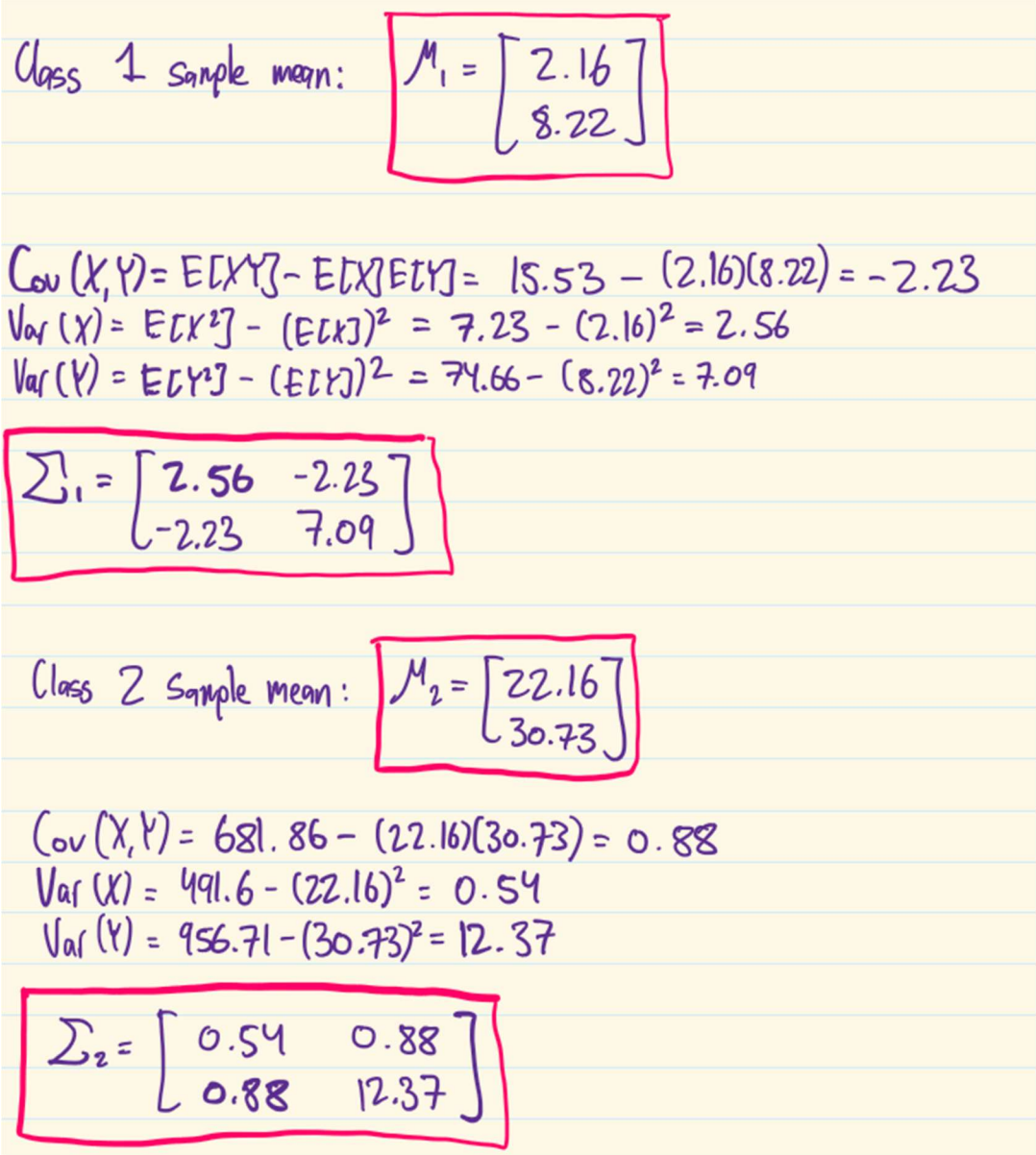
SYDE 572 Assignment 2

Kritik Kaushal, k3kausha@uwaterloo.ca, 20854655

Exercise 1

1. The sample mean and covariance matrices for the two classes were calculated by hand. The results are as follows:



2. Then, the eigenvalues and eigenvectors of each covariance matrix of each class were calculated by hand as shown below:

