^{-1}b - (A^{-1}b)^TA\theta + (A^{-1}b)^TA(A^{-1}b)$$

$(A^{-1}b)^T A = b^T (A^{-1})^T A.$

Note since $I = A^{-1}A = AA^{-1}$, $I^T = (A^{-1}A)^T = (AA^{-1})^T$

$I = I^T = (AA^{-1})^T = (A^{-1})^T A^T$

Note that A is a symmetric matrix $A = mX^TX + nI$ where m, n are constants

$$A^T = (mX^TX) + nI)^T = (nI)^T + (mX^TX)^T$$

$$= nI^T + m(X^TX)^T = nI + m\, X^T(X^T)^T$$
$$= nI + mX^TX = mX^TX + nI = A.$$

Thus, A is symmetric.

So, $I = (A^{-1})^T A^T = (A^{-1})^T A.$

Then, $(A^{-1}b)^T A = b^T I = b^T.$

Continuing the LHS expansion:

$$\theta^T A\theta - \theta^T b - b^T\theta + (b^T \underbrace{(A^{-1})^T A}_{I} A^{-1} b)$$

$$= \theta^T A\theta - 2b^T\theta + b^T(A^{-1})b \quad 7$$

Thus, $P(\theta|D) \propto \exp\left[-\frac{1}{2}\left((\theta - A^{-1}b)^T A(\theta - A^{-1}b) - b^TA^{-1}b + c\right)\right]$

continuing w/ our exponential factoring:

let $F = (b^TA^{-1}b + c)\frac{1}{2}$

Then in the exponent, we have

$$-\frac{1}{2}(\theta - A^{-1}b)^T A (\theta - A^{-1}b) + F.$$

so:

$$P(\theta|D) \propto \exp\left(-\frac{1}{2}(\theta - A^{-1}b)^T A(\theta - A^{-1}b) + F\right)$$

$$\propto \exp\left(-\frac{1}{2}\left((\theta - A^{-1}b)^T A(\theta - A^{-1}b)\right)\right) \cdot e^F$$

constant b.

$$\propto \exp\left(-\frac{1}{2}(\theta - A^{-1}b)^T A(\theta - A^{-1}b)\right)$$

multivariate Gaussian expression

w/ mean $\theta^* = A^{-1}b$, $\Sigma_0 = A^{-1}$.

Thus, $P(\theta|D) \propto \exp\left(\frac{-1}{2}(\theta - \theta^*)^T \Sigma_0^{-1}(\theta - \theta^*)\right)$

where $\theta^* = \left[\frac{X^TX}{\delta^2} + \frac{I}{\lambda^2}\right]^{-1} \frac{X^Ty}{\delta^2} = \left[X^TX + I\frac{\delta^2}{\lambda^2}\right]^{-1} X^Ty$

$$\Sigma_0 = \left[\frac{X^TX}{\delta^2} + \frac{I}{\lambda^2}\right]^{-1} = \delta^2\left[X^TX + \frac{\delta^2}{\lambda^2}I\right]^{-1}$$

Note about scalar inverse pg 7

we recognize θ^* as a least squares solution w/ regularization $\frac{\delta^2}{\lambda^2}$

let $\hat{y} = X\theta$, then we have a least squares problem:

$\theta^* = \underset{\theta}{\text{argmin}} (y - X\theta)^T(y - X\theta) + \frac{\gamma}{2}\|\theta\|_2^2$, γ is a regularizer

The solution to the L_2 regularized solution is

loss:

$$J(\theta) = \|y - X\theta\|^2 + \frac{\gamma}{2}\|\theta\|^2.$$

$$\nabla_\theta J(\theta) = X^T X\theta - X^T y + \gamma\theta = 0.$$

$$\theta(X^T X + \gamma I) = X^T y$$

positive definate ; $\theta^*_{MAP} = (X^T X + \gamma I)^{-1} X^T y$.

Thus, for $P(\theta|D)$, $\theta^* = \left(X^T X + \frac{\delta^2}{\lambda^2} I\right)^{-1} X^T y$

which is simply the MAP solution for θ^* via least squares, w/ regularizer $\gamma = \frac{\delta^2}{\lambda^2}$.

Note that the mean of $P(\theta|D)$ is θ^*, since

$$P(\theta|D) \propto \exp\left(-\tfrac{1}{2}(\theta - \theta^*)^T \Sigma_0 (\theta - \theta^*)\right)$$

has the same form as the gaussian w/ mean $\mu = \theta^*$ & covariance Σ_0.

Σ_0, thus, is: $\boxed{\Sigma_0 = \left[\frac{X^T X}{\delta^2} + \frac{I}{\lambda^2}\right]^{-1} = \delta^2\left[X^T X + \frac{\delta^2}{\lambda^2} I\right]^{-1}}$

① $(cA)^{-1} = c^{-1}A^{-1}$ $\forall\ c \in \mathbb{R}$, $A \in \mathbb{R}^{d\times d}$ $d \in \mathbb{R}$

↓ A must be invertible

Proof:

Suppose cA has inverse B:

$(cA)B = I$; $c(AB) = I$; $AB = \frac{1}{c} I$

$A^{-1}AB = \frac{1}{c}A^{-1}I$; $IB = \frac{1}{c}A^{-1}$

$\therefore B = \frac{1}{c}A^{-1} = c^{-1}A^{-1}$

Note that $P(\theta|D) = N(\theta^*_{MAP}, \Sigma_0)$; we can change from $\propto$ to $=$ since we know $P(\theta|D)$ must be a bona fide probability distribution, so, $\int_\theta P(\theta|D)\, d\theta = 1$.

If $P(\theta|D) = c \cdot N(\theta^*_{MAP}, \Sigma_0)$ for some constant (scalar) c, then,

$$\int_\theta P(\theta|D)\, d\theta = \int_{-\infty}^{\infty} c \cdot \frac{\exp\left(\frac{-1}{2}(\theta - \theta^*_{MAP})^T \Sigma_0^{-1} (\theta - \theta^*_{MAP})\right)}{(2\pi)^{d/2} |\Sigma_0|^{1/2}} \times d\theta$$

where d is the dimension of the data

we know, then $\int_\theta P(\theta|D)\, d\theta = c \cdot \int_{-\infty}^{\infty} N(\theta^*_{MAP}, \Sigma_0)\, d\theta$

per the definition of a normal distribution, we know

$$\int_{-\infty}^{\infty} N(\theta^*_{MAP}, \Sigma_0)\, d\theta = 1$$

(area under normal distribution $= 1$).

Then, $\int_\theta P(\theta|D)\, d\theta = c \cdot 1$. so, $\underline{c = 1}$.

This means $P(\theta|D) = N(\theta^*_{MAP}, \Sigma_0)$.

The coefficient $\frac{1}{(2\pi)^{d/2} |\Sigma_0|^{1/2}}$ comes from the various constants we ignored in our proportion analysis done previously. For example, one of the constants being ignored is $P(D)$ when we solve for $P(\theta|D)$. This will help allow $P(\theta|D)$ to be normalized.

p.3. Now, to find $P(y|x, D)$:

$$P(y|x, D) = \int_\theta P(y|x, \theta)\, P(\theta|D)\, d\theta$$

$$= \int_\theta \prod_{i=1}^{n} N(\theta^T x^i, \delta) \cdot N(\theta_{MAP}, \Sigma_0)\, d\theta$$

note that

$$\prod_{i=1}^{n} N(\theta^T x^i, \delta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\delta^2}} \exp\left(\frac{-1}{2\delta^2}(y^i - \theta^T x^i)^2\right)$$

$$= \frac{1}{(2\pi)^{n/2}\delta^n} \exp\left(\frac{-1}{2\delta^2}(y - X\theta)^T(y - X\theta)\right)$$

where X is the feature matrix.

So, $P(y|x, \theta) \cdot P(\theta|D) =$

$$\frac{1}{(2\pi)^{n/2}\delta^n} \cdot \frac{1}{(2\pi)^{n/2}|\Sigma_0|} \exp\Big(\underbrace{\frac{-1}{2\delta^2}(y - X\theta)^T(y - X\theta) + \frac{-1}{2}(\theta - \theta_{MAP})^T \Sigma_0^{-1}(\theta - \theta_{MAP})}_{M}\Big)$$

let M be the exponent term.

we expand M to get:

$$\frac{-1}{2\delta^2}(y^T y - 2y^T X\theta + \theta^T X^T X\theta) - \frac{1}{2}(\theta^T \Sigma_0^{-1}\theta - 2\theta^T \Sigma_0^{-1}\theta_{MAP} + \theta_{MAP}^T \Sigma_0^{-1}\theta_{MAP})$$

Grouping quadratic terms:

$$\mathcal{M} = -\frac{1}{2}\left(\theta^T \left(\frac{X^T X}{\delta^2} + \Sigma_0^{-1} \right) \theta \right.$$

$$\left. -2\theta^T \left(\frac{X^T y}{\delta^2} + \Sigma_0^{-1} \theta_{MAP} \right) \right)$$

$$\underbrace{\frac{-1}{2\delta^2} y^T y - \frac{1}{2} \theta_{MAP}^T \Sigma_0^{-1} \theta_{MAP}}_{\text{constant } C \text{ w.r.t } \theta}$$

C is a constant so, we can factor out e^C as a constant from the integral of $P(y|x, D)$.

