ISYE 6740 - Summer 2024

Homework 3

Parth Patel

1 - Implementing EM for MNIST dataset

I will implement the EM algorithm for fitting a Gaussian mixture model for the MNIST handwritten digits dataset. For this question, we will reduce the dataset to be only two cases, of digits "2" and "6" only. Thus, I will fit GMM with $C = 2$ using the data file `data.mat` or `data.dat`. True label of the data are also provided in `label.mat` and `label.dat`.

The matrix `images` is of size 784-by-1990, i.e., there are 1990 images in total, and each column of the matrix corresponds to one image of size 28-by-28 pixels (the image is vectorized; the original image can be recovered by mapping the vector into a matrix).

First, I will use PCA to reduce the dimensionality of the data before applying to EM. I will put all "2" and "6" digits together, to project the original data into 4-dimensional vectors.

Next, I will implement the EM algorithm for the projected data with 4-dimensions. In this question, I use the same set of data from the provided data files for training and testing.

1.1

Here, I will implement EM algorithm by myself using the following initialization:

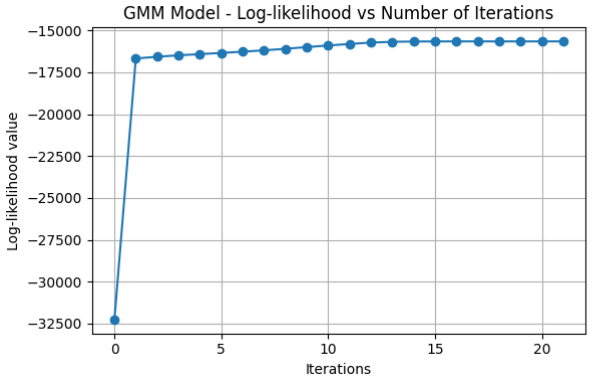
- initialization for mean: random Gaussian vector with zero mean
- initialization for covariance: generate two Gaussian random matrix of size n-by-n: S_1 and S_2, and initialize the covariance matrix for the two components are $\Sigma_1 = S_1 S_1^T + I_n$, and $\Sigma_2 = S_2 S_2^T + I_n$, where I_n is an identity matrix of size n-by-n.

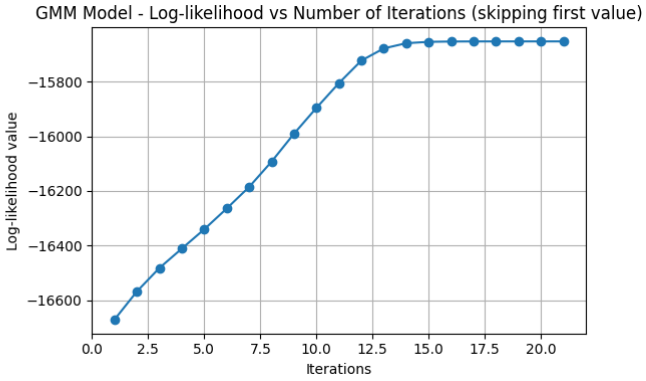
I will also plot the log-likelihood function versus the number of iterations to show that the algorithm is converging.

```
Running EM algorithm with set seed = 123456

Training converged in 21 iterations.

Progress: 100%|     | 100/100 [00:02<00:00, 38.94it/s]
```