. Class 1 Covariance Matrix evals and evectors:

$$\Sigma - \lambda I = \begin{bmatrix} 2.56-\lambda & -2.23 \\ -2.23 & 7.09-\lambda \end{bmatrix}$$

$$\begin{aligned} \det(\Sigma-\lambda I) &= (2.56-\lambda)(7.09-\lambda) - (-2.23)(-2.23) \\ &= 18.15 - 2.56\lambda - 7.09\lambda + \lambda^2 - 4.97 \\ &= \lambda^2 - 9.65\lambda + 13.18 \end{aligned}$$

Using quadratic formula: $\boxed{\lambda_1 = 1.65 \\ \lambda_2 = 8.00}$

Eigenvectors: $(\Sigma - \lambda_1 I)v_1 = 0$

$$\begin{bmatrix} 2.56-1.65 & -2.23 \\ -2.23 & 7.09-1.65 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\left.\begin{aligned} 1.91x - 2.23y = 0 \\ -2.23x + 5.44y = 0 \end{aligned}\right\} \text{ Set } x = 1$$

$$y = \frac{2.23}{5.44} = 0.41$$

$$\therefore v_1 = \begin{bmatrix} 1 \\ 0.41 \end{bmatrix} \text{ for } \lambda_1 = 1.65$$

$$(\Sigma - \lambda_2 I)v_2 = 0$$

$$\begin{bmatrix} 2.56-8 & -2.23 \\ -2.23 & 7.09-8 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\left.\begin{aligned} -5.44x - 2.23y = 0 \\ -2.23x - 0.91y = 0 \end{aligned}\right\} \text{ set } x = 1$$

$$y = \frac{2.23}{-0.91} = -2.45$$

$$\therefore v_2 = \begin{bmatrix} 1 \\ -0.91 \end{bmatrix} \text{ for } \lambda_2 = 8.00$$

The same process is repeated for Class 2. The eigenvalues and eigenvectors of the covariance matrix of Class 2 are found as follows:

Class 2 Covariance matrix e-vals and e-vecs:

$$\Sigma_2 - \lambda I = \begin{bmatrix} 0.54-\lambda & 0.88 \\ 0.88 & 12.37-\lambda \end{bmatrix}$$

$$\begin{aligned} \det(\Sigma_2 - \lambda I) &= (0.54-\lambda)(12.37-\lambda) - (0.88)(0.88) \\ &= 6.68 - 0.54\lambda - 12.37\lambda + \lambda^2 - 0.77 \\ &= \lambda^2 - 12.91\lambda + 5.91 \end{aligned}$$

Using quadratic formula: $\boxed{\lambda_1 = 0.48 \\ \lambda_2 = 12.44}$

E-vectors: $(\Sigma_2 - \lambda_1 I)v_1 = 0$

$$\begin{bmatrix} 0.54-0.48 & 0.88 \\ 0.88 & 12.37-0.48 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\left.\begin{aligned} 0.06x + 0.88y &= 0 \\ 0.88x + 11.89y &= 0 \end{aligned}\right\} \text{ set } x = 1$$

$$y = -14.67$$

$$\therefore v_1 = \begin{bmatrix} 1 \\ -14.67 \end{bmatrix} \text{ for } \lambda_1 = 0.48$$

$$(\Sigma_2 - \lambda_2 I)v_2 = 0$$

$$\begin{bmatrix} 0.54-12.44 & 0.88 \\ 0.88 & 12.37-12.44 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\left.\begin{aligned} -11.9x + 0.88y &= 0 \\ 0.88x - 0.07y &= 0 \end{aligned}\right\} \text{ set } x = 1$$

$$y = 0.08$$

$$\therefore v_2 = \begin{bmatrix} 1 \\ 0.08 \end{bmatrix} \text{ for } \lambda_2 = -12.44$$

Then, using the eigenvalues and eigenvectors, the inverse of each covariance matrix is found as follows:

$$\Sigma_1^{-1} = \Phi^T \Lambda^{-1} \Phi$$

Let V be a matrix of eigenvectors:

$$\Sigma_1^{-1} = V \Lambda^{-1} V^{-1}$$

$$= \begin{bmatrix} 1 & 1 \\ 0.41 & -0.91 \end{bmatrix} \begin{bmatrix} 1/1.65 & 0 \\ 0 & 1/8 \end{bmatrix} \frac{1}{\det \begin{bmatrix} 1 & 1 \\ 0.41 & -0.91 \end{bmatrix}} \begin{bmatrix} -0.91 & -1 \\ -0.41 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ 0.41 & -0.91 \end{bmatrix} \begin{bmatrix} 0.61 & 0 \\ 0 & 0.125 \end{bmatrix} \begin{bmatrix} 0.69 & 0.76 \\ 0.31 & -0.76 \end{bmatrix}$$