Thus, define $Q = -\frac{1}{2}\left(\theta^T A \theta - 2\theta^T b \right)$

be the exponent in the integral,

where $A = \frac{X^T X}{\delta^2} + \Sigma_0^{-1}$ & $b = \frac{X^T y}{\delta^2} + \Sigma_0^{-1} \theta_{MAP}$.

As before, we can complete the square

to have $Q = -\frac{1}{2}\left((\theta - A^{-1}b)^T A (\theta - A^{-1}b) \right) + \frac{1}{2} b^T A^{-1} b$.

Then, we have: $e^{\downarrow} \propto N(A^{-1}b, A^{-1})$

$$P(y|x, D) = \int_\theta \frac{1}{(2\pi)^{n/2} \delta^n |\Sigma_0|} \exp(Q)\, d\theta \cdot e^C$$

now with some factoring we see:

$$P(y|x,D) = \gamma \int_\theta \underbrace{N(A^{-1}b, A^{-1})}_{\text{function of } \theta} \cdot \underbrace{e^{\frac{1}{2} b^T A^{-1} b}}_{\text{not function of } \theta, \text{ constant}} d\theta$$

where $\gamma = \frac{e^c}{(2\pi)^{\frac{n}{2}} \delta^n |\Sigma_0|}$.

then,

$$P(y|x,D) = \gamma e^{\frac{1}{2} b^T A^{-1} b} \int_\theta \underbrace{N(A^{-1}b, A^{-1})\, d\theta}_{\text{gaussian distribution integral} = 1.}$$

$$= \gamma e^{\frac{1}{2} b^T A^{-1} b}$$

No where to proceed γ should lead to our Gaussian function for the posterior predictive distribution

1.

part 3 cont. We see pure integration does not lead to reasonable results - we are stuck.

So, we instead use normal distribution properties to compute the posterior predictive distribution:

Our Bayesian regression model:

$$y = X\theta + \epsilon, \quad \epsilon \sim N(0, \sigma^2 \underset{=}{I_n})$$

$y \in \mathbb{R}^n$, number of data points.
$X \in \mathbb{R}^{n \times d}$
$\theta \in \mathbb{R}^d \quad \Rightarrow \quad P(y|x,\theta) = N(\theta^T x, \sigma^2)$

($\theta^T x = x^T \theta$)

Our posterior:

$$P(\theta|D) \sim N(\theta_{MAP}, \Sigma_0)$$

Since $P(y|x,D) = \int_\theta P(y|x,\theta) \cdot P(\theta|D)\, d\theta$

or our posterior predictive model is the convolution of 2 gaussians.

So, $P(y|x, D)$ is a gaussian distribution.

Recall the linear transformation rule on a normal distribution:

normal w.r.t y

$$y = Ax, \quad x \sim N(\mu, \Sigma), \quad P(y) = N(y \mid A\mu, A\Sigma A^T)$$

Proof:

$X \sim N(\mu, \Sigma)$, $\mathbb{E}[X] = \mu$.

$Y = AX$. $\mathbb{E}[Y] = \mathbb{E}[AX] = A\,\mathbb{E}[X] = A\mu$.

So, mean of Y, $\mathbb{E}[Y] = A\mu$.

$$Cov(X) = \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^T] = \Sigma$$

$$Cov(Y) = Cov(AX) = \mathbb{E}[(AX - \mathbb{E}[AX])(AX - \mathbb{E}[AX])^T]$$

$$= \mathbb{E}[(AX - A\mu)(Ax - A\mu)^T]$$

$$= \mathbb{E}[A(x - \mu) \cdot (A(x - \mu))^T]$$

$$= \mathbb{E}[A(x - \mu)(x - \mu)^T A^T]$$

$$= A\,\mathbb{E}[(x - \mu)(x - \mu)^T]A^T$$

$$= A \cdot \Sigma A^T.$$

Now, going back to $P(y^*|x^*, D)$: where x^*, y^* is the new point(s) being predicted:

$$\mathbb{E}[y^*|x^*, D] = \mathbb{E}[\underbrace{\mathbb{E}[y^*|x^*, \theta]}_{\text{from } P(y|x,\theta)}|D]$$

