Machine Learning Principles Homework 1

1. [Empirical Metric] Suppose we want to learn an ML model for binary classification in the 2-D feature domain and the circle is set as the hypothesis function set. It is defined by the two parameters: center (C_x, C_y) and radius (r), and its classification rule $\delta(x, y)$ as below. We train the circle classifier based on empirical error observed on a training set.

1.1 Suppose a student claims that a good classifier achieves a high precision rate, so we can use a precision metric to train the circle classifier. Do you agree with this claim? If not, what is your reasoning?

I believe that precision alone is not sufficient enough for the training of a classifier. This is because precision is the measurement of the true positives over the the number of true positives plus false positives, so the total number of positives returned by the classifier. This means that if we optimize for precision, the classifier can become overly conservative, by avoiding false positives by shrinking the radius of the circle a bit too much. This would mean that we could lose many actual positive points, resulting in a high precision but poor performance classifier.

1.2 Suppose a student claims that a good classifier achieves a high recall rate, so we can use a recall metric to train the circle classifier. Do you agree with this claim? If not, what is your reasoning?

For context recall measures how many actual positive samples are correctly classified, (as it is the number of true positives divided by the sum of the true positives and false negatives, so the actual positives in the data) so if you optimize solely for recall, the classifier may become overly inclusive by increasing the radius far too much. This could lead to a high number of false positives, which reduces precision and making the classifier unreliable, therefore a balance between precision and recall is necessary for a well-performing classifier.

1.3 Based on the answers, how would you design the learning metric to learn a reasonable circle?

The best method of designing the learning metric to learn a reasonable circle is to optimize for a balanced metric such as F1-score, which would consider both the precision and recall. This means that we would be ensuring that the classifier is not selective nor inclusive. if the dataset has about the same number of positives and negative examples, then accuracy can serve a purpose as well, because it measures in essence the correctness of the predictions in a more general sense. This is specifically for data that may have equal amounts of both, however, if the positives are on the rare side, then you should focus more on the cost and benefits of false positives and false negatives in the context of the problem. If missing actual positives is worse, we might emphasize recall, while if false positives are costly, we prioritize precision. We need to strike the right balance by maximizing true positives while minimizing the impact of errors.

1.4 Even when your metric design is good, you may not learn a good classifier (poor generalization). In what scenarios could this happen?

Even with a well-designed metric, a classifier may struggle in real-world scenarios. Data shifts over time can degrade performance, as the model may not adapt to new patterns. Noisy or mislabeled data can also be problematic—incorrect labels can lead the model to learn the wrong patterns. Another challenge is model mismatch—if the chosen model shape (e.g., a circle) isn't suitable for separating the data, no amount of fine-tuning will make it classify correctly. Poor optimization choices, such as an improper learning rate, can prevent the model from converging to the best solution. Lastly, overfitting is a major concern, where the model learns patterns that work only on the training data but fail to generalize. This often happens when the training set is too small or contains unusual outliers that disrupt learning.

2. [Galton Board] Suppose we have a Galton board as below and run an experiment to drop a ball into it. At each pin, the ball takes a path left or right with equal probability.

2.1 Define the sample spaces Ω1, Ω2, ..., ΩM , where Ωi is the set of all possible sides the ball can take at the level L_i

To define the sample spaces $\Omega_1, \Omega_2, \ldots, \Omega$ for each level L_i, we recognize that at the i-th pin, the ball has an equal chance of moving left or right. Thus, at level L_i, there are exactly two possible outcomes: "Left" or "Right." Therefore, we define each Ω_i as the set $\{\text{Left}, \text{Right}\}$.

2.2 Define the sample space Ω = Ω1 × Ω2 × ...ΩM . Each element of the set Ω encodes a path the ball could take until it arrives at the ground-level LG.

Since the ball moves through M levels, each associated with a two-element sample space, the overall sample space Ω is constructed as the Cartesian product of all individual Ω_i. Specifically, we define Ω as $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_M$, where each element is a sequence of length M representing a complete path through the board. For instance, one possible sequence in Ω could be $(\text{Left}, \text{Right}, \ldots, \text{Left})$.