$$\Sigma_1^{-1} = \begin{bmatrix} 0.45 & 0.37 \\ 0.14 & 0.28 \end{bmatrix}$$

$$\Sigma_2^{-1} = V \Lambda^{-1} V^{-1}$$

$$= \begin{bmatrix} 1 & 1 \\ -14.67 & 0.08 \end{bmatrix} \begin{bmatrix} 1/0.48 & 0 \\ 0 & 1/-2.44 \end{bmatrix} \frac{1}{\det(V)} \begin{bmatrix} 0.08 & -1 \\ 14.67 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 \\ -14.67 & 0.08 \end{bmatrix} \begin{bmatrix} 2.08 & 0 \\ 0 & -0.08 \end{bmatrix} \begin{bmatrix} 0.08/14.75 & -0.08 \\ 0.99 & 0.08 \end{bmatrix}$$

$$\Sigma_2^{-1} = \begin{bmatrix} -0.07 & -0.17 \\ -0.17 & 2.44 \end{bmatrix}$$

3. The decision boundary for the MDD classifier was found as follows:

$$3.\quad x^T(\Sigma_1^{-1}-\Sigma_2^{-1})x + 2[M_2^T\Sigma_2^{-1} - M_1^T\Sigma_1^{-1}]x + M_1^T\Sigma_1^{-1}M_1 - M_2^T\Sigma_2^{-1}M_2 = 0$$

$$x^T\left(\begin{bmatrix}0.45 & 0.37\\ 0.14 & 0.28\end{bmatrix} - \begin{bmatrix}-0.07 & -0.17\\ -0.17 & 2.44\end{bmatrix}\right)x + 2\left[\begin{bmatrix}22.16\\ 30.73\end{bmatrix}^T\begin{bmatrix}-0.07 & -0.17\\ -0.17 & 2.44\end{bmatrix} - \begin{bmatrix}2.16\\ 8.22\end{bmatrix}^T\begin{bmatrix}0.45 & 0.37\\ 0.14 & 0.28\end{bmatrix}\right]x$$

$$+\begin{bmatrix}2.16\\ 8.22\end{bmatrix}^T\begin{bmatrix}0.45 & 0.37\\ 0.14 & 0.28\end{bmatrix}\begin{bmatrix}2.16 & 8.22\end{bmatrix} - \begin{bmatrix}22.16\\ 30.73\end{bmatrix}^T\begin{bmatrix}-0.07 & -0.17\\ -0.17 & 2.44\end{bmatrix}\begin{bmatrix}22.16 & 30.73\end{bmatrix} = 0$$

$$x^T\begin{bmatrix}0.52 & 0.54\\ 0.31 & -2.16\end{bmatrix}x + \begin{bmatrix}-17.79 & 136.22\end{bmatrix}x + [30.07] - [2038.27] = 0$$

$$x^T\begin{bmatrix}0.52 & 0.54\\ 0.31 & -2.16\end{bmatrix}x + \begin{bmatrix}-17.79 & 136.22\end{bmatrix}x - 2008.2 = 0$$

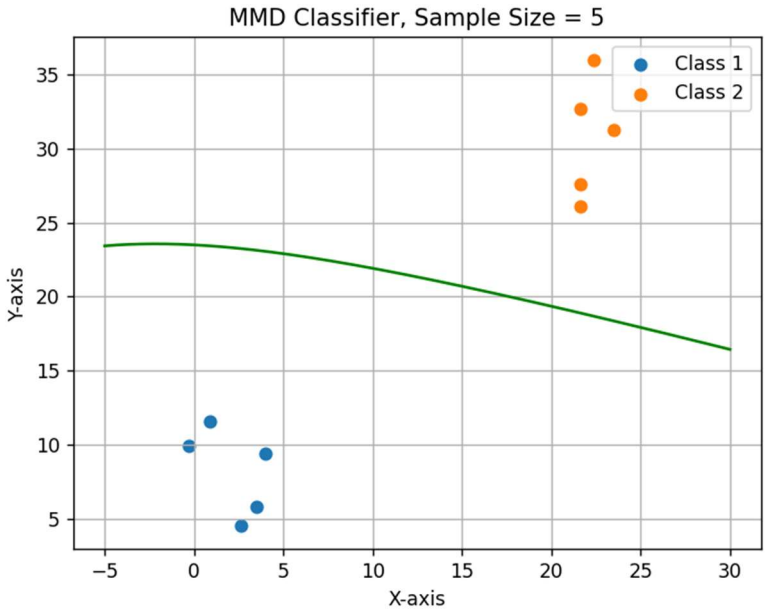
$$[x_1\ x_2]\begin{bmatrix}0.52 & 0.54\\ 0.31 & -2.16\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}-17.79 & 136.22\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} - [2008.2] = 0$$