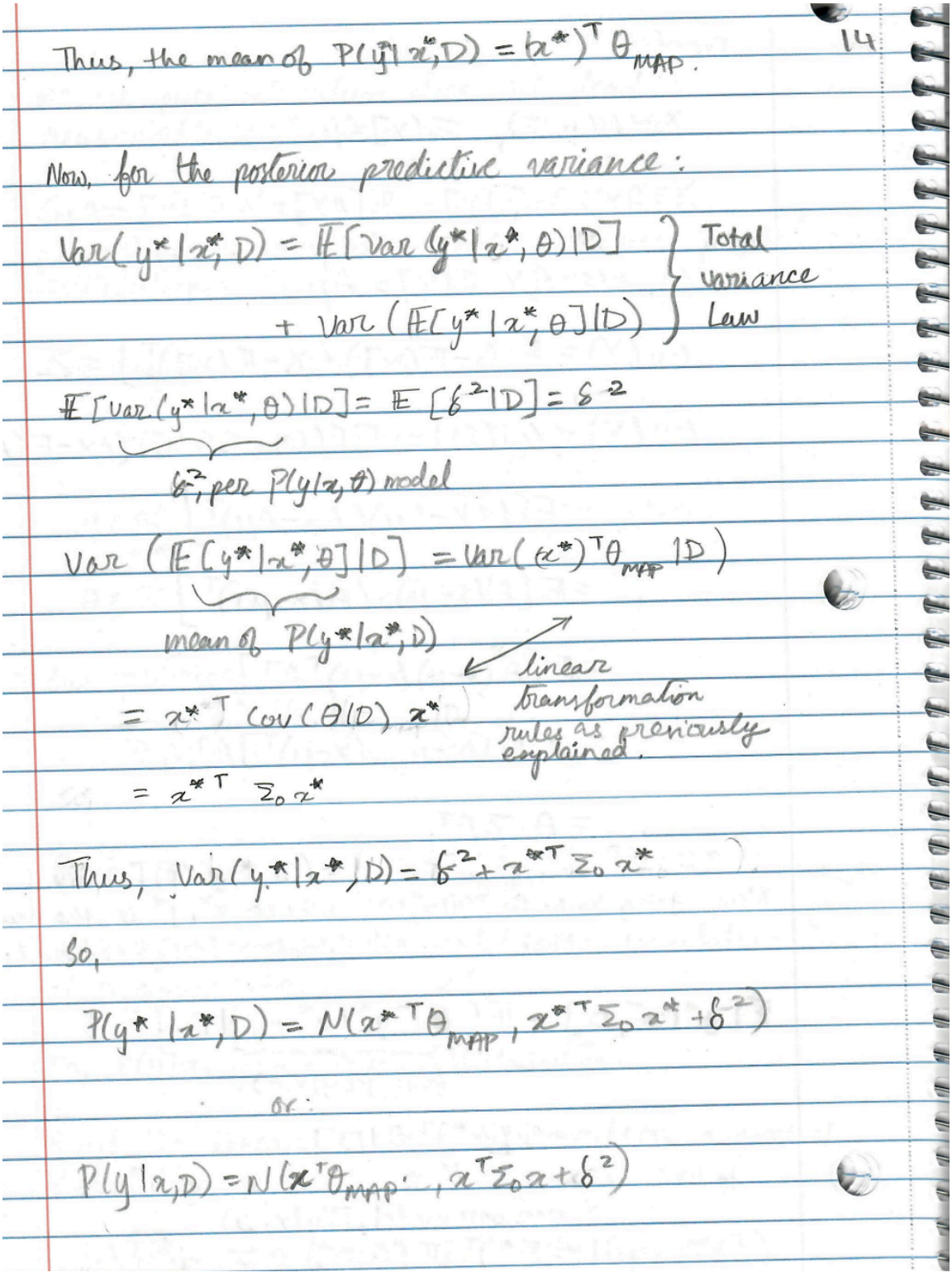
$$= \mathbb{E}[(x^*)^T \theta | D]$$

$\Downarrow$

per our model, $P(y|x, \theta)$

$$= (x^*)^T\,\mathbb{E}[\theta|D] = x^{*T}\theta_{MAP}.$$

per prior, $\Downarrow = \theta_{MAP}$



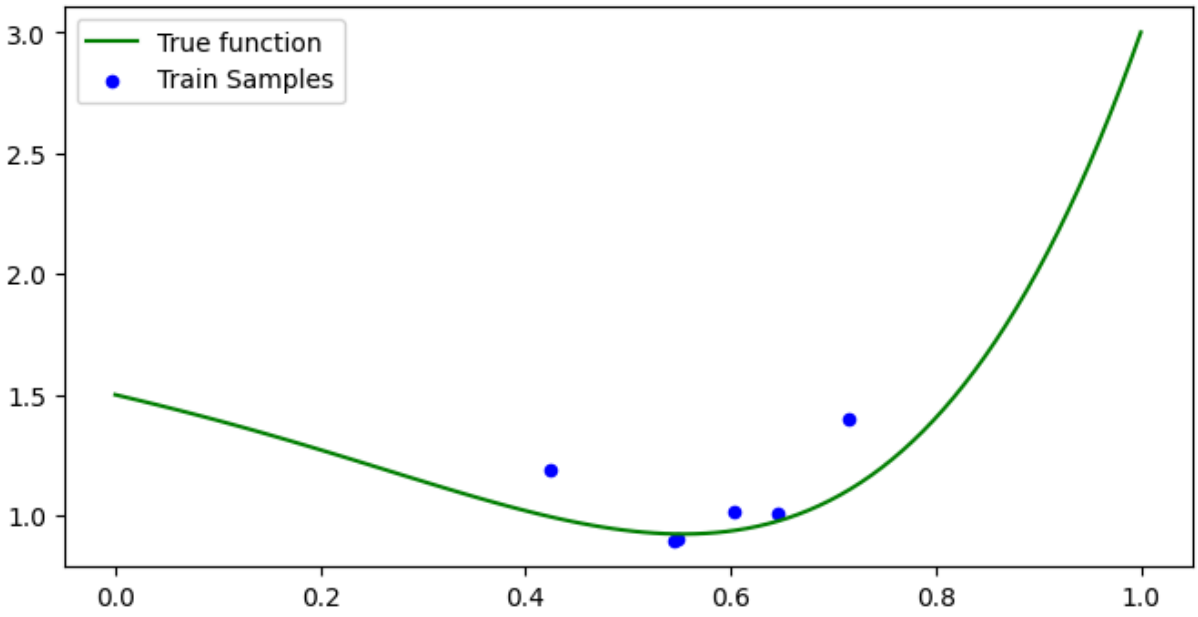
Law of Total Variance formula found/used from Wikipedia:
https://en.wikipedia.org/wiki/Law_of_total_variance

1.2 - plotting uncertainity band

In [3]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings('ignore')
```

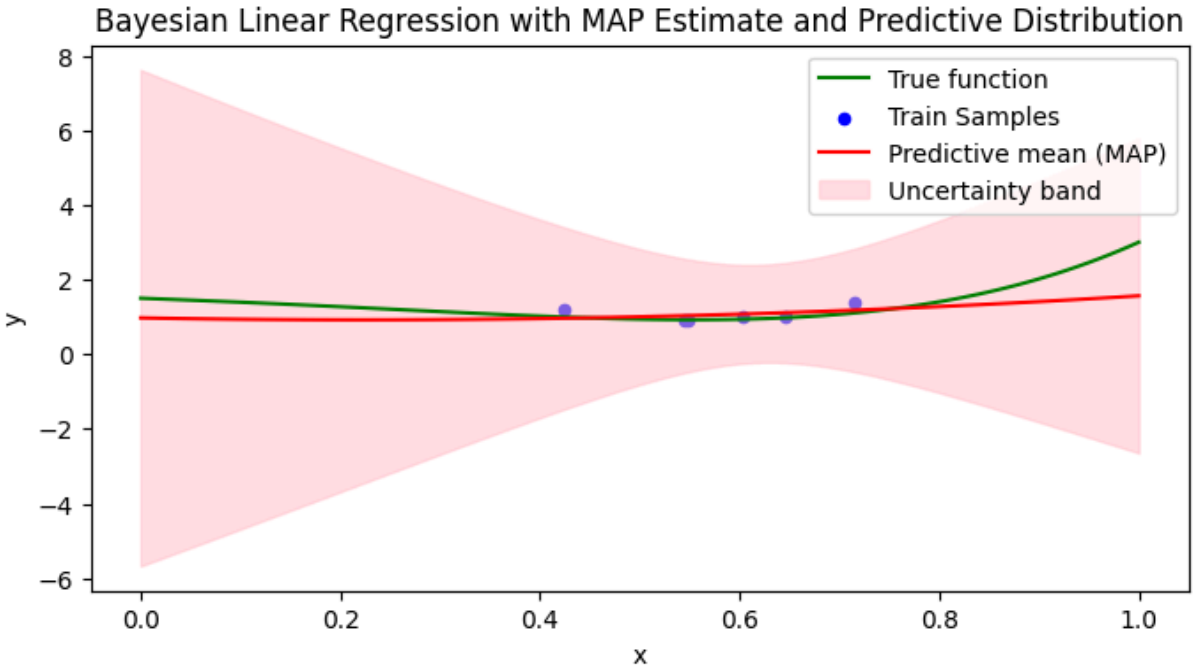
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [8, 4]
np.random.seed(0)

## please keep the below codes for generating x, DO NOT CHANGE!
# ================================================================================
n_train_samples = 6
X_train = np.sort(np.random.rand(n_train_samples))

order_true = 5
true_coefficient = np.array([.5, -1, -.5, -2, 5])

def true_fn(X):
    f = np.ones((X.shape))
    for i in range(order_true):
        f += true_coefficient[i] * X**i
    return f

X_test = np.linspace(0., 1., 100)
## adding y_train
y_train = true_fn(X_train) + (np.random.randn(n_train_samples)) * 0.2

plt.plot(X_test,true_fn(X_test),'g',label = 'True function')
plt.scatter(X_train, y_train, edgecolor='b', s=20, facecolor = 'b', label="T
plt.legend()

# [CONTINUE YOUR WORK FROM HERE!]
```

Out[3]: <matplotlib.legend.Legend at 0x7f126c761300>



In [7]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from numpy.linalg import inv
```

```
# Add Gaussian noise with delta = 0.2
delta = 0.2
# y_train = true_fn(X_train) + np.random.normal(0, delta, size=X_train.shape

# Plot the true function and noisy data samples
X_test = np.linspace(0, 1, 100).reshape(-1, 1)
plt.plot(X_test, true_fn(X_test), 'g', label='True function')
plt.scatter(X_train, y_train, edgecolor='b', s=20, facecolor='b', label='Tra
plt.legend()

# Bayesian Linear Regression with MAP estimate
def design_matrix(X, degree):
    poly = PolynomialFeatures(degree=degree)
    return poly.fit_transform(X.reshape(-1, 1))

degree = 2  # Assuming 2nd order model for PolynomialFeatures
Phi_train = design_matrix(X_train, degree)

# Prior settings (regularization parameter lambda)
lamda = 3 # Prior variance

# Compute theta_MAP (MAP estimate)
I = np.eye(Phi_train.shape[1])  # Identity matrix
theta_map = inv(Phi_train.T @ Phi_train + (delta**2 / lamda**2) * I) @ Phi_t

# Compute posterior covariance (sigma_0)
sigma_0 = inv( (Phi_train.T @ Phi_train)/delta**2 + I/lamda**2)

# Predictive distribution
def predictive_distribution(X_test, theta_map, sigma_0, degree):
    Phi_test = design_matrix(X_test, degree)

    mean_pred = Phi_test@ theta_map
    var_pred = np.sum(Phi_test @ sigma_0 @ Phi_test.T, axis=1) + delta**2
    return mean_pred, np.sqrt(var_pred)

mean_pred, std_pred = predictive_distribution(X_test, theta_map, sigma_0, de
# Plot predictive mean and uncertainty bands
plt.plot(X_test, mean_pred, 'r', label='Predictive mean (MAP)')
plt.fill_between(X_test.flatten(), mean_pred.flatten()-2*std_pred.flatten(),
                 mean_pred.flatten()+2*std_pred.flatten(), color='pink', alp

plt.legend()
plt.title('Bayesian Linear Regression with MAP Estimate and Predictive Distr
plt.xlabel('x')
plt.ylabel('y')
plt.show()