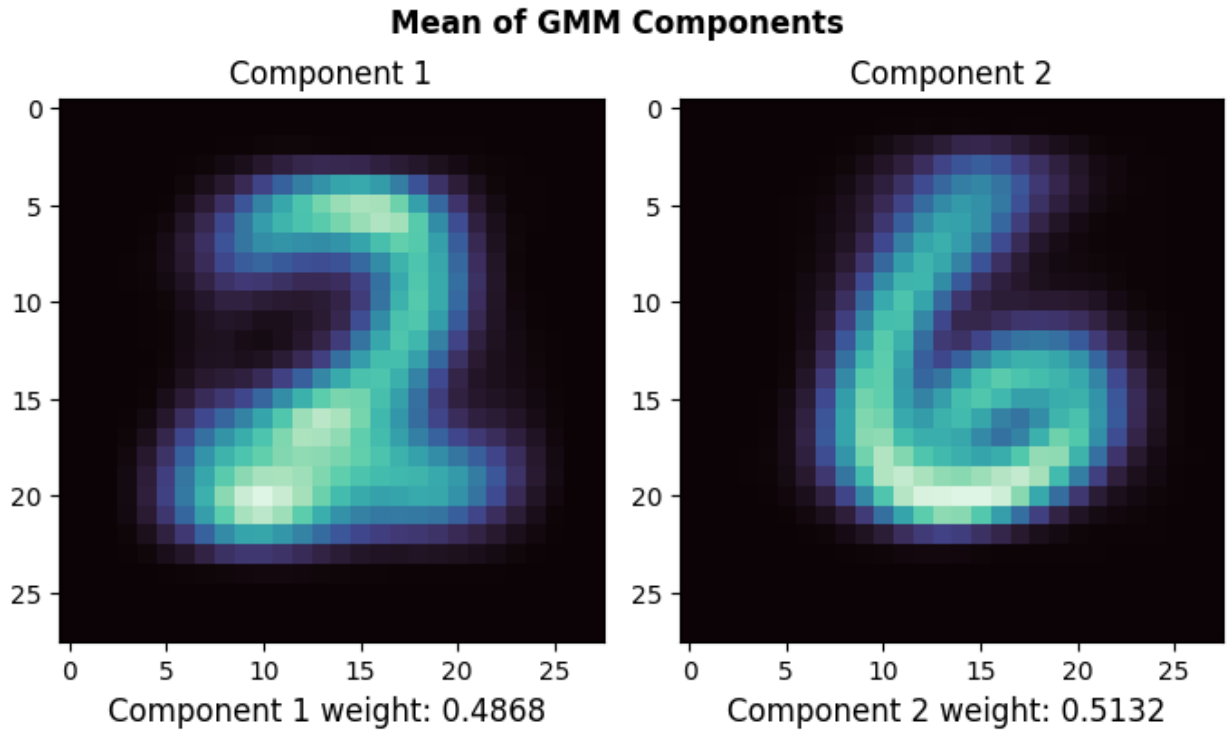


As seen from the plots above, the EM algorithm that I implemented from scratch using the demo code provided from the lectures converges after 21 iterations for this particular seed. I created two plots, one with all the values and another with the first value removed to better visualize the gradual converging of the algorithm. The performance of the EM algorithm can be seen in the following two animations:

- EM Algorithm - 21 Iterations
- EM Algorithm - 44 Iterations

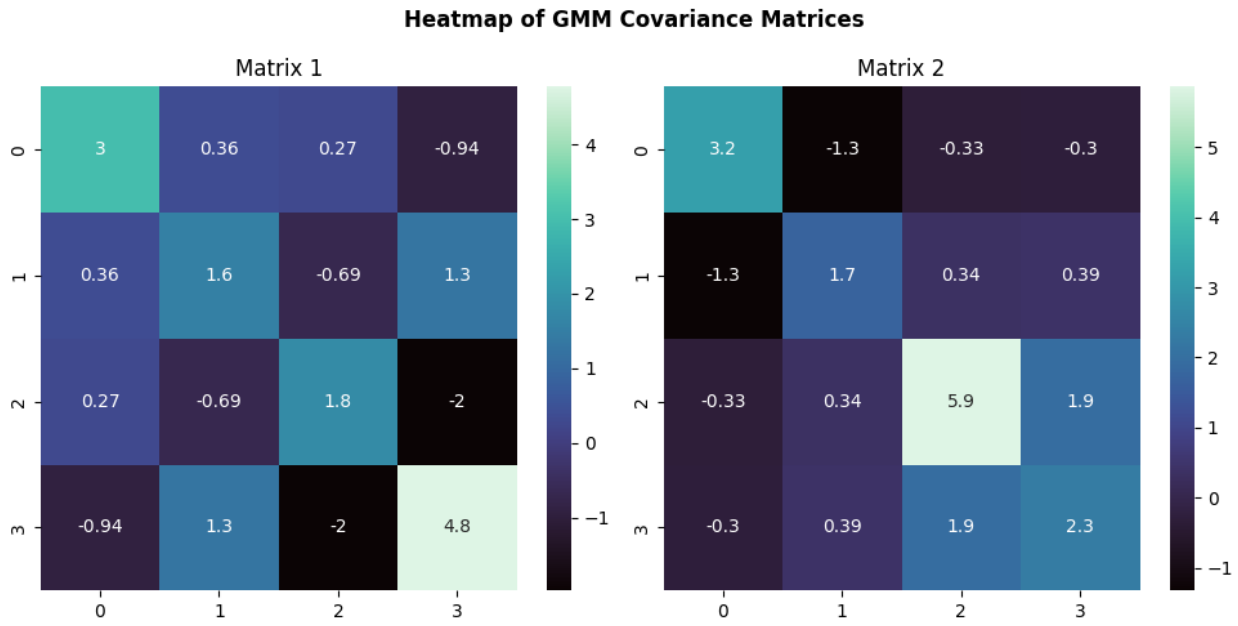
1.2

After the GMM was fitted, I plotted the mean of each component by mapping it back to the original space and reformated the vectors into 28-by-28 matrices.



