CDA Homework 3

Joel Quek
jquek7@gatech.edu

1 Implementing EM for MNIST dataset

1.1 Question 1: Implement EM algorithm

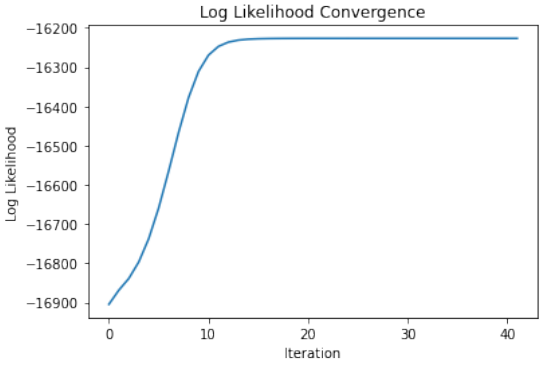

Figure 1: Log Likelihood Convergence

In reviewing the plot of log-likelihood convergence for the Expectation-Maximization (EM) algorithm that I implemented, several key observations can be made about its performance:

Initial Phase of Rapid Improvement: Initially, the log-likelihood value increases sharply within the first few iterations. This rapid increase is indicative of the EM algorithm quickly improving the fit of the Gaussian Mixture Model (GMM) to the data, starting from the random initial parameter settings.

Stabilization and Convergence: Following the steep initial increase, the log-likelihood value stabilizes and shows minimal changes as the iterations progress beyond about 30. This behavior suggests that the algorithm has reached a point of convergence where further iterations do not significantly alter the parameters, indicating that a local maximum or an optimal solution has likely been achieved.

Efficiency of the Algorithm: The plot demonstrates that my implementation of the EM algorithm is efficient, achieving convergence in a relatively

small number of iterations. This efficiency is crucial for practical applications, especially when dealing with larger datasets.

Overall, the plot confirms that the EM algorithm is functioning correctly under my implementation, optimizing the log-likelihood function effectively, and thus providing a strong fit of the GMM to the dataset. If needed, further refinements could be explored by adjusting the initialization settings or by enhancing the model complexity, but the current settings appear to be performing robustly.

1.2 Question 2: Report on the Fitted Gaussian Mixture Model (GMM)

After successfully implementing and running the Expectation-Maximization (EM) algorithm on the MNIST dataset filtered for digits "2" and "6", the model parameters were estimated and the results visualized. Here are the key outcomes:

1.2.1 Numerical Weights for Each Component

The Gaussian Mixture Model converged with the following mixture weights for its components:

- **Component 1 Weight:** 0.51010047
- **Component 2 Weight:** 0.48989953

These weights indicate the probability of each component within the model, suggesting a nearly balanced contribution from each Gaussian component to the model.

1.2.2 Mean Images of Each Component

The mean vectors of each component were projected back to the original 28x28 pixel space, providing a visual representation of what each component statistically "represents" within the data:

- **Mean of Component 1:** This image similarly shows features characteristic of the digit "6", serving as a statistical summary or an "average" appearance of this digit in the dataset.
- **Mean of Component 2:** The image shows a blurry but recognizable digit, which appears to be a composite of the digit "2". It captures common features most frequently present in the instances of this digit within the dataset.

These mean images help visualize what each component of the mixture model is capturing, essentially distilling the central tendencies of data points associated with each digit.

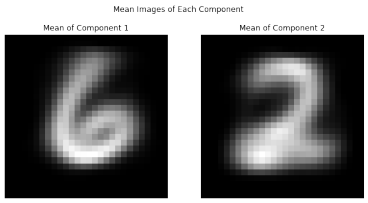

Figure 2: Mean Images of Each Component

1.2.3 Covariance Matrices

The covariance matrices for the 4-dimensional PCA-reduced data of each component were visualized as heatmaps:

- **Covariance Matrix of Component 1:** Displays significant variance along the primary dimensions, with notable off-diagonal entries indicating correlations between some of the dimensions.
- **Covariance Matrix of Component 2:** Similar to Component 1 but with its variance and correlations distributed differently, reflecting the unique ways in which the data points associated with the digit "6" spread around the mean.

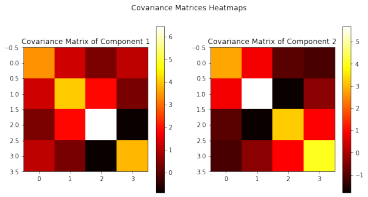

Figure 3: Covariance Matrices Heatmaps

These heatmaps are crucial for understanding the spread and dependency of the data around the mean vectors in the reduced-dimensional space. They illustrate not only how data is clustered but also how it varies and co-varies across different principal components.