2.3 What is the meaning of the location at LG where the ball finally arrives? (hint: think about The possible ball path results in different locations, such as the leftmost (blue star) or the second left location (green star).

The final location of the ball at L_G represents the number of times it moved to the right during its descent. Since each pin provides a binary choice (left or right), the position is determined by the total count of rightward moves. If the ball always moves left at each level, it will reach the leftmost position, whereas if it moves left at all but one level, it will land at the second leftmost position. Conversely, the rightmost position is reached if the ball always moves right. This results in a binomial distribution $\text{Bin}(M, 0.5)$, where M is the depth of the board. The distribution follows a symmetric pattern centered around $M/2$, meaning most balls will accumulate near the middle positions, with fewer balls reaching the extreme left or right ends.

2.4 How would you represent the location numerically? Please define a random variable that maps Ω to the numerical values.

The Galton board is a probabilistic system where a ball is dropped from the top and encounters a series of pins, each time making a random decision to move left or right with equal probability. As the ball progresses downward through multiple levels, it follows a unique path determined by its sequence of left and right movements. The final location of the ball at the bottom is dictated by the total number of rightward moves it made during its descent. Since each movement is independent and equally probable, the distribution of final positions follows a binomial pattern. This experiment serves as a physical demonstration of the Central Limit Theorem, as the resulting distribution of ball positions approximates a normal distribution when the depth of the board increases. In this way, we can denote X as $X \sim BIN(M, 0.5)$, where M is the depth of the board, and 0.5 is the probability of moving right at each pin, and X would simply represent the number of times the ball moves right as it descends through M levels.

2.5 Simulate 1,000 ball drops and estimate the Probability Mass Function (PMF) of your random variable for the depth M = 5, M = 10, M=100. Use numpy.random.uniform for your simulation. Plot them and check how your PMFs change as M goes to large. Please explain the phenomenon in relation to Central Limit Theorem.

To analyze the distribution of ball positions in the Galton board, we simulate 1,000 ball drops for different depths $M = 5, 10, 100$. Each simulation follows a binomial distribution $X \sim \text{Bin}(M, 0.5)$, where X represents the number of rightward moves taken by the ball. Using `numpy.random.uniform`,

we generate random paths and compute the Probability Mass Function (PMF) for each depth. By plotting the PMFs, we observe that as M increases, the distribution of final positions approximates a normal distribution. This aligns with the Central Limit Theorem, which states that as the number of independent random events increases, the binomial distribution converges to a normal distribution. The simulation demonstrates that with a small M, the distribution appears discrete and uneven, but as M grows, it becomes smoother and more symmetric around $M/2$, forming a bell-shaped curve.