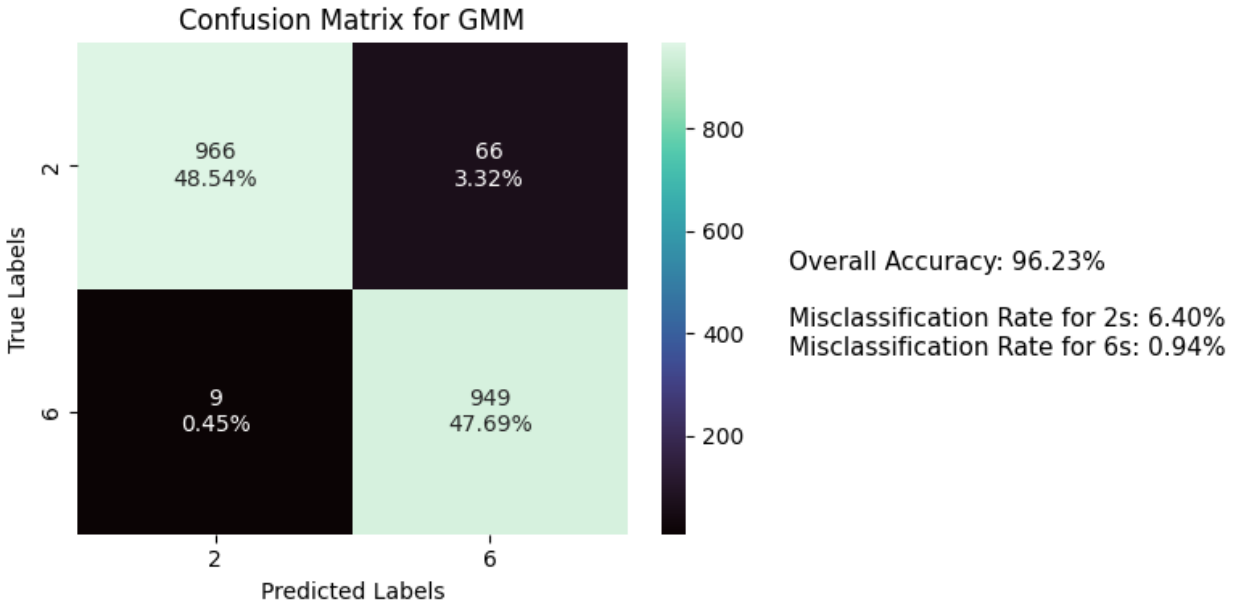
The means are displayed as images, corresponding to a kind of "average" of the images. Below each image, I have also provided the numerical weights of each component. Component 2 seems to have a slightly higher weight than component 1 which might indicate that it was a better fit with less errors. This can be verified in a moment with the misclassification results.

I have also plotted two 4-by-4 covariance matrices by visualizing their intensities using a heatmap. These are shown below for each component.

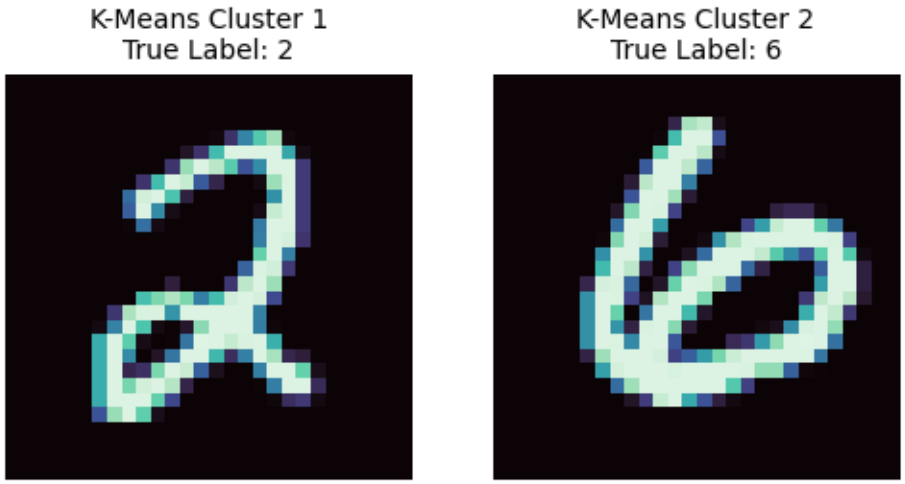


1.3

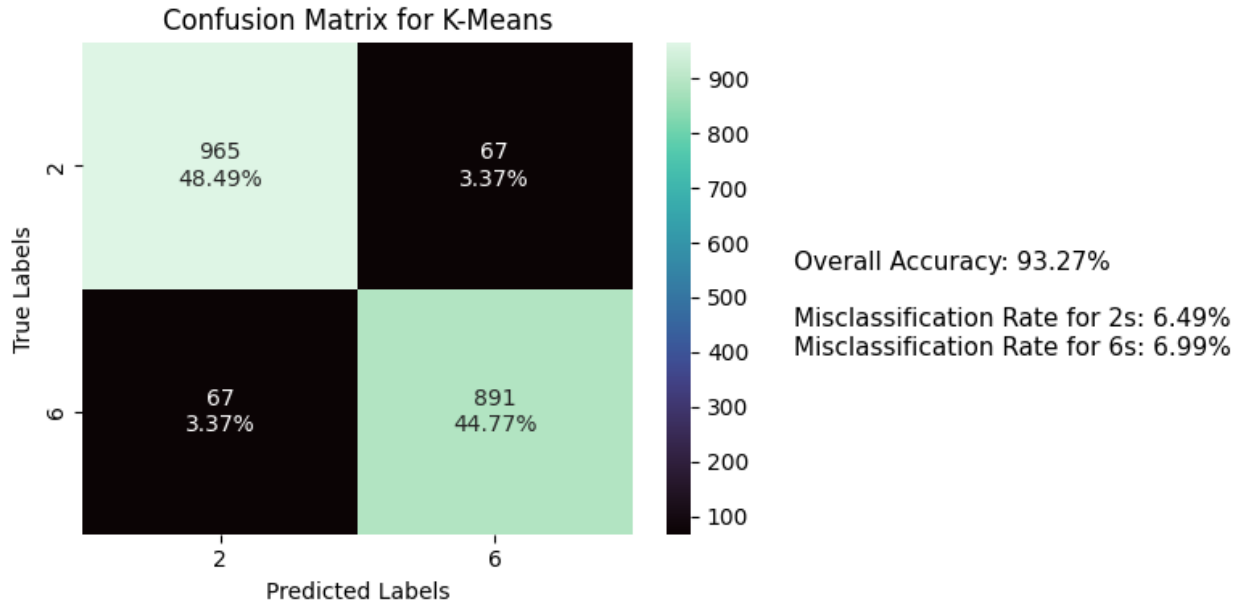
By using the τ_k^i to infer the labels of the images and compare with the true labels, I can calculate the mis-classification rate (1 - Accuracy) for digits 2 and 6 respectively. I also performed K-means clustering with $K = 2$ and found the mis-classification rate for digits 2 and 6 respectively.