$$\Rightarrow 0.52x_1^2 + 0.85x_1x_2 - 2.16x_2^2 - 17.79x_1 + 136.22x_2 - 2008.2 = 0$$ } MMD Classification Boundary, by hand

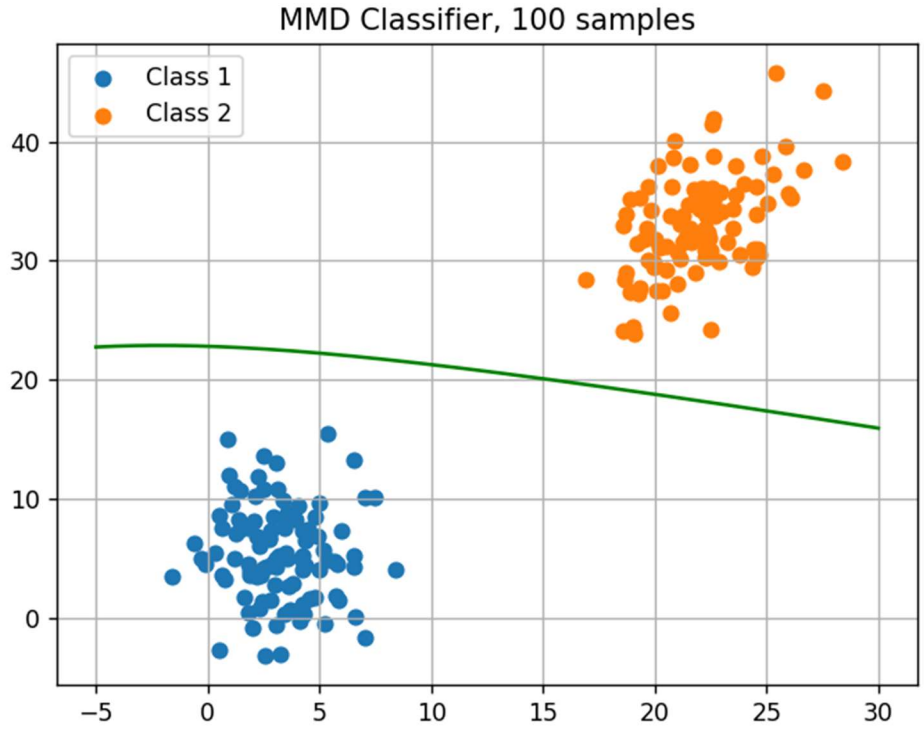
$$x_2 = -\frac{-85x_1 - 13622 + \sqrt{52153x_1^2 + 7786841x_1 + 12050404}}{432}$$

$$x_2 = -\frac{-85x_1 - 13622 - \sqrt{52153x_1^2 + 7786841x_1 + 12050404}}{432}$$

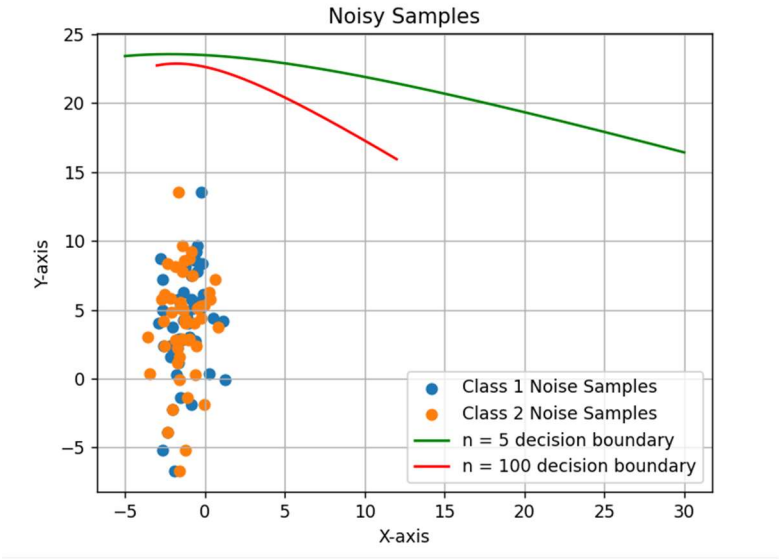
MMD decision boundary and the data were plotted, with great results as shown below. The classifier seems to be working well.