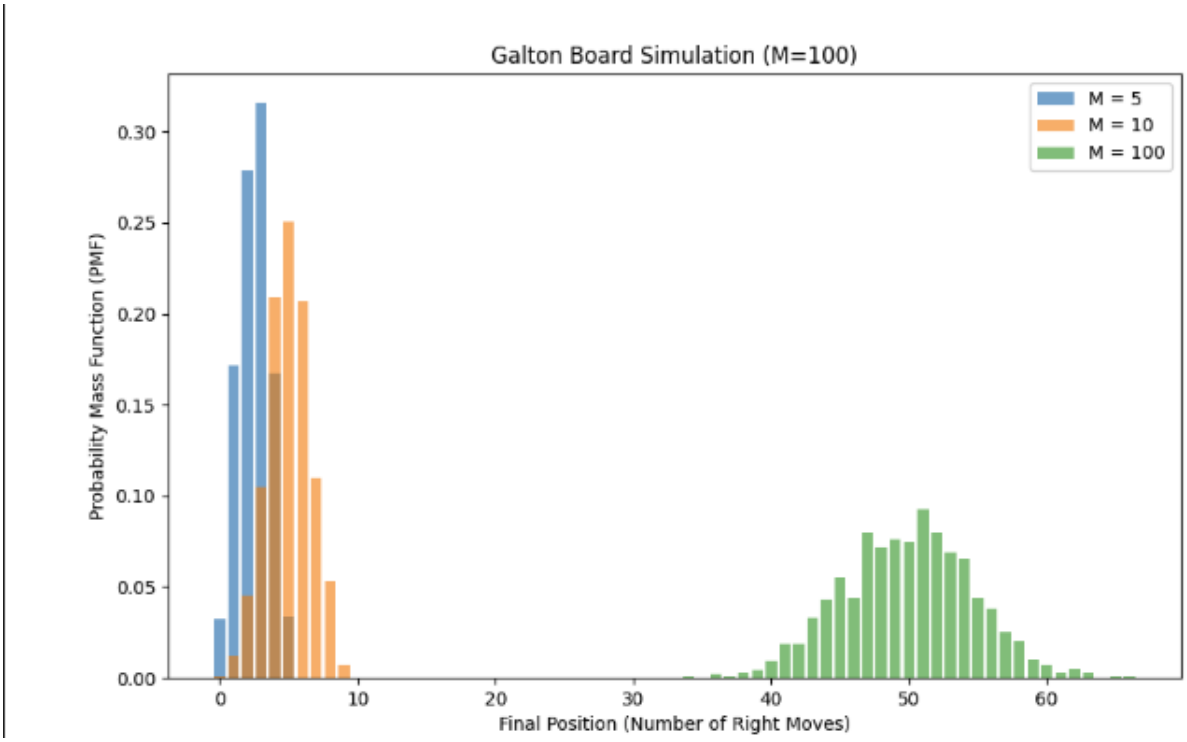

Figure 1: Problem 5 Output

As you can see in the diagram, for a smaller M, the PMF is pretty discrete and more spread out, which reflects the number of possible outcomes. However as M increase, the PMF starts to resemble a genuine distribution, specifically, a Gaussian one. This follows the Central limit theorem because the sum of any independent random variables will approach the CDF of the Gaussian random variable as the total number of variables starts to increase.

3. [**Bayes Rule**] It is known that the disease meningitis causes the patient to have a stiff neck 70 percent of the time. The prior probability of a patient having meningitis is 1/50,000, and the probability of a patient having a stiff neck without meningitis is 0.0099.

S = stiff neck
M = meningitis
no M = no meningitis
$P(S|M) = 0.7$
$P(S|noM) = 0.0099$
P(M) = 1/50000
P(no M) = 49999/50000

3.1 What is the chance that a patient will have a stiff neck?

The probability that a patient will have a stiff neck is the sum of the probabilities of having a stiff neck in both cases. $P(S) = P(S|M) * P(M) + P(S|noM) * P(noM) = 0.7 * 1/50000 + 0.0099 * 49999/50000 = 0.009138$

3.2 What is the chance that a patient will have meningitis given the patient has a stiff neck?

$P(M|S) = P(S|M)P(M)/P(S) = 0.7 * 1/50000/0.01 = 0.0014$

3.3 Suppose there was a sudden epidemic of meningitis, so the probability of meningitis went up significantly. Do you think it will affect the probability of having a stiff neck given that they have meningitis? How about the probability of a patient having meningitis, given that they have a stiff neck?

The probability of a stiff neck given meningitis is $P(S|M)$ which is already specified as 70 percent (0.7). This probability is intrinsic to the condition of meningitis itself and doesn't change based on the overall prevalence of meningitis in the population. Therefore, the probability of having a stiff neck given that you have meningitis will not change.

3.4 Based on the answer in problem 3.3, explain why conditional probability in the causal direction is generally provided as prior knowledge in probabilistic modeling in ML.

Conditional probability in the causal direction is generally treated as prior knowledge because it represents inherent, stable relationships that describe how one event (the cause) influences the probability of another event (the effect). For instance, in the case of meningitis, we know that having meningitis causes a stiff neck in 70 percent of the cases. This causal relationship is fundamental and doesn't fluctuate with changes in external factors like population prevalence. Such knowledge is stable and predictable, making it reliable as prior information in probabilistic models. In contrast, the reverse conditional probability, such as the probability of having meningitis given a stiff neck, is more context-dependent and influenced by factors like the overall prevalence of meningitis in the population, which can change due to circumstances such as an epidemic. Therefore, while the causal direction provides a fixed, data-independent probability that can be modeled as prior knowledge, the reverse direction depends on updated data and needs to be adjusted dynamically. By treating causal relationships as prior knowledge, probabilistic models can rely on a stable foundation while incorporating new evidence to adjust and update posterior probabilities, ensuring that the model is both robust and adaptable.