The confusion matrix above shows that the GMM model had an overall accuracy of 96.23% with misclassification rates of 6.40% for images of the digit `2` and 0.94% for images of the digit `6`. The model was able to fit images of the digit `6` much better than images of the digit `2` and this was implied earlier when component 2 corresponding to the digit `6` had a slightly higher weight.



I also performed K-means clustering on the the reduced image dataset with 4 dimensions using PCA and the plot of each cluster is shown above. This was important to plot since depending on the seed for randomization, cluster 1 could show images of `6` instead of `2` which would not match with the GMM model from before. I simply plotted the clusters to make sure that my function that calculates misclassification rates for the labels worked correctly by mapping predicated labels of `0` to `2` and `1` to `6`.



As seen from the confusion matrix for the K-Means algorithm output above, the K-Means algorithm performed slightly worse than the GMM model. The overall accuracy was lower (93.27% vs 96.23% of the GMM model) and the misclassification rates for both digits were higher than the GMM model. For the digit `2`, the misclassification rate was 6.49% which is slightly higher than the 6.40% rate from the GMM model, and for the digit `6`, the misclassification rate was 6.99% which is also much higher than the 0.94% rate from the GMM model.

Overall, the GMM model with the EM algorithm had better performance than the K-means algorithm in identifying the correct labels for the MNIST dataset.

2 - Optimization

Consider a simplified logistic regression problem. Given m training samples (x^i, y^i), $i = 1, \ldots, m$, the data $x^i \in \mathbb{R}$, and $y^i \in \{0, 1\}$. In order to fit a logistic regression model for classification, we solve the following optimization problem, where $\theta \in \mathbb{R}$ is a parameter we aim to find:

$$\max_{\theta} \ell(\theta), \tag{1}$$

where the log-likelihood function

$$\ell(\theta) = \sum_{i=1}^{m} \left\{ -\log(1 + \exp\{-\theta^T x^i\}) + (y^i - 1)\theta^T x^i \right\}.$$

2.1

Question: Show step-by-step mathematical derivation for the gradient of the cost function $\ell(\theta)$ in (1).

To get the gradient of the cost function $\ell(\theta)$, I can take the derivative of $\ell(\theta)$.

$$\ell'(\theta) = \sum_{i=1}^{m} \frac{d}{d\theta} \left\{ -\log(1 + \exp\{-\theta x^i\}) \right\} + \frac{d}{d\theta} \left\{ (y^i - 1)\theta x^i \right\}$$

I can break the expression into two parts and find the derivatives separately.

Derivative of $(-\log(1 + \exp\{-\theta x^i\})$:

$$f(\theta) = -\log(1 + \exp\{-\theta x^i\})$$

Using the chain rule:

$$\frac{df(\theta)}{d\theta} = -\frac{1}{1 + \exp\{-\theta x^i\}} \cdot \frac{d}{d\theta}(1 + \exp\{-\theta x^i\})$$

$$\frac{df(\theta)}{d\theta} = -\frac{1}{1 + \exp\{-\theta x^i\}} \cdot (-x^i \exp\{-\theta x^i\}) = \frac{x^i \exp\{-\theta x^i\}}{1 + \exp\{-\theta x^i\}}$$

Derivative of $(y^i - 1)\theta x^i$:

$$g(\theta) = (y^i - 1)\theta x^i$$

$$\frac{dg(\theta)}{d\theta} = (y^i - 1)x^i$$

Combining both parts:

$$\boxed{\ell'(\theta) = \sum_{i=1}^{m} \left\{ \frac{x^i \exp\{-\theta x^i\}}{1 + \exp\{-\theta x^i\}} + (y^i - 1)x^i \right\}}$$

2.2

> **Question:** Write a pseudo-code for performing **gradient descent** to find the optimizer θ^*. This is essentially what the training procedure does.

Initialize parameters θ^0, tolerance level ϵ, max iterations T, and step size or learning rate $\gamma > 0$.

Iteratively do:

- $\theta^{t+1} \leftarrow \theta^t + \gamma \sum_{i=1}^{m} \left\{ \frac{x^i \exp\{-\theta^t x^i\}}{1+\exp\{-\theta^t x^i\}} + (y^i - 1)x^i \right\}$

While $\|\theta^{t+1} - \theta^t\| > \epsilon$ and $t \leq T$.

2.3

> **Question:** Write the pseudo-code for performing the **stochastic gradient descent** algorithm to solve the training of logistic regression problem (1). Please explain the difference between gradient descent and stochastic gradient descent for training logistic regression.

The pseudo-code for performing the stochastic gradient descent is:

Initialize parameter θ^0, tolerance level ϵ, number of subsets K, max iterations T, and initial step size or learning rate $\gamma^t > 0$.