4. Following the same procedure as above, 100 samples were generated for each class. Then, the MDD decision boundary was determined using Python. The boundary and data were plotted, as shown below:



5. White noise was generated and added to each class. The resultant data as well as the two MMD classifiers from before were plotted. See below.



From the figure above, neither classifier worked. The noise had a great effect on the data, making it very hard to fit an MDD decision boundary to it. Both classifiers have an accuracy of 0%.

6. In this case, neither of the two classifiers are good. They both have an accuracy of 0%. However, in general, a higher sample size will often result in a more accurate classifier, so the classifier learned from a sample size of 100 is expected to be better than the one learned from 5 samples.

7. In this case, since the data is very linearly separable, both the MMD and MED classifier have comparable accuracy. In general, MMD classifiers are useful when the data has different variances and is not spherical in shape. MMD classifiers take into account the covariance structure within the data and are more resistant to outliers.

Exercise 2

1. Since we used *random.multivariate_normal*, we can estimate the probability distribution of the data to be Gaussian.

   Then, an ML classifier using the Gaussian distribution was developed. Based on this, I got an accuracy of 50%. The classifier labelled all noisy class 1 data correctly but labelled all the class 2 data incorrectly as class 1.

   My implementation is shown below:

```python
def MLClassifier(x, mean, cov):
    d = x - mean
    inv_cov = np.linalg.inv(cov)
    exponent = -0.5 * np.dot(np.dot(d, inv_cov), d)
    return np.exp(exponent) / np.sqrt((2 * np.pi) ** len(mean) * np.linalg.det(cov))

MLClassificationResults_class1 = []
MLClassificationResults_class2 = []

for new_data in class1_50_new_samples_noisy:
    likelihood_class1 = MLClassifier(new_data, mean_class1, covariance_class1)
    likelihood_class2 = MLClassifier(new_data, mean_class2, covariance_class2)

    if likelihood_class1 > likelihood_class2:
        MLClassificationResults_class1.append(1)
    else:
        MLClassificationResults_class1.append(2)

for new_data in class2_50_new_samples_noisy:
    likelihood_class1_2 = MLClassifier(new_data, mean_class1, covariance_class1)
    likelihood_class2_2 = MLClassifier(new_data, mean_class2, covariance_class2)

    if likelihood_class1_2 > likelihood_class2_2:
        MLClassificationResults_class2.append(1)
    else:
        MLClassificationResults_class2.append(2)

print(MLClassificationResults_class1)
print("\n")
print(MLClassificationResults_class2)
```

   I printed both arrays, and got this:

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]


[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

   The bottom array should be all “2”, but instead they are all “1”. This is likely due to the noise. The noisy data is clustered very close together, which is why the classifier thinks it is all class 1.

2. The MAP classifier was implemented as shown below:

```python
def MAPClassifier(new_point, class_means, class_covariances, priors):

    num_classes = len(class_means)
    posteriors = []

    for i in range(num_classes):
        mean = class_means[i]
        covariance = class_covariances[i]
        prior_prob = priors[i]
        diff = new_point - mean
        likelihood = (1 / (2 * np.pi * np.sqrt(np.linalg.det(covariance)))) * np.exp(-0.5 * np.dot(np.dot(diff, np.linalg.inv(covariance)), diff.T))

        # Calculate posterior probability
        posterior = likelihood * prior_prob
        posteriors.append(posterior)

    # Return the class with the highest posterior probability
    predicted_class = np.argmax(posteriors)
    # Adding +1 to the return because argmax will return either 0 or 1 (based on index), but we have
    # class 1 or class 2. So to be consistent we need to add 1
    return predicted_class+1
```

Then, I ran the classifier for both sets of noisy data. Just like the ML classifier, the MAP classifier classified both sets of noisy data as belonging to class 1. The MAP classifier had an accuracy of 50% on the noisy data.