# Discussion:
# - theta_map is the result of optimizing the regularized loss function.
# - The prior variance (lambda^2) acts as regularization to avoid overfittin
# - The predictive distribution takes into account the posterior uncertainty
```



1.2 - Notes

The prior variance of 2 ($\lambda^2 = 3$) versus the data's variance of 0.04 ($\delta^2 = 0.04$) is chosen to have a reasonable regularization value. Note that the optimal theta (via the MAP approach) is a least squares solution with regularization = $\delta^2/\lambda^2 = 0.0133$. Also, the uncertainity band is shown with 2 standard deviations.

1.3 - benefits of Bayesian vs Maximum Likelihood

By incorporating prioir information of the parameters with the use of prioir distributions, the Bayesian methods can take into consideration previous data or domain knowledge to inform current analysis. The maximum likelihood approach does not allow incorporation of prior information or results, and relies solely on the observed data at the current time.

Bayesian methods provide a full posterior distribution (posterioir predictive distribution) which provides the complete picture of both the appropriate value and uncertainity for each estimate; meanwhile, maximum likelihood can only provide an estimate of the parameter without quantifying the uncertainity of the estimate.

With the use of the prior, Bayesian methods can be more stable for small sample sizes; with just the small sample size (and no prior), maximum likelihood estimates may produce widely unreliable results.

Question 2 (20 Points) Mean and Covariance

In lecture 7, we maximize the likelihood:

$$\max_{\boldsymbol{\mu}_k, \Sigma_k} \sum_{i:y^{(i)}=k} \log P(\boldsymbol{x}^{(i)}|y^{(i)}; \boldsymbol{\mu}_k, \Sigma_k) = \max_{\boldsymbol{\mu}_k, \Sigma_k} \sum_{i:y^{(i)}=k} \log \mathcal{N}(\boldsymbol{x}^{(i)}|\boldsymbol{\mu}_k, \Sigma_k).$$

Prove that the empirical means and covariances of each class k are

$$\begin{aligned}\mu_k &= \frac{\sum_{i:y^{(i)}=k} \boldsymbol{x}^{(i)}}{n_k} \\ \Sigma_k &= \frac{\sum_{i:y^{(i)}=k} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k)(\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k)^\top}{n_k}.\end{aligned}$$

where $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$, $\boldsymbol{\mu} \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$, n is the number of features of training sample $\boldsymbol{x}^{(i)}$.

2.1 - proof for μ_k

2. Prove: $\mu_K = \frac{\sum_{i:y^i=K} x^i}{n_K}$, $\Sigma_K = \frac{\sum_{i:y^i=K} (x^i-\mu_K)(x^i-\mu_K)^T}{n_K}$

for $x^i \in \mathbb{R}^n$, $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$, n is number of features in training sample x^i. (dimension)

Let there be K classes. we maximize:

$$\max_{\mu_K, \Sigma_K} \sum_{i, y^i=K} \log P(x^i | y^i, \mu_K, \Sigma_K)$$

$$= \max_{\mu_K, \Sigma_K} \sum_{i:y^i=K} \log (N(x^i | \mu_K, \Sigma_K)) = \max_{\mu_K, \Sigma_K} L(\mu_K, \Sigma_K)$$

let n_K be the number of data samples (x^i, y^i) such that $y^i = K$ class k, $K \in [1, K]$

$$N(x^i | \mu_K, \Sigma_K) = \frac{1}{(2\pi)^{n/2} |\Sigma_K|^{1/2}} \exp\left(\frac{-1}{2}(x^i - \mu_K)^T \Sigma_K^{-1} (x^i - \mu_K)\right)$$

$$\log N(x^i | \mu_K, \Sigma_K) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma_K| - \frac{1}{2}(x^i - \mu_K)^T \Sigma_K^{-1} (x^i - \mu_K)$$

Thus, our problem likelihood $L(\mu_K, \Sigma_K)$ is

$$L(\mu_K, \Sigma_K) = -\frac{n_K n}{2}\log(2\pi) - \frac{n_K}{2}\log|\Sigma_K| - \sum_{i:y^i=K} \frac{1}{2}(x^i - \mu_K)^T \Sigma_K^{-1} (x^i - \mu_K)$$

To maximize $L(\mu_K, \Sigma_K)$, we set $\frac{\partial L}{\partial \mu_K}, \frac{\partial L}{\partial \Sigma_K} = 0$.

1) $\frac{\partial L}{\partial \mu_K} = \frac{-1}{2} \sum_{i:y^i=K} \frac{\partial}{\partial \mu_K} (x^i - \mu_K)^T \Sigma_K^{-1} (x^i - \mu_K)$

we know $\frac{d}{dx} x^T A x = 2Ax$.

(shown in HW1).

So, with the chain rule,

$$\frac{\partial L}{\partial \mu_k} = \frac{1}{2} \sum_{i:y^i=k} \Sigma_k^{-1} (x^i - \mu_k)(-2)$$

$$= \sum_{i:y^i=k} \Sigma_k^{-1} (x^i - \mu_k)$$

$$= \Sigma_k^{-1} \cdot \sum_{i:y^i=k} (x^i - \mu_k)$$

$$= \Sigma_k^{-1} \left(-n_k \mu_k + \sum_{i:y^i=k} x^i \right)$$

Setting $\frac{\partial L}{\partial \mu_k} = 0$, we have:

$$\Sigma_k^{-1} \left(\sum_{i:y^i=k} x^i - n_k \mu_k \right) = \vec{0}$$

Since Σ_k^{-1} is invertible $(\Sigma_k^{-1})^{-1} = \Sigma_k$ exists & Σ_k^{-1} is not a null matrix ($\Sigma_k \Sigma_k^{-1} = I$, not null),

we have $\underbrace{\Sigma_k \Sigma_k^{-1}}_{I} \left(\sum_{i:y^i=k} x^i - n_k \mu_k \right) = \Sigma_k \cdot \vec{0}$

Thus, $\sum_{i:y^i=k} x^i - n_k \mu_k = \vec{0}$.

So, $$\boxed{\mu_k = \frac{\sum_{i:y^i=k} x^i}{n_k}}$$

2.2 - proof for Σ_k

Now, setting $\frac{\partial L}{\partial \Sigma_k} = 0$:

$\partial L / \partial \Sigma_k =$

$$\frac{\partial}{\partial \Sigma_k} \left(\frac{n_k}{2} \log |\Sigma_k| - \frac{1}{2} \sum_{i : y^i = k} (x^i - \mu_k)^T \Sigma_k^{-1} (x^i - \mu_k) \right)$$

$$\frac{\partial}{\partial \Sigma_k} \frac{n_k}{2} \log |\Sigma_k| = \frac{n_k}{2} \frac{\partial}{\partial \Sigma_k} \log |\Sigma_k|$$

$$\frac{\partial}{\partial \Sigma_k} \log |\Sigma_k| = \frac{1}{|\Sigma_k|} \cdot \frac{\partial}{\partial \Sigma_k} |\Sigma_k|$$

Math Derivations:

we know Σ_K is a square, invertible matrix.

we know $C_{ij} = (-1)^{i+j} M_{ij}$; $M_{ij} = \det(\Sigma_{i,j})$

where C is the cofactor at row i, column j & M is the minor.

Then, $|\Sigma_K| = \sum_{j=1}^{n} \Sigma_{1j} C_{1j}$

$= \sum_{j=1}^{n} \Sigma_{lj} C_{lj}$ ↙ for any. $\forall l \in [1, d]$

let $l = a$

$$\frac{\partial |\Sigma|}{\partial \Sigma_{ab}} = \frac{\partial}{\partial \Sigma_{ab}} \left(\sum_{j=1}^{n} \Sigma_{aj} C_{aj} \right)$$

$$= \sum_{j=1}^{n} \frac{\partial}{\partial \Sigma_{ab}} \Sigma_{aj} C_{aj}$$

$= C_{ab} = C_{ba}$ → $i+j = a+b$ for C_{ab} & C_{ba}.
$M_{ij} = M_{ji}$ since Σ is symmetric.

$|\Sigma| = \Sigma \cdot C$ $\quad \Sigma \cdot \Sigma^{-1} = I_d$

$\Sigma = \frac{|\Sigma|}{C}$ $\quad \Sigma^{-1} = \frac{Id}{\Sigma} = \frac{C}{|\Sigma|}$; $C = |\Sigma| \Sigma^{-1}$

So, $\frac{\partial}{\partial \Sigma} |\Sigma| = C = |\Sigma| \Sigma^{-1}$.

Or, $\frac{\partial}{\partial \Sigma_K} |\Sigma_K| = |\Sigma_K| \Sigma_K^{-1}$

so, $\frac{\partial}{\partial \Sigma_k} \log|\Sigma_k| = \frac{1}{|\Sigma_k|} |\Sigma_k| \Sigma_k^{-1} = \Sigma_k^{-1}$

F Aside: $\frac{\partial}{\partial A} A^{-1} = ?$

$A^{-1}A = I \Rightarrow \frac{\partial}{\partial A}(A^{-1}A) = 0.$

w/ product rule:

$\frac{\partial A^{-1}}{\partial A} \cdot A + \frac{\partial A}{\partial A} \cdot A^{-1} = 0$

$\frac{\partial A^{-1}}{\partial A} \cdot A + A^{-1} = 0 \Rightarrow \frac{\partial A^{-1}}{\partial A} A = -A^{-1}$

$\frac{\partial A^{-1}}{\partial A} A A^{-1} = -A^{-1}A^{-1}$; $\frac{\partial A^{-1}}{\partial A} = -A^{-1}A^{-1}$

$\frac{\partial}{\partial A} v^T A^{-1} v = ?$ let $f(A) = v^T A^{-1} v$

$\frac{\partial}{\partial A}$ ↑ vector

$\frac{\partial}{\partial A} f(A) = v^T \frac{\partial}{\partial A} A^{-1} v = v^T(-A^{-1} \cdot A^{-1}) v$

$\frac{\partial}{\partial A}(v^T A^{-1} v) = -v^T A^{-1} A^{-1} v$ ∎

$\frac{\partial L}{\partial \Sigma_k} = -\frac{n_k}{2} \frac{\partial}{\partial \Sigma_k}(\log|\Sigma_k|) - \frac{1}{2} \sum_{i: y^i = k} \frac{\partial}{\partial \Sigma_k} (x^i - \mu_k)^T \Sigma_k^{-1} (x^i - \mu_k)$

$\frac{\partial L}{\partial \Sigma_k} = -\frac{n_k}{2} \Sigma_k^{-1} + \frac{1}{2} \sum_{i: y^i = k} (x^i - \mu_k)^T \Sigma_k^{-1} \Sigma_k^{-1} (x^i - \mu_k)$

† Aside: $v^T A^{-1} A^{-1} v = A^{-1} v v^T A^{-1}$ when A is symmetric?
↳ $v \in \mathbb{R}^d$, $A \in \mathbb{R}^{d \times d}$

let $p = A^{-1}v$. $p^T = v^T (A^{-1})^T$

for $A = A^T$:

$A^{-1}A = I$;

$I = (A^T)^{-1}(A^T) = (A^T)^{-1} A = I$; $(A^T)^{-1} = A^{-1}$

So, $p^T = v^T A^{-1}$

Thus, $v^T A^{-1} A^{-1} v = p^T p = \sum_{i=1}^{d} (p_i)^2 = \|A^{-1}v\|^2$

Now, we use proof by contradiction:

we evaluate the RHS of the hypothesis:

$$A^{-1}v \cdot v^T A^{-1} = p \cdot p^T = \sum_{i=1}^{d} (p_i)^2 = \|A^{-1}v\|^2$$

Thus for the hypothesis to be false we must have $\|A^{-1}v\|^2 \neq \|A^{-1}v\|^2$ which is obviously false.

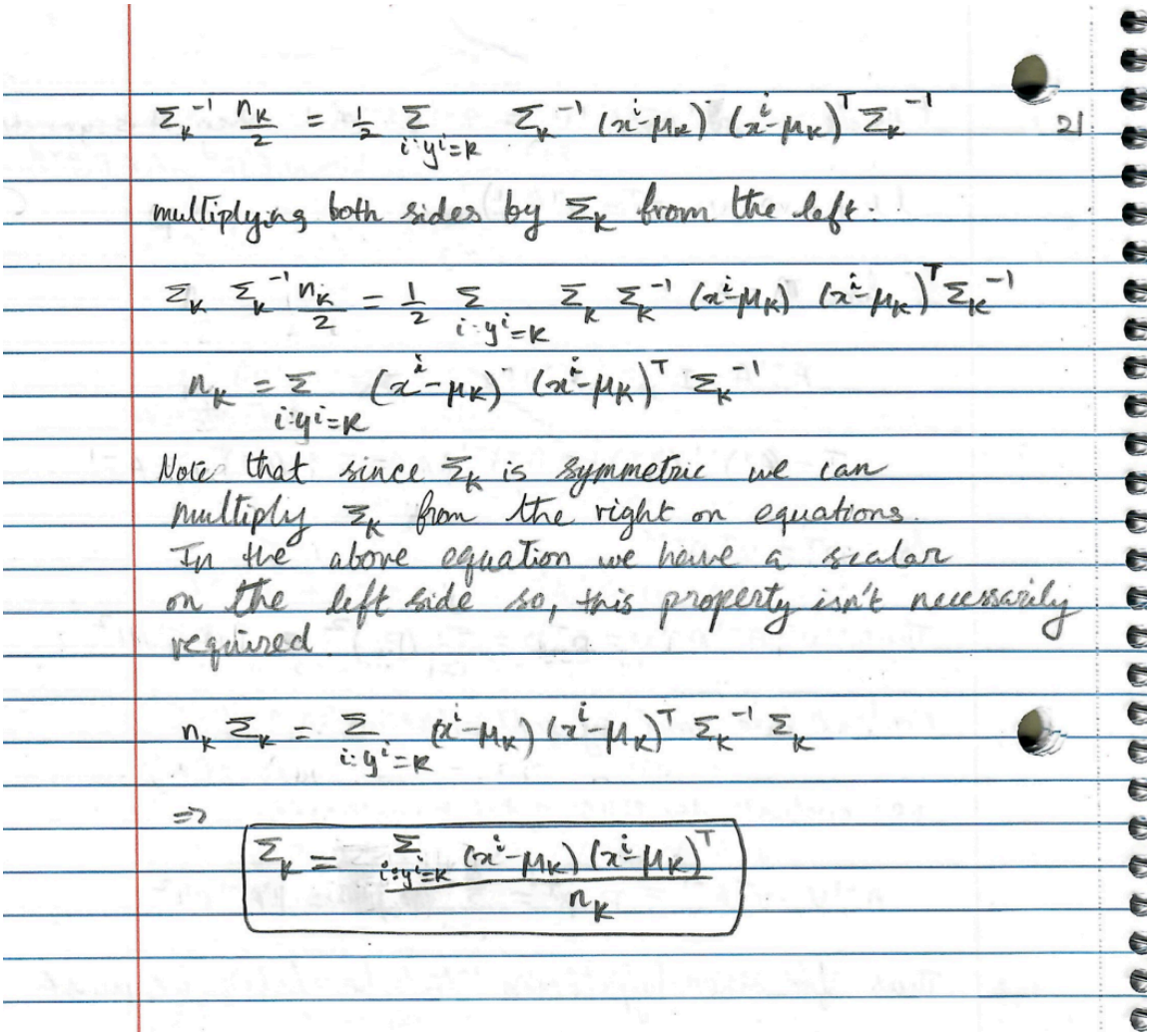
Both sides can be simplified to the same value so

$$v^T A^{-1} A^{-1} v = A^{-1} v v^T A^{-1} \quad \square$$

Thus, we have, setting $\frac{\partial L}{\partial \Sigma_k} = 0$,:

$$\frac{n_k}{2} \Sigma_k^{-1} = \frac{1}{2} \sum_{i : y^i = k} (x^i - \mu_k)^T \Sigma_k^{-1} \Sigma_k (x^i - \mu_k)$$

scalar → $\frac{n_k}{2} \Sigma_k^{-1} = \frac{1}{2} \sum_{i : y^i = k} \Sigma_k^{-1} (x^i - \mu_k)(x^i - \mu_k)^T \Sigma_k$



Question 3: (25 Points) Generative VS Discriminative Models

We are going to use the Iris flower dataset. We will use three features sepal length (cm), sepal width (cm), petal length (cm) to classify types of flowers in generative model.

1. Compute the empirical means and covariance matrices for each type of flowers.
2. Generating new flowers, denoted as set S, from the learned propability and plotting these generated flowers in 3D. Observe and discuss your results.
3. We assume that set S is test data. Classify the test data S using the generative model that you have found in question 4.1. Discuss you results.
4. We assume that set S is test data. Using logistic regression to training the model, then verify the learn model with test data S. Compare and discuss your results with question 4.3

In [141…