Randomly split the dataset into subsets S_k, where $k = 1, 2, ..., K$.

For each subset S_k, do:

- $\theta^{t+1} \leftarrow \theta^t + \gamma^t \sum_{i \in S_k} \left\{ \frac{x^i \exp\{-\theta^t x^i\}}{1+\exp\{-\theta^t x^i\}} + (y^i - 1)x^i \right\}$
- $\gamma^{t+1} = \frac{\gamma^t}{(t+1)}$

While $\|\theta^{t+1} - \theta^t\| > \epsilon$ and $t \leq T$.

The difference between gradient descent and stochastic gradient descent for training logistic regression is that stochastic gradient descent uses a randomly sampled small subset at each iteration whereas gradient descent uses the whole data set to compute the gradient. Due to this difference, gradient descent converges very slowly for large datasets while stochastic gradient decent can converge much faster since it uses a small subset of the data for each iteration.

2.4

> **Question:** We will **show that the training problem in basic logistic regression problem is concave.** Derive the Hessian matrix of $\ell(\theta)$ and based on this, show the training problem (1) is concave. Explain why the problem can be solved efficiently and gradient descent will achieve a unique global optimizer, as we discussed in class.

To show that the training problem in basic logistic regression is concave, the Hessian matrix of $\ell(\theta)$ must be negative semi-definite. To get the Hessian matrix, I can take the second derivative of the gradient of the cost function.

Given the first derivative of the log-likelihood function:

$$\ell'(\theta) = \sum_{i=1}^{m} \left\{ \frac{x^i \exp\{-\theta x^i\}}{1 + \exp\{-\theta x^i\}} + (y^i - 1)x^i \right\}$$

I can take the second derivative:

$$\ell''(\theta) = \frac{d}{d\theta} \left(\sum_{i=1}^{m} \left\{ \frac{x^i \exp\{-\theta x^i\}}{1 + \exp\{-\theta x^i\}} + (y^i - 1)x^i \right\} \right)$$

The second derivative of the term $(y^i - 1)x^i$ is zero because it does not depend on θ.

Differentiating the term $\frac{x^i \exp\{-\theta x^i\}}{1+\exp\{-\theta x^i\}}$ by using the quotient rule for differentiation:

$$\frac{d}{d\theta}\left(\frac{x^i \exp\{-\theta x^i\}}{1+\exp\{-\theta x^i\}}\right) = \frac{(-(x^i)^2 \exp\{-\theta x^i\})(1+\exp\{-\theta x^i\}) - (x^i \exp\{-\theta x^i\})(-x^i \exp\{-\theta x^i\})}{(1+\exp\{-\theta x^i\})^2}$$

$$= \frac{-(x^i)^2 \exp\{-\theta x^i\}}{(1+\exp\{-\theta x^i\})^2}$$

Simplifying, we get the Hessian Matrix:

$$\ell''(\theta) = \sum_{i=1}^{m}\left(\frac{-(x^i)^2 \exp\{-\theta x^i\}}{(1+\exp\{-\theta x^i\})^2}\right)$$

This Hessian Matrix is always negative, meaning that the log-likelihood function $\ell(\theta)$ is concave which ensures that there is a unique global maximum. This is an efficient solution because the gradient ascent will efficiently converge to the unique global optimizer, guaranteeing that logistic regression can be solved effectively with a unique global solution.

3 - Bayes Classifier for spam filtering

In this problem, I will use the Bayes Classifier algorithm to fit a spam filter by hand. This will enhance my understanding of the Bayes classifier and build intuition. This question does not involve any programming but only derivation and hand calculation. Tools can be used (Python, Excel, Etc.) but all calculations and derivations will still be provided.

Spam filters are used in all email services to classify received emails as `Spam` or `Not Spam`. A simple approach involves maintaining a vocabulary of words that commonly occur in `Spam` emails and classifying an email as `Spam` if the number of words from the dictionary that are present in the email is over a certain threshold. We are given the vocabulary consists of 15 words $V = \{$secret, offer, low, price, valued, customer, today, dollar, million, sports, is, for, play, healthy, pizza$\}$.

We will use V_i to represent the ith word in V. As our training dataset, we are also given 3 example spam messages,

- million dollar offer for today
- secret offer today
- secret is secret

and 4 example non-spam messages

- low price for valued customer today
- play secret sports today
- sports is healthy
- low price pizza today

Recall that the Naive Bayes classifier assumes the probability of an input depends on its input feature. The feature for each sample is defined as $x^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_d^{(i)}]^T$, $i = 1, \ldots, m$ and the class of the ith sample is $y^{(i)}$. In our case the length of the input vector is $d = 15$, which is equal to the number of words in the vocabulary V. Each entry $x_j^{(i)}$ is equal to the number of times word V_j occurs in the i-th message.

3.1

I can calculate class prior $\mathbb{P}(y = 0)$ and $\mathbb{P}(y = 1)$ from the training data, where $y = 0$ corresponds to spam messages, and $y = 1$ corresponds to non-spam messages. Note that these class prior essentially corresponds to the frequency of each class in the training sample. I will also write down the feature vectors for each spam and non-spam messages.