This is how I tested the MAP classifier, with the priors mentioned in the question:

```python
for x in class1_50_new_samples_noisy:
    predicted_class = MAPClassifier(x, [mean_class1, mean_class2], [covariance_class1, covariance_class2], [0.58, 0.42])
    print(predicted_class)

print("\n")

for x in class2_50_new_samples_noisy:
    predicted_class = MAPClassifier(x, [mean_class1, mean_class2], [covariance_class1, covariance_class2], [0.58, 0.42])
    print(predicted_class)
```

Again, the effect of the noise is evident here. The noise makes the data non-separable, making it hard for both the ML and MAP classifiers to correctly classify data in class 2.

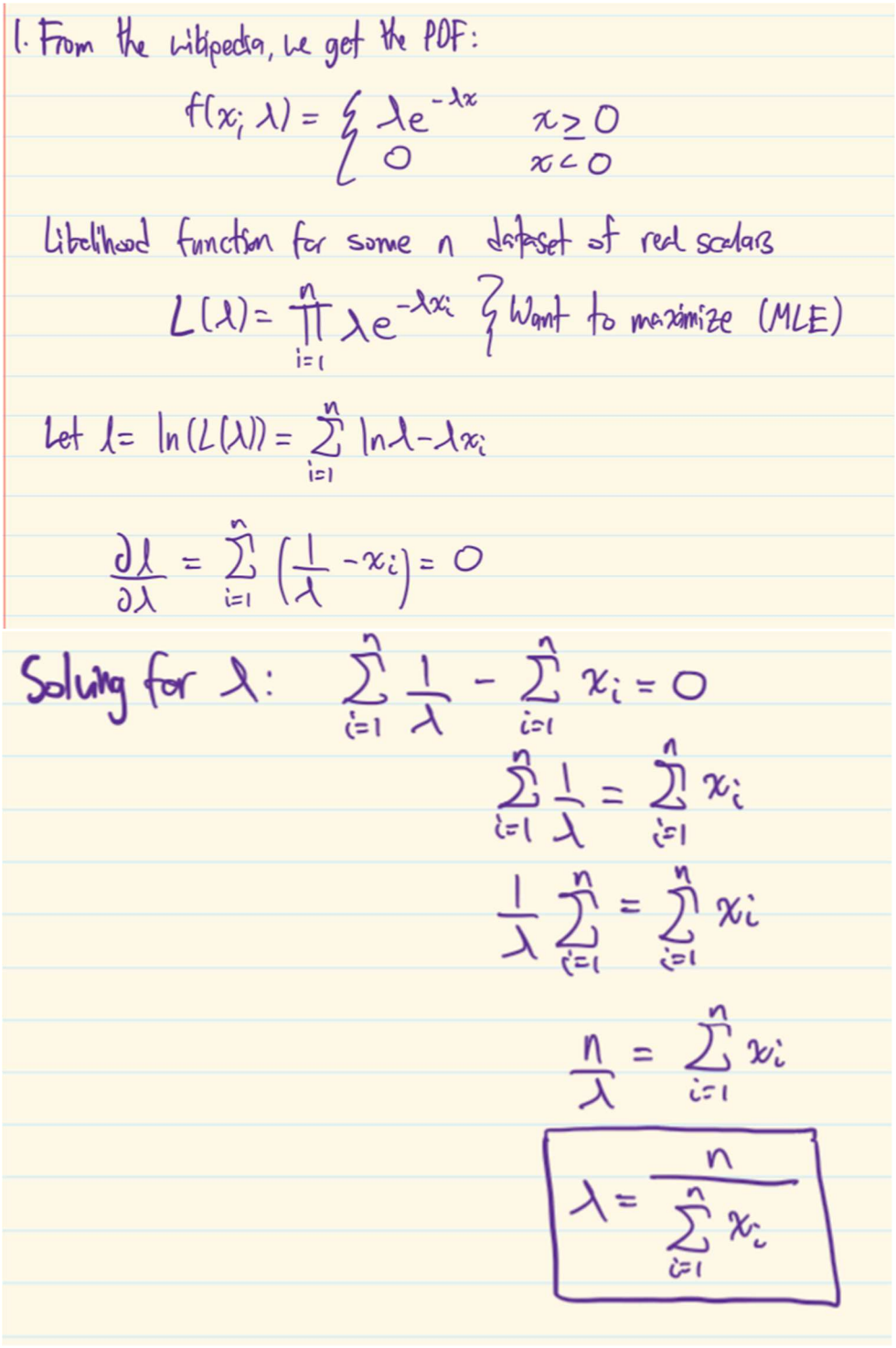
3. I think assuming the probability distributions of the two classes as Gaussian was correct because the numpy function directly says that it is normally distributed random data. I think that removing both the noise from step 1 in exercise 1 as well as the white noise would make both of these classifiers better.