```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn import datasets

# Load the Iris dataset
iris = datasets.load_iris(as_frame=True)

# print part of the dataset
iris_X, iris_y = iris.data, iris.target
pd.concat([iris_X, iris_y], axis=1).head()

#print(iris_X.shape, iris_y.shape)
```

Out[141…

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

3.0 - data

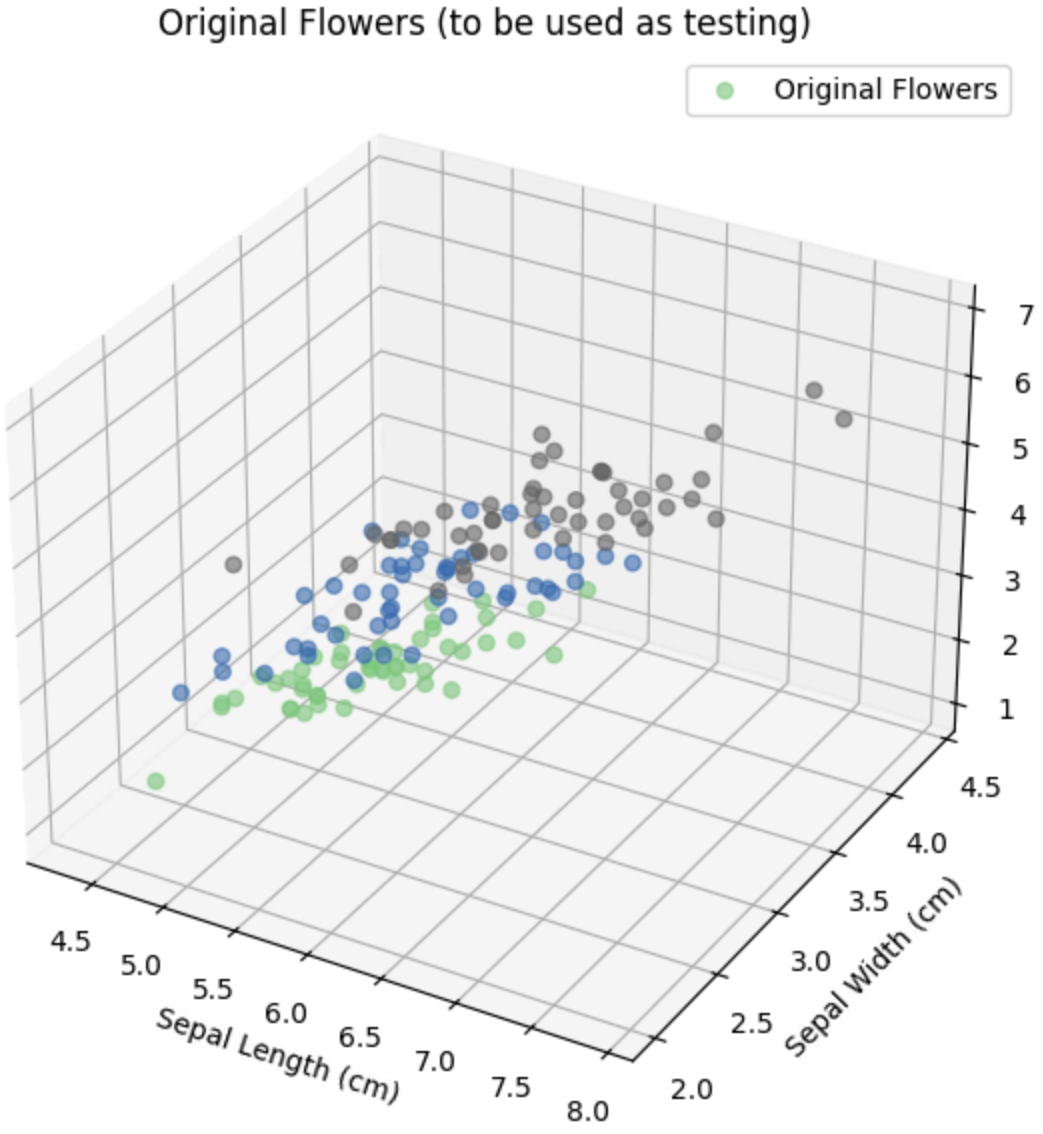
Note that the different colors for each cluster in the plots is representative of a different iris flower type (i.e., setosa, versicolor, etc.).

In [142…