To first calculate the class priors $\mathbb{P}(y = 0)$ and $\mathbb{P}(y = 1)$, I need to determine the frequency of each class in the training data. Here, $y = 0$ corresponds to spam messages, and $y = 1$ corresponds to non-spam messages.

$$\mathbb{P}(y = 0) = \frac{\text{Number of spam messages}}{\text{Total number of messages}} = \frac{3}{7}$$

$$\mathbb{P}(y = 1) = \frac{\text{Number of non-spam messages}}{\text{Total number of messages}} = \frac{4}{7}$$

Next, I can get the feature vectors for each spam and non-spam message using Python. I can represent each message as a vector $x^{(i)}$ where each entry $x_j^{(i)}$ is the number of times word V_j occurs in the i-th message. The Python code can be found in the attached zip file.

Table 1: Feature Vectors of Spam Messages

Spam Message	Feature Vector
million dollar offer for today	[0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0]
secret offer today	[1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
secret is secret	[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

Table 2: Feature Vectors of Non-Spam Messages

Non-Spam Message	Feature Vector
low price for valued customer today	[0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0]
play secret sports today	[1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0]
sports is healthy	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0]
low price pizza today	[0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1]

3.2

Assuming the keywords follow a multinomial distribution, the likelihood of a sentence with its feature vector x given a class c is given by

$$\mathbb{P}(x|y=c) = \frac{n!}{x_1!\cdots x_d!}\prod_{k=1}^{d}\theta_{c,k}^{x_k}, \quad c=\{0,1\}$$

where $n = x_1 + \cdots x_d$, $0 \leq \theta_{c,k} \leq 1$ is the probability of word k appearing in class c, which satisfies

$$\sum_{k=1}^{d}\theta_{c,k} = 1, \quad c=\{0,1\}.$$

Given this, the complete log-likelihood function for our training data is given by

$$\ell(\theta_{0,1},\ldots,\theta_{0,d},\theta_{1,1},\ldots,\theta_{1,d}) = \sum_{i=1}^{m}\sum_{k=1}^{d} x_k^{(i)} \log \theta_{y^{(i)},k}$$

(In this example, $m = 7$). I can calculate the maximum likelihood estimates of $\theta_{0,1}$, $\theta_{0,7}$, $\theta_{1,1}$, $\theta_{1,15}$ by maximizing the log-likelihood function above.

(Note/Hint: We are solving a constrained maximization problem and will need to introduce Lagrangian multipliers and consider the Lagrangian function.)

Steps to calculate the MLE estimates for $\theta_{c,k}$:

To start, I can first get the joint probability of y (class label) and X (feature vector).

Given the multinomial distribution assumption, the joint probability of y and X is:

$$\mathbb{P}(y=c, X=x) = \mathbb{P}(y=c)\cdot\mathbb{P}(X=x \mid y=c)$$

where:

- $\mathbb{P}(y=c)$ is the prior probability of class c.
- $\mathbb{P}(X=x \mid y=c)$ is the likelihood of observing feature vector x given class c.

The log-likelihood function $\ell(\theta_{0,1},\ldots,\theta_{0,d},\theta_{1,1},\ldots,\theta_{1,d})$ for the entire dataset can be written as:

$$\ell(\theta_{0,1},\ldots,\theta_{0,d},\theta_{1,1},\ldots,\theta_{1,d}) = \sum_{i=1}^{m}\log\left(\mathbb{P}(X^{(i)}=x^{(i)}, y^{(i)}=c)\right)$$

where m is the number of training examples, $X^{(i)}$ is the feature vector of the i-th example, and $y^{(i)}$ is its class label.

To handle the constraint $\sum_{k=1}^{d}\theta_{c,k} = 1$, I can use Lagrange multipliers λ_c for each class c.

The Lagrangian function is:

$$\mathcal{L}(\theta_{0,1},\ldots,\theta_{0,d},\theta_{1,1},\ldots,\theta_{1,d},\lambda_0,\lambda_1)=\sum_{i=1}^{m}\sum_{k=1}^{d}x_k^{(i)}\log\theta_{y^{(i)},k}+\lambda_0\left(1-\sum_{k=1}^{d}\theta_{0,k}\right)+\lambda_1\left(1-\sum_{k=1}^{d}\theta_{1,k}\right)$$

The objective function to maximize can be obtained by setting the derivative of $\mathcal{L}$ with respect to each $\theta_{c,k}$ to 0:

$$\frac{\partial\mathcal{L}}{\partial\theta_{c,k}}=\sum_{i=1}^{m}\frac{x_k^{(i)}}{\theta_{c,k}}-\lambda_c=0$$

From the above derivative, I can solve for $\theta_{c,k}$:

$$\theta_{c,k}=\frac{\sum_{i=1}^{m}x_k^{(i)}}{\lambda_c}$$

Using the constraint $\sum_{k=1}^{d}\theta_{c,k}=1$:

$$\lambda_c=\sum_{k=1}^{d}\sum_{i=1}^{m}x_k^{(i)}$$

Therefore,

$$\boxed{\theta_{c,k}=\frac{\sum_{i=1}^{m}x_k^{(i)}}{\sum_{k=1}^{d}\sum_{i=1}^{m}x_k^{(i)}}}$$

This gives us the maximum likelihood estimate $\theta_{c,k}$ for each class c and each word k in the vocabulary.

This expression makes sense and represents the maximum likelihood estimate of the probability that word V_k appears in messages of class c, where c could be spam (c=0) or non-spam (c=1). It measures how frequently each word occurs relative to all words observed in the training data for that class. This estimate also ensures that probabilities are normalized across all words in the vocabulary for each class.