   Removing the noise is an important part of preprocessing in machine learning because noisy data can significantly reduce classifier/model accuracy. This is exactly what we see happening here.

4. Based on the classification accuracy on the noisy data, the ML and MAP classifiers are better than MMD and MED classifiers. However, I think all these classifiers can offer comparable performance with denoised, clean data. MED works well when data clusters are well-separated. ML works well when the data has a known probability distribution. MAP is suited for when prior information about the class probabilities is available. Each type of classification has its own uses, benefits, and disadvantages.

Exercise 3

1. Using MLE, an expression for lambda is derived:



1. From the wikipedia, we get the PDF:

$$f(x;\lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Likelihood function for some n dataset of real scalars

$$L(\lambda) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i}$$ } Want to maximize (MLE)

Let $l = \ln(L(\lambda)) = \sum_{i=1}^{n} \ln\lambda - \lambda x_i$

$$\frac{\partial l}{\partial \lambda} = \sum_{i=1}^{n} \left(\frac{1}{\lambda} - x_i\right) = 0$$

Solving for λ: $\sum_{i=1}^{n} \frac{1}{\lambda} - \sum_{i=1}^{n} x_i = 0$

$$\sum_{i=1}^{n} \frac{1}{\lambda} = \sum_{i=1}^{n} x_i$$

$$\frac{1}{\lambda} \sum_{i=1}^{n} = \sum_{i=1}^{n} x_i$$

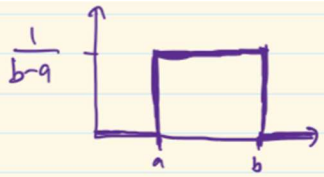
$$\frac{n}{\lambda} = \sum_{i=1}^{n} x_i$$

$$\boxed{\lambda = \frac{n}{\sum_{i=1}^{n} x_i}}$$

2. For a uniform distribution, the expressions for a and b are derived:

2. PDF of uniform distribution is:

$$f(x; a, b) = \frac{1}{b-a}$$



Likelihood function: $L(a,b) = \left(\frac{1}{b-a}\right)^n$ } Want to maximize this

wrt a, $L(a,b)$ is maximized when 'a' is the minimum value in dataset
wrt b, $L(a,b)$ is maximized when 'b' is the maximum value in dataset } This makes sense intuitively

Let X be $\{x_1, x_2, \cdots, x_n\}$

Then, $\boxed{\begin{aligned} a &= \min(X) \\ b &= \max(X) \end{aligned}}$

3. Using MLE, expressions for deriving each parameter are shown below:

$$3.\ p(X|\Theta) = \frac{1}{2}\left(\mathcal{N}(X|\theta_1,\theta_2) \times f(x;\lambda)\right)$$

$$\text{PDF: } \frac{1}{2}\left(\frac{1}{\sqrt{2\pi}\theta_2} e^{-\frac{1}{2}\left(\frac{x-\theta_1}{\theta_2}\right)^2} \cdot \lambda e^{-\lambda x}\right)$$

$$L(\theta_1,\theta_2,\lambda) = \frac{1}{2}\prod_{i=1}\left(\frac{1}{\sqrt{2\pi}\theta_2} e^{-\frac{1}{2}\left(\frac{x-\theta_1}{\theta_2}\right)^2} \cdot \lambda e^{-\lambda x}\right)$$

$$l(\theta_1,\theta_2,\lambda) = \ln(L(\theta_1,\theta_2,\lambda)) = \ln\left(\frac{1}{2}\left(\prod_{i=1}^{n}\left(\frac{1}{\sqrt{2\pi}\theta_2} e^{-\frac{1}{2}\left(\frac{x-\theta_1}{\theta_2}\right)^2} \cdot \lambda e^{-\lambda x}\right)\right)\right)$$

$$= \frac{1}{2}\sum_{i=1}^{n}\left(-\frac{1}{2}\ln(2\pi\theta_2) - \frac{(x_i-\theta_1)^2}{2\theta_2^2} + \ln(\lambda) - \lambda x_i\right)$$