4. [Naive Bayes] You will build a Naive Bayes Classifier to recognize the species of Iris flower based on the length and width of sepal and petal. Let the random variable I be the species of Iris:
(1) Iris virginica, (2) Iris setosa, and (3) Iris versicolor, and the four random variables SL, SW, PL, and PW denote (1) sepal length, (2) sepal width, (3) petal length, and (4) petal width each. (note: you can use numpy or math but please do not use other libraries like SciPY.)

4.1 Naıve Bayes is a probabilistic model based on Bayes Theorem. Rewrite the formula below as SL, SW , P L, and P W are conditionally independent given I = Iris virginica. Also, write the formulas for the other Iris species: Iris setosa and Iris versicolor. How would you use your three formulas to classify an Iris when you have a record about sepal length, sepal width, petal length, and petal width (sl, sw, pl, pw)?

P [I = Iris virginica—SL = sl, SW = sw, PL = pl, PW = pw]
= P [I = Iris virginica, SL = sl, SW = sw, PL = pl, PW = pw]
P [SL = sl, SW = sw, PL = pl, PW = pw]
= P [SL = sl, SW = sw, PL = pl, PW = pw—I = Iris virginica] · P [I = Iris virginica]
P [SL = sl, SW = sw, PL = pl, PW = pw]

=?

$$\text{P(I} \mid \text{SL} = \text{sl}, \text{SW} = \text{sw}, \text{PL} = \text{pl}, \text{PW} = \text{pw}) = [\text{P(SL} = \text{sl}, \text{SW} = \text{sw}, \text{PL} = \text{pl}, \text{PW} = \text{pw} \mid \text{I})\text{P(I)}]/$$
$$\text{P(SL} = \text{sl}, \text{SW} = \text{sw}, \text{PL} = \text{pl}, \text{PW} = \text{pw})$$

Given that the features (sepal length, sepal width, petal length, petal width) are conditionally independent given the species I, we can rewrite:

$$P(SL = sl, SW = sw, PL = pl, PW = pw \mid I) = P(SL = sl \mid I)P(SW = sw \mid I)P(PL = pl \mid I)P(PW = pw|I)$$

Thus, the probability for a given species I (e.g., Iris virginica) is:
Iris Virginica:

$$P(I = \text{ Iris virginica } \mid SL = sl, SW = sw, PL = pl, PW = pw)$$
$$= \frac{P(I = \text{ Iris virginica, } SL = sl, SW = sw, PL = pl, PW = pw)}{P(SL = sl, SW = sw, PL = pl, PW = pw)}$$
$$= \frac{P(SL = sl, SW = sw, PL = pl, PW = pw \mid I = \text{ Iris virginica }) \cdot P(I = \text{ Iris virginica })}{P(SL = sl, SW = sw, PL = pl, PW = pw)}$$
$$= \frac{P(SL{=}sl|I{=}\text{ Iris virginica }) \cdot P(SW{=}sw|I{=}\text{ Iris virginica }) \cdot P(PL{=}pl|I{=}\text{ Iris virginica }) \cdot P(PW{=}pw|I{=}\text{ Iris virginica }) \cdot P(I{=}\text{ Iris virginica })}{P(SL{=}sl, SW{=}sw, PL{=}pl, PW{=}pw)}$$