I wrote some code in Python that calculates the estimates using the boxed expression above and the output tables are shown below. The code can be found in the zip file attached to this report.

Table 3: Spam Word Frequencies

	Word	Sum of Spam Vectors	Total Num of Spam Words	Theta (spam)
1	secret	3	11	0.273
2	offer	2	11	0.182
3	low	0	11	0.000
4	price	0	11	0.000

	Word	Sum of Spam Vectors	Total Num of Spam Words	Theta (spam)
5	valued	0	11	0.000
6	customer	0	11	0.000
7	today	2	11	0.182
8	dollar	1	11	0.091
9	million	1	11	0.091
10	sports	0	11	0.000
11	is	1	11	0.091
12	for	1	11	0.091
13	play	0	11	0.000
14	healthy	0	11	0.000
15	pizza	0	11	0.000

Table 4: Non-spam Word Frequencies

	Word	Sum of Non-Spam Vectors	Total Num of Non-Spam Words	Theta (non-spam)
1	secret	1	17	0.059
2	offer	0	17	0.000
3	low	2	17	0.118
4	price	2	17	0.118
5	valued	1	17	0.059
6	customer	1	17	0.059
7	today	3	17	0.176
8	dollar	0	17	0.000
9	million	0	17	0.000
10	sports	2	17	0.118
11	is	1	17	0.059
12	for	1	17	0.059
13	play	1	17	0.059
14	healthy	1	17	0.059
15	pizza	1	17	0.059

Using the tables above, the maximum likelihood estimates are:

$$\theta_{0,1} = \frac{3}{11}, \quad \theta_{0,7} = \frac{2}{11}$$
$$\theta_{1,1} = \frac{1}{17}, \quad \theta_{1,15} = \frac{1}{17}$$

where in $\theta_{c,k}$, $c = 0$ is spam and $c = 1$ is non-spam, and $k = 1$ is the word "secret", $k = 7$ is the word "today" and $k = 15$ is the word "pizza".

3.3

Given a test message **today is secret** and by using the Naive Bayes classier that I trained in Part (1)-(2), I can calculate the posterior and decide whether it is spam or not spam. All derivations are shown below.

To classify the test message using the Naive Bayes classifier trained previously, I first need to convert the message into a feature vector based on the vocabulary. I used my Python code from before to get the following feature vector and θ values.

```
Test message: today is secret
Feature vector: [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
```

```
'today':
Theta_0,7 = 2/11
Theta_1,7 = 3/17

'is':
Theta_0,11 = 1/11
Theta_1,11 = 1/17

'secret':
Theta_0,1 = 3/11
Theta_1,1 = 1/17
```

From the previous calculations, we know the class priors:

$$\mathbb{P}(y=0) = \frac{3}{7}, \quad \mathbb{P}(y=1) = \frac{4}{7}$$

Using the estimated $\theta_{c,k}$ values from the previous calculations and my python code from before, I can calculate the likelihood of the feature vector given each class.

For Spam ($y=0$):

$$\theta_{0,7} = \frac{2}{11}, \quad \theta_{0,11} = \frac{1}{11}, \quad \theta_{0,1} = \frac{3}{11}$$

For Non-Spam ($y=1$):

$$\theta_{1,7} = \frac{3}{17}, \quad \theta_{1,11} = \frac{1}{17}, \quad \theta_{1,1} = \frac{1}{17}$$

Using the multinomial distribution for the likelihood from before, I can simply take the product of the likelihoods for each class:

$$\mathbb{P}(x|y=0) = \frac{2}{11} \cdot \frac{1}{11} \cdot \frac{3}{11} = \frac{6}{1331}$$

$$\mathbb{P}(x|y=1) = \frac{3}{17} \cdot \frac{1}{17} \cdot \frac{1}{17} = \frac{3}{4913}$$

Using Bayes' theorem, the posterior probability is:

$$\mathbb{P}(y = c|x) = \frac{\mathbb{P}(x|y = c) \cdot \mathbb{P}(y = c)}{\mathbb{P}(x|y = 0) \cdot \mathbb{P}(y = 0) + \mathbb{P}(x|y = 1) \cdot \mathbb{P}(y = 1)}$$

To calculate the posterior probability of the message being spam, I can evaluate the expression below where x is the test message and $c = 0$:

$$\mathbb{P}(y = 0|x) = \frac{\mathbb{P}(x|y = 0) \cdot \mathbb{P}(y = 0)}{\mathbb{P}(x|y = 0) \cdot \mathbb{P}(y = 0) + \mathbb{P}(x|y = 1) \cdot \mathbb{P}(y = 1)}$$

$$\mathbb{P}(y = 0|x) = \frac{\frac{6}{1331} \cdot \frac{3}{7}}{\frac{6}{1331} \cdot \frac{3}{7} + \frac{3}{4913} \cdot \frac{4}{7}}$$

$$\mathbb{P}(y = 0|x) = \frac{14739}{17401} \approx 0.8470$$

$$\mathbb{P}(y = 0|x) \approx 0.8470 > 0.5$$

Given the posterior probabilities, the Naive Bayes classifier classifies the message "today is secret" as `spam` since it is above a chosen 0.5 threshold. The spam and non-spam probabilities sum up to 1 and so, we know that the probability of the test message being non-spam is about 0.153, or 15% which is very low compared to the 84.7% probability of the test message being spam.