```
# Create a DataFrame for easier handling
# Select only the desired columns: sepal length, sepal width, and petal leng
iris_X = iris_X[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)

# Rename the columns
iris_X.columns = ['sepal_length', 'sepal_width', 'petal_length']
iris_y.name = 'target'

df = pd.concat([iris_X, iris_y], axis=1)
df.columns = ['sepal_length', 'sepal_width', 'petal_length', 'target']


# Display the first few rows of the dataset
print(df.head())
# compare generate vs train
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Sepal Width (cm)')
ax.set_zlabel('Petal Length (cm)')
plt.title('Original Flowers (to be used as testing)')
scatter1 = ax.scatter(iris_X['sepal_length'], iris_X['sepal_width'], iris_X[
```

```
ax.legend()
plt.show()
```

	sepal_length	sepal_width	petal_length	target
0	5.1	3.5	1.4	0
1	4.9	3.0	1.4	0
2	4.7	3.2	1.3	0
3	4.6	3.1	1.5	0
4	5.0	3.6	1.4	0



3.1 - mean/covariance for each flower

In [147…

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


# 1. Compute the empirical means and covariance matrices for each type of fl
means = {}
```

```
covariances = {}

for target in np.unique(iris_y):
    flower_data = df[df['target'] == target][['sepal_length', 'sepal_width',
    means[target] = flower_data.mean().values
    covariances[target] = flower_data.cov().values

print("Empirical Means:")
for target, mean in means.items():
    print(f"Class {target}: {mean}")

print("\nCovariance Matrices:")
for target, cov in covariances.items():
    print(f"Class {target}:\n{cov}\n")
```

```
Empirical Means:
Class 0: [5.006 3.428 1.462]
Class 1: [5.936 2.77  4.26 ]
Class 2: [6.588 2.974 5.552]

Covariance Matrices:
Class 0:
[[0.12424898 0.09921633 0.0163551 ]
 [0.09921633 0.1436898  0.01169796]
 [0.0163551  0.01169796 0.03015918]]

Class 1:
[[0.26643265 0.08518367 0.18289796]
 [0.08518367 0.09846939 0.08265306]
 [0.18289796 0.08265306 0.22081633]]

Class 2:
[[0.40434286 0.09376327 0.3032898 ]
 [0.09376327 0.10400408 0.07137959]
 [0.3032898  0.07137959 0.30458776]]
```

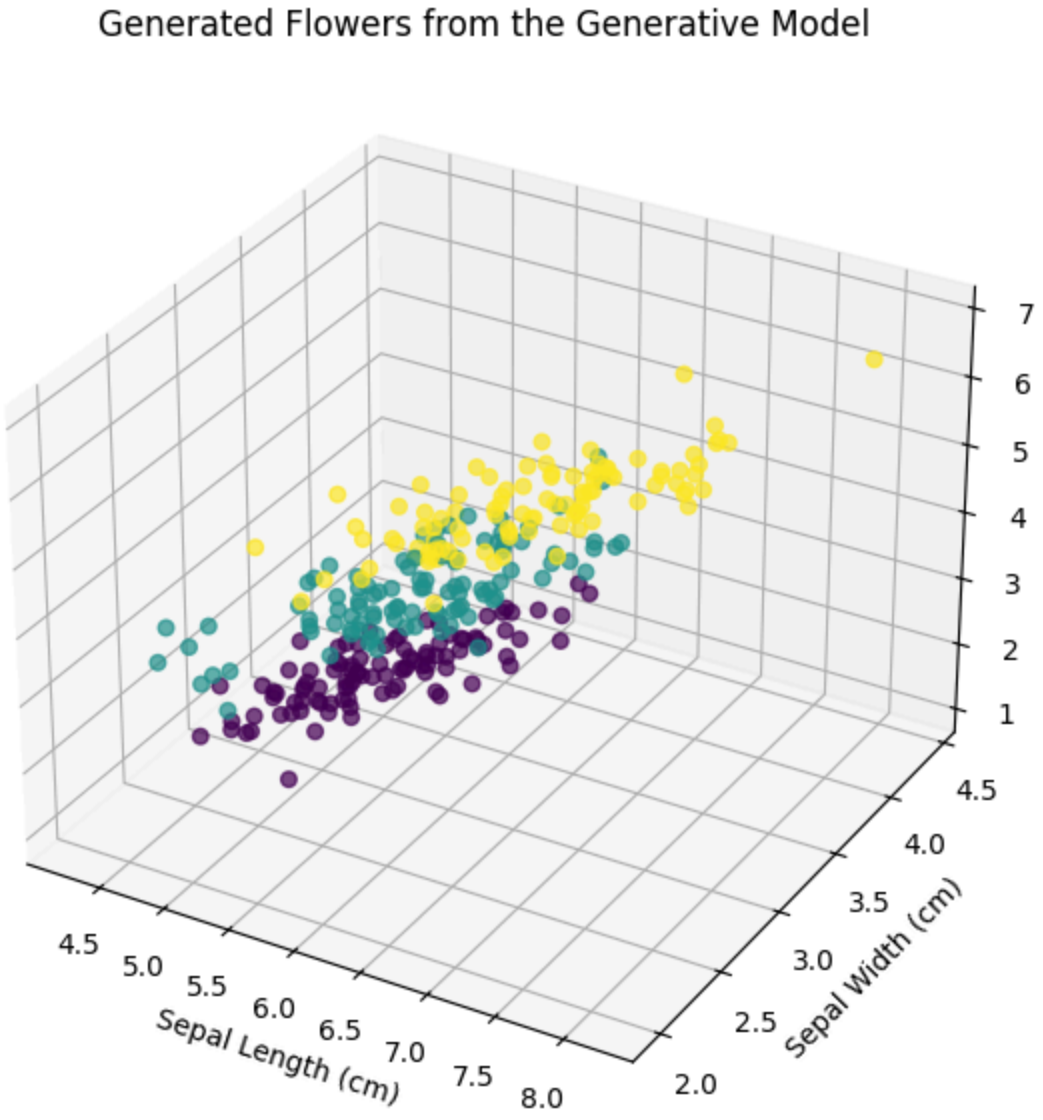
3.2 - generate new flowers

In [163…

```
# 2. Generate new flowers from the learned probability and plot them in 3D.
num_samples = 100
S = []

for target in np.unique(iris_y):
    mean = means[target]
    cov = covariances[target]
    # Generate samples from a multivariate normal distribution
    new_samples = np.random.multivariate_normal(mean, cov, num_samples)
    S.append(new_samples)

# Convert the list of arrays to a single array
S = np.vstack(S)
S_labels = np.concatenate([[target]*num_samples for target in np.unique(iris

# 3D Plotting of the generated flowers
```

```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(S[:, 0], S[:, 1], S[:, 2], c=S_labels, cmap='viridis', s=30, alph
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Sepal Width (cm)')
ax.set_zlabel('Petal Length (cm)')
plt.title('Generated Flowers from the Generative Model')
```

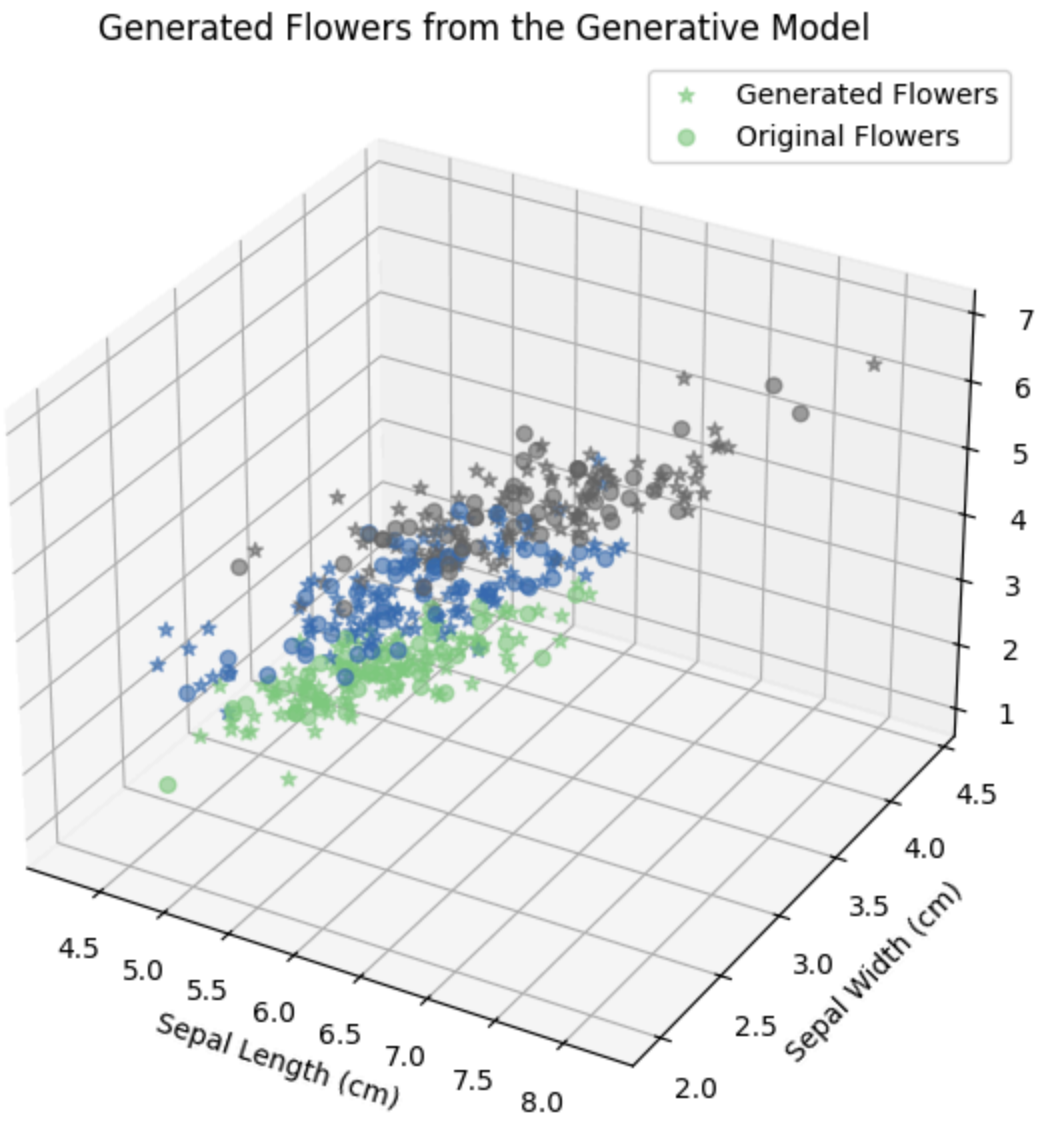
Out[163… Text(0.5, 0.92, 'Generated Flowers from the Generative Model')