As you can see, this method of solving can be used for the other flower species
Iris Setosa:

$$P(I = \text{ Iris setosa } \mid SL = sl, SW = sw, PL = pl, PW = pw)$$
$$= \frac{P(SL = sl, SW = sw, PL = pl, PW = pw \mid I = \text{ Iris setosa }) \cdot P(I = \text{ Iris setosa })}{P(SL = sl, SW = sw, PL = pl, PW = pw)}$$
$$= \frac{P(SL{=}sl|I{=}\text{ Iris setosa }) \cdot P(SW{=}sw|I{=}\text{ Iris setosa }) \cdot P(PL{=}pl|I{=}\text{ Iris setosa }) \cdot P(PW{=}pw|I{=}\text{ Iris setosa }) \cdot P(I{=}\text{ Iris setosa })}{P(SL{=}sl, SW{=}sw, PL{=}pl, PW{=}pw)}$$

Iris Versicolor:

$$P(I = \text{ Iris versicolor } \mid SL = sl, SW = sw, PL = pl, PW = pw)$$
$$= \frac{P(SL = sl, SW = sw, PL = pl, PW = pw \mid I = \text{ Iris versicolor }) \cdot P(I = \text{ Iris versicolor })}{P(SL = sl, SW = sw, PL = pl, PW = pw)}$$
$$= \frac{P(SL{=}sl|I{=}\text{ Iris versicolor }) \cdot P(SW{=}sw|I{=}\text{ Iris versicolor }) \cdot P(PL{=}pl|I{=}\text{ Iris versicolor }) \cdot P(PW{=}pw|I{=}\text{ Iris versicolor }) \cdot P(I{=}\text{ Iris versicolor})}{P(SL{=}sl, SW{=}sw, PL{=}pl, PW{=}pw)}$$

Since the denominator $P(SL = sl, SW = sw, PL = pl, PW = pw)$ is the same for all classes, classification is done by choosing the class I that maximizes: $P(I)P(SL = sl|I)P(SW = sw|I)P(PL = pl|I)P(PW = pw|I)$

This is the Naïve Bayes classification rule: assign the iris to the class with the highest posterior probability (the probability of the hypothesis given the observed evidence which in this case would be the various attributes of the petals).

4.2 The data file "train.csv" contains the records: sepal length, sepal width, petal length, and petal width and Iris species. Based on the formulas you wrote in 4.1, estimate the necessary densities from

"train.csv" by using ML (Maximum Likelihood) estimators we learned in class. We assume each of the four factors, conditioned on a species, follows a Gaussian distribution. Write your code and save it as "yourname-nb-train.py".

Check code file as part of submission

P.S. The training file with output a model with the means and variances of every single frequency class pair, which is then provided as an input for the classifier code.

4.3 Write a Naıve Bayes classifier code "yourname-nb-cls.py" based on your estimated densities.

Check code file as part of submission

4.4 Evaluate your classifier using "test.csv" and report your test accuracy

As you can see I received 100 percent accuracy

```
Sample 1: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 2: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 3: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 4: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 5: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 6: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 7: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 8: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 9: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 10: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 11: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 12: Predicted = Iris-virginica, Actual = Iris-virginica
Sample 13: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 14: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 15: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 16: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 17: Predicted = Iris-virginica, Actual = Iris-virginica
Sample 18: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 19: Predicted = Iris-virginica, Actual = Iris-virginica
Sample 20: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 21: Predicted = Iris-virginica, Actual = Iris-virginica
Sample 22: Predicted = Iris-virginica, Actual = Iris-virginica
Sample 23: Predicted = Iris-virginica, Actual = Iris-virginica
Sample 24: Predicted = Iris-setosa, Actual = Iris-setosa
Sample 25: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 26: Predicted = Iris-virginica, Actual = Iris-virginica
Sample 27: Predicted = Iris-virginica, Actual = Iris-virginica
Sample 28: Predicted = Iris-versicolor, Actual = Iris-versicolor
Sample 29: Predicted = Iris-virginica, Actual = Iris-virginica
Sample 30: Predicted = Iris-virginica, Actual = Iris-virginica

Model Accuracy: 100.00%
```

Figure 2: Classifier Output

5. [Data Whitening] Whitening data is a key prepossessing in ML. By using "x.npz", estimate the first and second order statistics and whiten the data. For your computation, you can use numpy.matmul, numpy.sum, and numpy.linalg.svd.