Then, we can get the expressions for θ_1, θ_2, and λ by taking the respective partial derivatives of $\ell(\theta_1, \theta_2, \lambda)$, and set it to 0 (maximize)

For θ_1: $$\frac{d\ell}{d\theta_1} = \sum_{i=1}^{n}\left(\frac{x_i - \theta_1}{\theta_2}\right) = 0$$

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

$$\boxed{\theta_1 = \frac{1}{n}\sum_{i=1}^{n} x_i} \Rightarrow \text{sample mean, from Gaussian distribution, makes sense intuitively}$$

For θ_2: $$\frac{d\ell}{d\theta_2} = \frac{d}{d\theta_2}\left(\sum_{i=1}^{n}\left(-\frac{1}{2}\ln(2\pi\theta_2) - \frac{(x_i - \theta_1)^2}{2\theta_2^2}\right)\right)$$

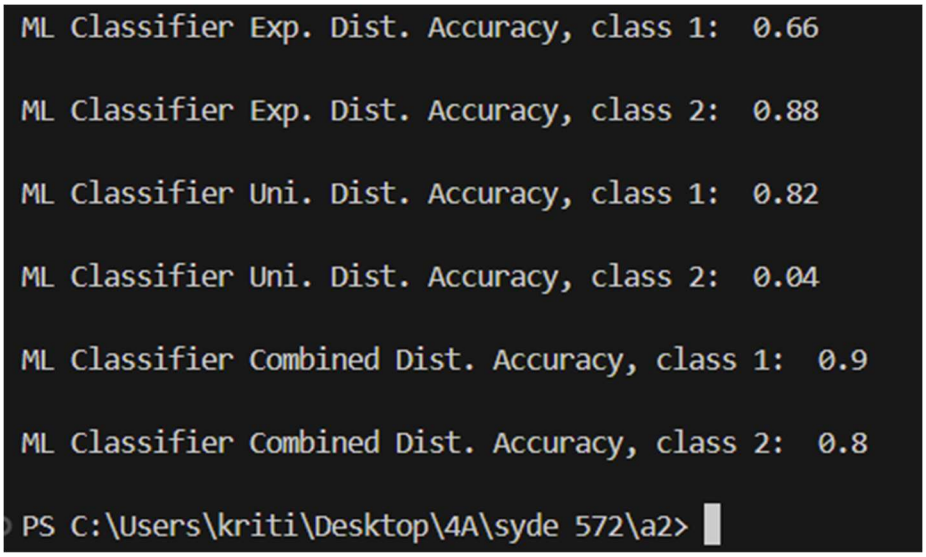
$$= \sum_{i=1}^{n}\left(-\frac{1}{2\theta_2} + \frac{(x_i - \theta_1)^2}{2\theta_2^2}\right) = 0$$

$$\boxed{\theta_2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \theta_1)^2} \Rightarrow \text{sample variance, makes sense intuitively}$$

For λ: $$\frac{d\ell}{d\lambda} = \frac{d}{d\lambda}\left(\frac{1}{2}\sum_{i=1}^{n}\ln\lambda - \lambda x_i\right)$$

Intuitively: $$\boxed{\lambda = \frac{n}{\sum_{i=1}^{n} x_i}} \Rightarrow \text{same as previous problem, makes sense intuitively}$$

4. Using the derived expressions above, ML Classifiers were developed for each type of probability distribution (see python file, section is labeled Exercise 3).
   The ML classifiers were tested on noisy data. Below are the accuracy results for each type of classifier.



As can be seen from above, the classifier with the combined Gaussian and exponential distribution has the highest accuracy (85%). This makes sense because the function used to generate the random 1D data was *np.random.normal*. This data is normally distributed, so the "Gaussian part" of our Gaussian-exponential ML classifier gives us higher accuracy compared to the purely exponential distribution ML classifier and uniform distribution ML classifier.

Also, the white noise we added to make the data noisy is also normally distributed. Thus, it makes sense to see the highest accuracy with the combined Gaussian-exponential ML classifier.