4 - Comparing classifiers: Divorce classification/prediction

In lectures, we learned different classifiers; this question compares them on two datasets.

This dataset is about participants who completed the personal information form and a divorce predictors scale. The data is a modified version of the publicly available at https://archive.ics.uci.edu/dataset/539/divorce+predictors+data+set (by injecting noise so we will not get the exactly same results as on UCI website). The dataset `marriage.csv` is contained in the data folder. There are 170 participants and 54 attributes (or predictor variables) that are all real-valued. The last column of the CSV file is label y (1 means `divorce`, 0 means `no divorce`). Each column is for one feature (predictor variable), and each row is a sample (participant). A detailed explanation for each feature (predictor variable) can be found at the website link above. Our goal is to build a classifier using training data, such that given a test sample, we can classify (or essentially predict) whether its label is 0 (`no divorce`) or 1 (`divorce`).

We are going to compare the following classifiers: **Naive Bayes, Logistic Regression, and KNN** using the first 80% data for training and the remaining 20% for testing. I will import **scikit-learn** and use **train_test_split** to split the dataset.

Remark: Please note that, here, for Naive Bayes, this means that we have to estimate the variance for each individual feature from training data. When estimating the variance, if the variance is zero to close to zero (meaning that there is very little variability in the feature), we can set the variance

to be a small number, e.g., $\epsilon = 10^{-3}$. We do not want to have include zero or nearly variance in Naive Bayes. This tip holds for both Part One and Part Two of this question.

4.1

I can fit the models and report the testing accuracy for each of the three classifiers.

Table 5: Evaluation Results of Classifier Models

Model	Training Accuracy	Testing Accuracy
Naive Bayes	0.977941	0.970588
Logistic Regression	1.000000	0.941176
KNN	0.977941	0.970588

As seen from the table above, all three classification models perform very well on the test dataset. Naive Bayes and KNN both had a testing accuracy of 97.1% while logistic regression had a slightly lower testing accuracy of 94.12%. This would make sense since logistic regression has a linear decision boundary whereas Naive Bayes and KNN have non-linear decision boundaries that are more flexible for classifying points in the dataset. The data is also easily separable and so all models in this case perform very well as expected.

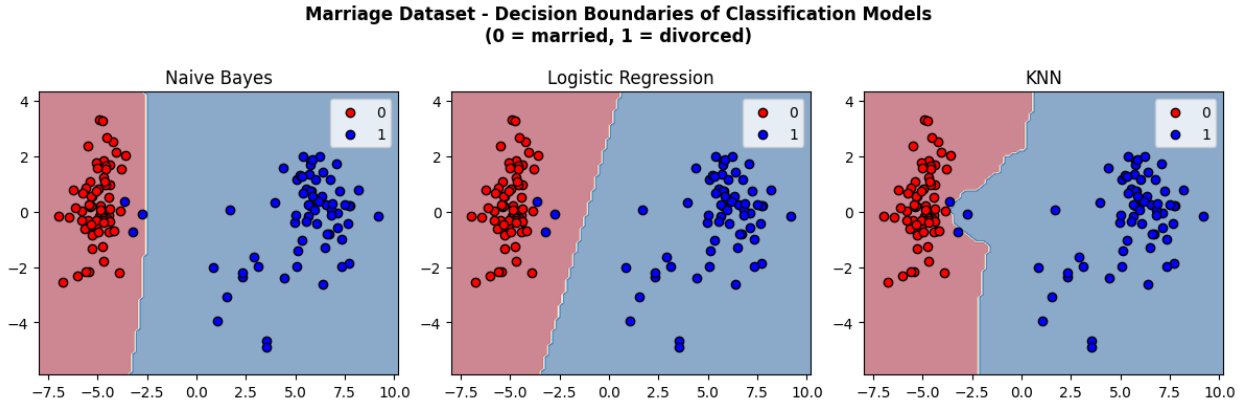
It should be noted that by changing the seed for the split function responsible for splitting the dataset into training and testing subsets, it changes the testing accuracies to times where all models have perfect accuracy or the ranking changes among them. I believe this is due to the lack of a large number of data points and the dataset being too generic that the models are able to overfit to the underlying patterns fairly easily.

4.2

I will now perform PCA to project the data into two-dimensional space. As before, I will build the classifiers **(Naive Bayes, Logistic Regression, and KNN)** using the two-dimensional PCA results. Below, I have plotted the data points and decision boundary of each classifier in the two-dimensional space.

Table 6: Evaluation Results of Classifier Models on 2D Data with PCA

Model	Training Accuracy	Testing Accuracy
Naive Bayes	0.977941	0.970588
Logistic Regression	0.977941	0.970588
KNN	0.985294	0.970588



Evident from the plots above, Naive Bayes and KNN both have non-linear decision boundaries whereas logistic regression has a linear decision boundary. The KNN decision boundary is very noisy and this may be due to the small default k value of 5 in the algorithm that decides how many neighbors a point should have to classify a label. Compared to the testing accuracies from before, logistic regression on the 2-dimensional data had an improvement in the testing accuracy to where all three classification models now performed equally with an accuracy of 97.1%.