5.1 Estimate $E[X] = \mu$ and $COV[X, X] = \Sigma$

$$\mu = \frac{1}{N}\sum_{i=1}^{N} X_i$$

The covariance matrix $COV[X, X] = \Sigma$ is given by:

$$\Sigma = \frac{1}{N-1}(X-\mu)(X-\mu)^T$$

Because you must center the initial matrix with $X - \mu$

```
# 5.1

X=np.load("x.npz")["x"]


# First we want to compute the mean of X which would equal E[X]

mu = np.mean(X, axis=1)

N = X.shape[1] # getting the number of samples

# We want to compute the covariance matrix of X which is COV[X, X]


X_centered = X - mu[:, np.newaxis] # We want to cetner the data by subtracting the mean from each individu

COV_X = np.matmul(X_centered, X_centered.T) / (N - 1)

print("\nMean (E[X]):\n", mu)
print("\nCovariance Matrix COV[X, X]: \n", COV_X)
```

Figure 3: 5.1

5.2 Let W be a random vector defined by W = AX + b. Express E[W] and COV [W, W] in terms of $E[X] = \mu$ and $COV[X, X] = \Sigma$.

$$W = AX + b$$

Then, the expected value of W is:

$$E[W] = E[AX + b] = AE[X] + b = A\mu + b$$

Similarly, the covariance matrix of W is:

$$COV[W, W] = COV[AX + b, AX + b] = ACOV[X, X]A^T = A\Sigma A^T$$

```
# 5.2 Let W be a random vector defined by W = AX + b. Express E[W] and COV [W, W] in terms of E[X] = μ and COV [X, X] = Σ

# We will be performing SVD on the covariance matrix, in order to find the whitening matrix of A -> A = S^(-1/2)*U.T

U, S, Vt = np.linalg.svd(COV_X)
print("\nThe Eigenvalues are: \n", S)
print("\nThe EigenVectors are: \n", U)

# With the eigenvalues and eigenvectors we can compute the resulting whitening matrix A
A = np.matmul(np.diag(1.0/np.sqrt(S)), U.T)
print("\nThe Whitening Matrix (A) is: ")
```

Figure 4: 5.2

5.3 Compute A and b to whiten X. i.e. E[W] = 0 and COV [W, W] = I

$$E[W] = 0, \quad COV[W, W] = I$$

We have $E[W] = 0$:

$$A\mu + b = 0 \Rightarrow b = -A\mu$$

We have $COV[W, W] = I$:

$$A\Sigma A^T = I$$

We will use Singular Value Decomposition (SVD), where we can express Σ as:

$$\Sigma = USU^T$$

where U is the eigenvectors and S is a diagonal matrix of the eigenvalues. The formula for the whitening matrix (A) is then as follows:

$$A = US^{-1/2}U^T$$

After transformation, we should have the covariance matrix of W being approximately an identity matrix:

$$COV[W, W] = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 \\ 0.0000 & 0.0000 & 1.0000 \end{bmatrix}$$

This would indicate that we have successfully whitened X.

```
# 5.3 Now we can solve for E[W] and COV[W, W]

"""
    Now that we have the whitening matrix, we can solve for b, which then allows us to solve for W
    W = AX + b

    b = -Au
"""

b = -np.matmul(A, mu)

"""
    Now we can solve for E[W] and COV[W, W]

    E[W] = E[AX + b] = AE[X] + b = Au + b

    COV[W, W] = COV[AX+b, AX+b]

    covariance is not affect by constant shifts, therefore b will disappear
    COV[W,W] = ACOV[X,X]A^T = A * Σ * A^T

    E[W] = 0, COV[W, W] = I
"""

# Compute E[W]
E_W = np.matmul(A, mu) + b
print("\nE[W]: This should just be zeroes\n", E_W)

# Compute the covariance of the this new transformed data
COV_WW = np.matmul(np.matmul(A, COV_X), A.T)
print("\nCovariance matrix: This should be an Identity Matrix:\n", COV_WW)
```