In [167…

```
# compare generate vs train
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(S[:, 0], S[:, 1], S[:, 2], c=S_labels, cmap='Accent', s=30, alpha
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Sepal Width (cm)')
ax.set_zlabel('Petal Length (cm)')
plt.title('Generated Flowers from the Generative Model')
scatter1 = ax.scatter(iris_X['sepal_length'], iris_X['sepal_width'], iris_X[
```

```
ax.legend()
plt.show()
```



Discussion

Generated Flowers: The new flowers were generated using a multivariate normal distribution based on the empirical means and covariance matrices of the original Iris dataset. This process simulates how new data points might look if they follow the same statistical properties as the original data.

3D Plot: The plot shows the generated flowers in a 3D space defined by sepal length, sepal width, and petal length. Each point is colored according to its class label, providing a visual representation of the distribution of the generated flowers.

Distribution: The generated flowers should ideally follow the same distribution as the original flowers. If the generative model has captured the underlying data distribution well, the generated points will be spread out in a similar manner to

the original data points. This is especially obvious in the second plot, where the true data and the generated data are overlayed/plotted on the same plot. Note that the different colors are representative of different iris groups; the similar groupings of the generated and true data are obvious, indicating a well-built generative model.

Model Validation: This visualization helps validate the generative model by comparing the spread and clustering of the generated flowers to the original dataset. If the generated flowers appear to be well-distributed and form similar clusters, it indicates that the model has effectively learned the underlying distribution of the data (this is true, per our visual observation).

3.3 - classify test data S

In [168…

```
# 3. Classify the test data using the generative model.
# We'll use the multivariate Gaussian likelihood for classification
def classify_generative(samples, means, covariances):
    predictions = []
    for sample in samples:
        probs = []
        for target in np.unique(iris_y):
            mean = means[target]
            cov = covariances[target]
            # Calculate the multivariate Gaussian probability density
            diff = sample - mean
            prob = np.exp(-0.5 * diff.T @ np.linalg.inv(cov) @ diff) / np.sc
            probs.append(prob)
        predictions.append(np.argmax(probs))
    return np.array(predictions)

predictions_generative = classify_generative(S, means, covariances)
accuracy_generative = accuracy_score(S_labels, predictions_generative)
print(f"Generative Model Classification Accuracy: {accuracy_generative:.4f}"
```

```
Generative Model Classification Accuracy: 0.9467
```

Discussion

The generative model accuracy is $\approx 94.67\%$. The high accuracy is indicative of an effective generative model (effective at classifying test data). This suggests that the model has successfully captured the underlying data distribution and can distinguish between different classes.

Note that the generative model assumes the features follow a gaussian distribution; the high accuracy implies that this assumption is reasonably valid for th Iris dataset, which led the model to perform well.

3.4 - logistic regression

In [169…

```python
# Fit logistic regression
logistic_model = LogisticRegression(max_iter=500)
logistic_model.fit(iris_X, iris_y)

# Verify the learned model with test data S
predictions_logistic = logistic_model.predict(S)
accuracy_logistic = accuracy_score(S_labels, predictions_logistic)
print(f"Logistic Regression Classification Accuracy: {accuracy_logistic:.4f}")
```

```
Logistic Regression Classification Accuracy: 0.9400
```

Discussion

The discriminative logistic regression model accuracy is $\approx 94.00\%$. The high accuracy is indicative of an effective discriminative model (effective at classifying test data). This suggests that the model has successfully captured the underlying data distribution and can distinguish between different classes. Note that discriminative models directly model $P(y|x)$ and do not need assumptions about the gaussian features. So, we would expect the logistic regression to be at a higher accuracy than the generative model. Generally, since discriminative models don't model (or assume anything about) the joint distribution $P(y, x|\theta)$ or $P(x|\theta)$, discriminative models are more accurate for classification than compared with generative models. In this case, we can visually see that the each feature follows an approximate gaussian distribution. Thus, this is a type of prior information that allows GDA to be a better model than the discriminative logistic regression model. Also, since logistic regression only allows linear boundaries, and since the iris dataset is obviously not linearly separable, GDA, which allows for quadratic boundaries, can be a more appropriate classification model.

Notes about different probability distributions:

$P(x|y)$: This is the probability of the features (x) given the class label (y). Generative models model this conditional probability. They assume a specific distribution for the features given the class (e.g., Gaussian distribution).

$P(y)$: This is the prior model probability of the class label (y). Generative models also model/postulate this probability.

$P(x)$: This is the marginal probability of the features (x). It can be computed using the law of total probability: ($P(x) = \sum_y P(x|y)P(y)$), but is generally not needed for analysis (acts as a constant).

$P(y|x)$: This is the posterior probability of the class label (y) given the features (x). Generative models use Bayes' theorem to compute this. This is the main purpose of a generative model.

Question 4: (25 Points) Bernoulli Naive Bayes Model

1. Derive the formula, from lecture 8, for the optimal value of ψ_{jk}

$$\psi_{jk} = \frac{n_{jk}}{n_k}.$$

2. In lecture 8, we classified a document by the Naive Bayes (bag of words) method. You will now make a simple comparison between Naive Bayes and Logistic Regression. Remove class 2 and 3 from the training data. By doing so, our data becomes a binary classification problem. Perform the Naive Bayes (bag of words) method and logistic regression method for the obtained 2-class data. Discuss the results.

4.1 - optimal ψ_{jk}

4

1) Derive $\psi_{jK} = \frac{n_{jK}}{n_K}$

ψ_{jK}: $P(x=1 \mid y=K)$, probability of seeing word that belongs to vocabulary set $\phi(x)$, given restriction to category K (i.e spam email category).

n_{jK}: frequency of word j in class K

n_K: number of documents in class k.

for dataset $\mathcal{D} = \{(x^i, y^i) \mid i = 1, 2 \cdots n\}$ we optimize/maximize the log-likelihood $\log(P_\theta(x,y))$ w/ maximum likelihood:

$$\theta = \{\phi_1, \phi_2 \cdots \phi_K, \psi_{11}, \psi_{21} \cdots \psi_{dK}\}$$

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log P_\theta(x^{(i)}, y^{(i)}) = \arg\max_\theta \ell(\theta)$$

$$P_\theta(x,y) = P_\theta(x \mid y) \cdot P_\theta(y).$$

Note:
$P_\theta(x_j = 1 \mid y = K) = \text{Bernoulli}(\psi_{jK})$

$P_\theta(y) = \text{Categorical}(\phi_1 \cdots \phi_K)$

so,

$$\ell(\theta) = \sum_{i=1}^{n} \log \underbrace{P_\theta(x^{(i)} \mid y^{(i)})}_{\text{only dependent on } \vec{\psi}} + \sum_{i=1}^{n} \log \underbrace{P_\theta(y^{(i)})}_{\text{only dependent on } \vec{\phi}}$$

$$= \sum_{k=1}^{K} \sum_{j=1}^{d} \sum_{i: y^i = k} \log P(x^{(i)} \mid \psi_{jK}) + \sum_{i=1}^{n} \log P(y^{(i)} \mid \vec{\phi})$$

we know:

$$P(x_j^{(i)} | \psi_{jk}) = \psi_{jk}^{x_j^{(i)}} (1-\psi_{jk})^{1-x_j^{(i)}}$$

so,

$$l(\theta) = \sum_{k=1}^{K} \sum_{j=1}^{d} \sum_{i:y^i=k} \left(x_j^{(i)} \log \psi_{jk} + (1-x_j^{(i)})(1-\psi_{jk}) \right) + \sum_{i=1}^{n} \log P(y^i | \vec{\phi})$$