Figure 5: 5.3

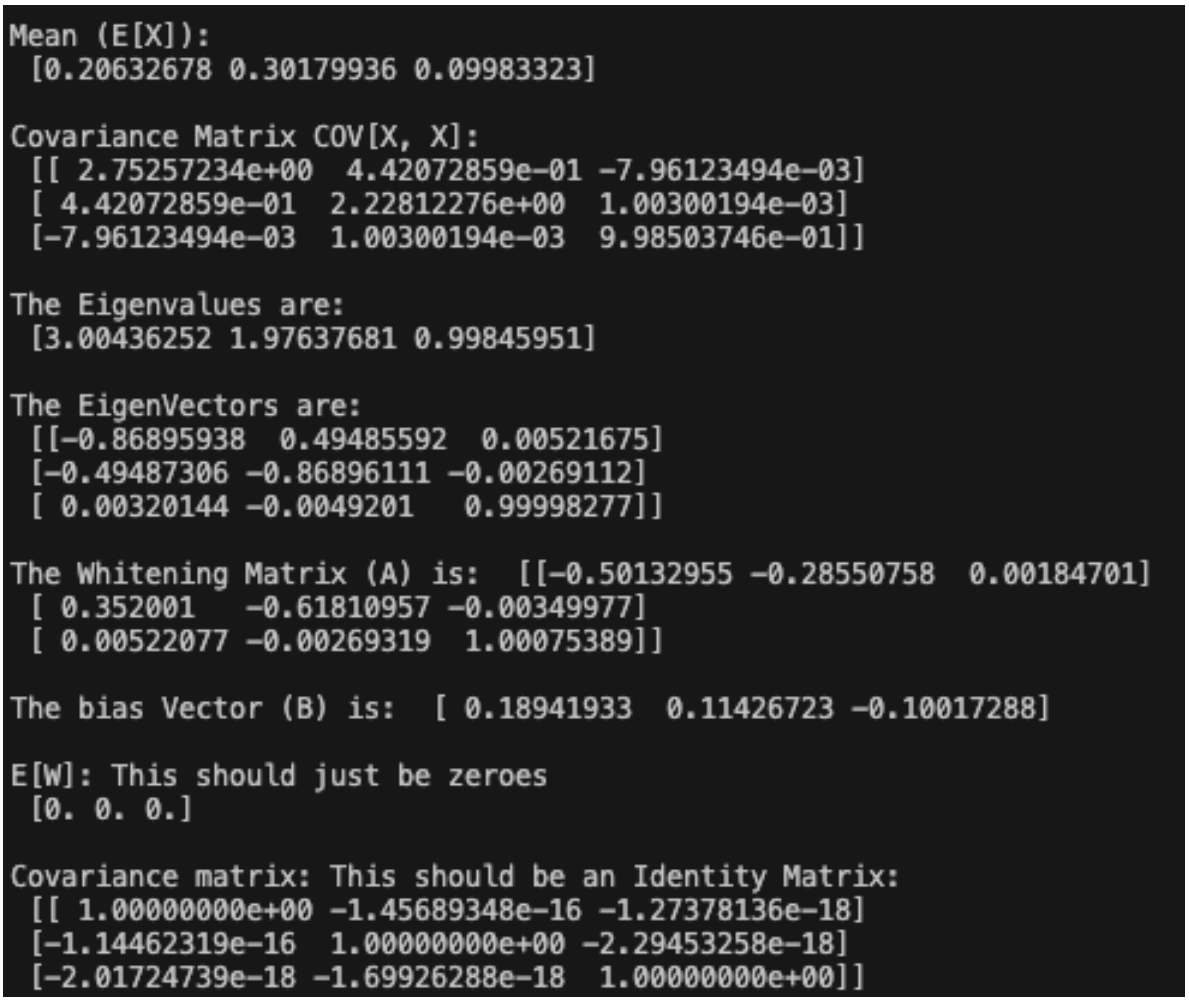

Figure 6: Problem 5 Output

6: [ML and MAP Estimation]

6.1

To determine the maximum likelihood estimate (MLE) of μ, you need to maximize the likelihood function $f(\vec{x} \mid \mu)$. this is done by minimizing the negative log-likelihood, which can be written as:

$$J(\mu) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2\right)$$

Because we are trying to find the minimum, we want to take the derivative of $J(\mu)$ and evaluate it at μ^*:

$$\frac{\partial J(\mu^*)}{\partial \mu} = 0, \quad \text{where } J(\mu) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2\right)$$

$$\frac{\partial J(\mu)}{\partial \mu} = -\frac{1}{\sigma^2}\sum_{n=1}^{N}(x_n-\mu)$$

At the point where this derivation is equal to zero, so $\mu = \mu^*_{\text{ML}}$, we evaluate the derivative as:

$$\frac{\partial J(\mu^*)}{\partial \mu} = 0 \Longrightarrow \sum_{n=1}^{N}(x_n-\mu^*_{\text{ML}}) = 0$$

$$\sum_{n=1}^{N} x_n - N\cdot\mu^*_{\text{ML}} = 0$$

$$N\cdot\mu^*_{\text{ML}} = \sum_{n=1}^{N} x_n \Longrightarrow \mu^*_{\text{ML}} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

This gives us the maximum likelihood estimate of μ, which is:

$$\mu^*_{\text{ML}} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

6.2

We need to basically showcase the likelihood as well as the prior distributions

- The likelihood of the data is:

$$p(\vec{x} \mid \mu) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2\right)$$

- The prior distribution for μ is:

$$p(\mu) = \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right)$$

This means that because that because the posterior $p(\mu \mid \vec{x}) \propto p(\vec{x} \mid \mu) * p(\mu)$, we have:

$$p(\mu \mid \vec{x}) \propto \exp\left(-\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2 - \frac{1}{2\sigma_0^2}(\mu-\mu_0)^2\right)$$

$$-\frac{1}{2\sigma_N^2}(\mu-\mu_N)^2 \quad \text{is the same as} \quad -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2-\frac{1}{2\sigma_0^2}(\mu-\mu_0)^2$$

Combining the terms involving μ^2 and μ, we get:

$$-\frac{1}{2\sigma_N^2}(\mu-\mu_N)^2=-\frac{1}{2}\left[\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right]\mu^2+\mu\left[\frac{\sum_{n=1}^{N}x_n}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}\right]$$

$$\frac{1}{\sigma_N^2}=\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}$$

$$\sigma_N^2=\frac{\sigma^2\sigma_0^2}{N\sigma_0^2+\sigma^2}$$

$$\mu\left[\frac{\sum_{n=1}^{N}x_n}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}\right]$$

$$\mu_N=\sigma_N^2\left(\frac{\sum_{n=1}^{N}x_n}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}\right)$$

$$\mu_N=\frac{\sigma^2}{N\sigma_0^2+\sigma^2}\mu_0+\frac{N\sigma_0^2}{N\sigma_0^2+\sigma^2}\mu_{\mathrm{ML}}^*$$

6.3

$$\mu_{MAP}^*=\frac{\sigma^2}{N\sigma_0^2+\sigma^2}\cdot\mu_0+\frac{N\sigma_0^2}{N\sigma_0^2+\sigma^2}\cdot\mu_{ML}^*$$

- $N\to\infty: \frac{\sigma^2/N}{\sigma_0^2+\sigma^2/N}\cdot\mu_0+\frac{\sigma_0^2}{\sigma_0^2+\sigma^2/N}\cdot\mu_{ML}^*=\mu_{ML}^*$ - $N\to 0: \frac{\sigma^2}{0\cdot\sigma_0^2+\sigma^2}\cdot\mu_0+\frac{0\cdot\sigma_0^2}{0\cdot\sigma_0^2+\sigma^2}\cdot\mu_{ARL}^*=\mu_0$

The formula illustrates how posterior estimation balances the impact of observed data and prior knowledge, depending on the number of data points available. As N approaches ∞, the posterior estimate converges to μ_{ML}^*. Conversely, when N is close to zero, the estimate is solely determined by the mean of the prior distribution, with no contribution from the maximum likelihood estimator. This highlights that relying on MLE is reasonable when a sufficient amount of data is available, making prior knowledge less critical. Moreover, determining the prior distribution is often computationally expensive.