to find the (local) maxima of $l(\theta)$, we set $\frac{\partial l}{\partial \psi_{jk}} = 0$ & to get

$$\frac{\partial l}{\partial \phi} = 0.$$

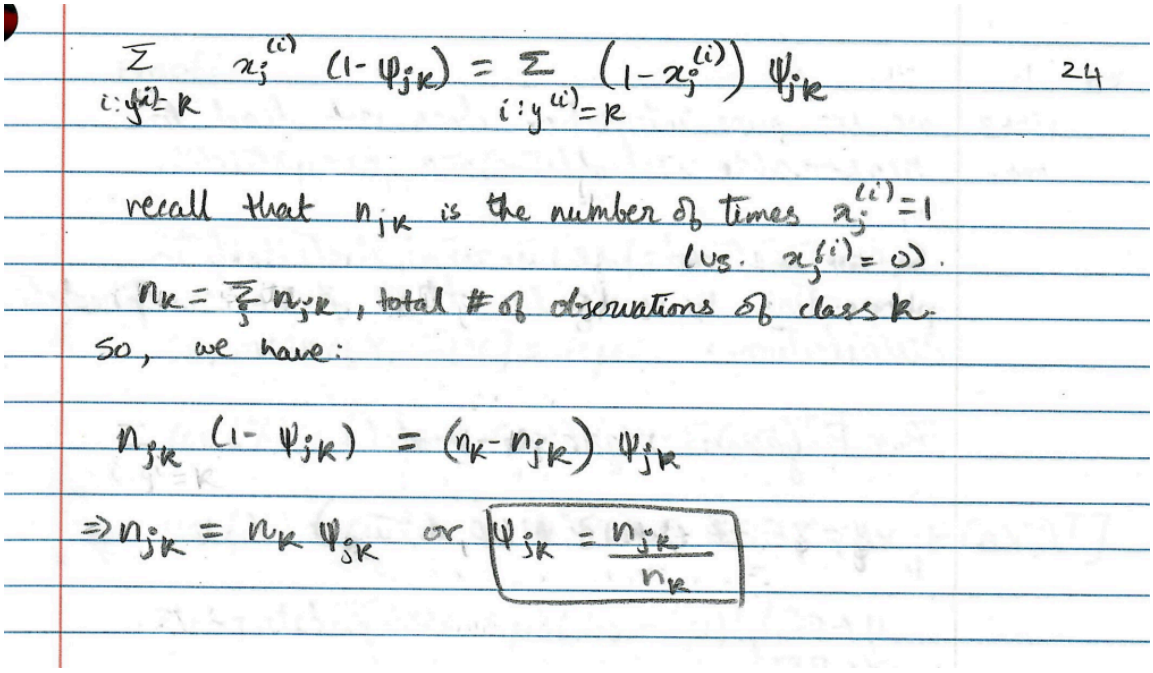
$$1) \quad \frac{\partial l}{\partial \psi_{jk}} = \sum_{i:y^{(i)}=k} \frac{\partial}{\partial \psi_{jk}} \left[x_j^{(i)} \log \psi_{jk} + (1-x_j^{(i)})(1-\psi_{jk}) \right]$$

$$\frac{\partial l}{\partial \psi_{jk}} = \sum_{i:y^i=k} \frac{1}{\psi_{jk}} x_j^{(i)} - \frac{1}{1-\psi_{jk}} (1-x_j^{(i)})$$

$$\Rightarrow \sum_{i:y^i=k} \frac{x_j^{(i)}}{\psi_{jk}} - \frac{1-x_j^{(i)}}{1-\psi_{jk}} = 0.$$

$$\sum_{i:y^i=k} \frac{x_j^{(i)}}{\psi_{jk}} = \sum_{i:y^i=k} \frac{1-x_j^{(i)}}{1-\psi_{jk}}$$

using our definition for n_{jk} and rearranging:



4.2 - Naive Bayes

In [89]:
```
import numpy as np
# import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer

# for this lecture, we will restrict our attention to just 4 different newsg
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci

# load the dataset
twenty_train = fetch_20newsgroups(subset='train', categories=categories, shu

# vectorize the training set
count_vect = CountVectorizer(binary=True, max_features=1000)
y_train = twenty_train.target
X_train = count_vect.fit_transform(twenty_train.data).toarray()
X_train.shape

print('NOTE: our data has 4 class group')
print(y_train)
print('You NEED to remove class 2 and 3 to create the 2-class data')
y_train
```

```
NOTE: our data has 4 class group
[1 1 3 ... 2 2 2]
You NEED to remove class 2 and 3 to create the 2-class data
```

Out[89]:
```
array([1, 1, 3, ..., 2, 2, 2])
```

In [92]:
```
import numpy as np
# import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report


twenty_test = fetch_20newsgroups(subset='test', categories=categories, shuff

# vectorize the training set
count_vect_test = CountVectorizer(binary=True, max_features=1000)
y_test = twenty_test.target
X_test = count_vect_test.fit_transform(twenty_test.data).toarray()



# Filter out classes 2 and 3
mask_train = np.isin(y_train, [0, 1])  # Keep only classes 0 and 1
X_train_split = X_train[mask_train]
y_train_split = y_train[mask_train]
mask_test = np.isin(y_test, [0, 1])  # Keep only classes 0 and 1
X_test_split = X_test[mask_test]
y_test_split = y_test[mask_test]
# # Split the data into training and testing sets
# X_train_split, X_test_split, y_train_split, y_test_split = train_test_spli
#     X_train_binary, y_train_binary, test_size=0.2, random_state=42
# )

# Train Naive Bayes
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_split, y_train_split)
y_pred_nb = nb_classifier.predict(X_test_split)
y_pred_train_nb = nb_classifier.predict(X_train_split)
# Train Logistic Regression
logistic_classifier = LogisticRegression()
logistic_classifier.fit(X_train_split, y_train_split)
y_pred_logistic = logistic_classifier.predict(X_test_split)
y_pred_train_logistic = logistic_classifier.predict(X_train_split)

print("TRAINING RESULTS")

print("Naive Bayes Results:")
print("Accuracy:", accuracy_score(y_train_split, y_pred_train_nb))
print(classification_report(y_test_split, y_pred_nb, target_names=twenty_tra

print("Logistic Regression Results:")
print("Accuracy:", accuracy_score(y_train_split, y_pred_train_logistic))
print(classification_report(y_test_split, y_pred_logistic, target_names=twer

print("\nTESTING RESULTS")
# Evaluate and compare results
print("Naive Bayes Results:")
print("Accuracy:", accuracy_score(y_test_split, y_pred_nb))
print(classification_report(y_test_split, y_pred_nb, target_names=twenty_tra

print("Logistic Regression Results:")
```

```
print("Accuracy:", accuracy_score(y_test_split, y_pred_logistic))
print(classification_report(y_test_split, y_pred_logistic, target_names=twer
```

```
TRAINING RESULTS
Naive Bayes Results:
Accuracy: 0.9821428571428571
              precision    recall  f1-score   support

  alt.atheism       0.52      1.00      0.69       319
comp.graphics       1.00      0.26      0.41       389

     accuracy                           0.59       708
    macro avg       0.76      0.63      0.55       708
 weighted avg       0.79      0.59      0.53       708

Logistic Regression Results:
Accuracy: 1.0
              precision    recall  f1-score   support

  alt.atheism       0.81      0.77      0.79       319
comp.graphics       0.82      0.85      0.84       389

     accuracy                           0.82       708
    macro avg       0.82      0.81      0.82       708
 weighted avg       0.82      0.82      0.82       708


TESTING RESULTS
Naive Bayes Results:
Accuracy: 0.5918079096045198
              precision    recall  f1-score   support

  alt.atheism       0.52      1.00      0.69       319
comp.graphics       1.00      0.26      0.41       389

     accuracy                           0.59       708
    macro avg       0.76      0.63      0.55       708
 weighted avg       0.79      0.59      0.53       708

Logistic Regression Results:
Accuracy: 0.8177966101694916
              precision    recall  f1-score   support

  alt.atheism       0.81      0.77      0.79       319
comp.graphics       0.82      0.85      0.84       389

     accuracy                           0.82       708
    macro avg       0.82      0.81      0.82       708
 weighted avg       0.82      0.82      0.82       708
```

4.2 - Discussion

Generally, since discriminative models don't model (or assume anything about) the joint distribution P(y,x|theta) or P(x|theta), discriminative models are more

accurate for classification than compared with generative models. Since we have no information (prior) in regards to the newspaper/article classification, the discriminative model will obviously outperform the Naive Bayes model which must assume a gaussian distribution (P(x|y=Y) ~ Normal) and that each feature can independently predict the class.

Comparing the testing results with the training results, as expected, our training results are more accurate; the model is trained to fit the training data, not the testing data, so, obviously, the model is more well-fit to the training than testing data, and thus shows a higher accuracy on the training data.

The 100% accuracy on the logistic regression training data suggests that both classes can be seperated (with respect to the training data). The high Naive Bayes training accuracy (98%) suggests that the training data can also be well fit with a quadratic decision boundary. This is obviously true, since, if a linear model is sufficient, any higher dimensional model is unecessary, yet can still relatively well fit to the training data. The higher dimensional model, in this case, is less well suited to the training data, due to the above reasoning (assumptions about gaussian data distribution).