1.3 Question 3: Comparison of Misclassification Rates

In this question, we use the responsibilities (τ_{ik}) from the Gaussian Mixture Model (GMM) to infer the labels of the images and compare them with the true labels. We also perform K-means clustering with $K = 2$ and compare the misclassification rates between GMM and K-means.

1.3.1 Gaussian Mixture Model (GMM)

Using the responsibilities from the GMM, the inferred labels for the digits "2" and "6" were compared with the true labels. The misclassification rate for the GMM was calculated as follows:

- **Misclassification Rate:** 0.0362

This low misclassification rate indicates that the GMM provides a good fit to the data and effectively distinguishes between the digits "2" and "6".

1.3.2 K-means Clustering

We also applied K-means clustering to the same dataset with $K = 2$. The inferred labels were compared with the true labels, and the misclassification rate for K-means was calculated as follows:

- **Misclassification Rate:** 0.0698

The higher misclassification rate for K-means clustering suggests that it is less effective at distinguishing between the digits "2" and "6" compared to the GMM.

1.3.3 Comparison of GMM and K-means

Based on the misclassification rates, we conclude that the GMM achieves better performance than K-means clustering for this dataset.

- **GMM Misclassification Rate:** 0.0362
- **K-means Misclassification Rate:** 0.0698

Therefore, the GMM provides a more accurate classification of the digits "2" and "6" compared to K-means clustering.

2 Optimization

2.1 Question 1: Derive the Gradient of the Cost Function

Given the log-likelihood function:

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

Step 1: Differentiation of the Linear Term

The linear term with respect to θ is differentiated as follows:

$$\frac{\partial}{\partial \theta} \left((y^i - 1)\theta^T x^i \right) = (y^i - 1)x^i$$

since the derivative of $\theta^T x^i$ with respect to θ is x^i.

Step 2: Differentiation of the Logistic Function Term

The logistic function term $\log(1 + \exp(-\theta^T x^i))$, where log is the natural logarithm, requires applying the chain rule:

$$\begin{aligned}\frac{\partial}{\partial\theta}\log(1+\exp(-\theta^T x^i)) &= \frac{1}{1+\exp(-\theta^T x^i)} \cdot \exp(-\theta^T x^i) \cdot \frac{\partial}{\partial\theta}(-\theta^T x^i) \\ &= \frac{\exp(-\theta^T x^i)}{1+\exp(-\theta^T x^i)}(-x^i) \\ &= \frac{1}{1+\exp(\theta^T x^i)}(-x^i) \\ &= \frac{-x^i}{1+\exp(\theta^T x^i)}\end{aligned}$$

Step 3: Summing the Derivatives

Combining both parts, the gradient of $l(\theta)$ with respect to θ is:

$$\frac{\partial}{\partial\theta}l(\theta) = \sum_{i=1}^{m}\left[(y^i - 1)x^i + \frac{x^i}{1+\exp(\theta^T x^i)}\right]$$

which is equivalent to saying

$$\frac{\partial l(\theta)}{\partial\theta} = \sum_{i=1}^{m}\left[(y^i - 1)x^i + \frac{\exp(-\theta^T x^i)x^i}{1+\exp(-\theta^T x^i)}\right]$$

This results in the final gradient expression used for updating the parameters θ in the gradient ascent optimization method.

2.2 Question 2: Pseudocode for Gradient Descent for Logistic Regression

Algorithm: Gradient Descent for Logistic Regression
Input: Training set $\{(x^1, y^1), \ldots, (x^m, y^m)\}$, learning rate η, maximum iterations T, tolerance ϵ
Output: Optimized parameters θ

1. Initialize θ to zeros or small random values.
2. Set iteration counter $t = 1$.
3. While $\|\theta^{t+1} - \theta^t\| > \epsilon$ and $t < T$:
   (a) Compute the gradient: $V = \sum_{i=1}^{m}(y^i - \sigma(\theta^T x^i))x^i$
   (b) Update parameters: $\theta^{t+1} = \theta^t - \eta V$
   (c) Increment iteration count: $t = t + 1$
4. Return optimized parameters θ

Here, $\sigma(\theta^T x_i) = \frac{1}{1+e^{-\theta^T x_i}}$ is the sigmoid function.

2.3 Question 3: Pseudocode for Stochastic Gradient Descent for Logistic Regression

Algorithm: Stochastic Gradient Descent for Logistic Regression
Input: Training set $\{(x^1, y^1), \ldots, (x^m, y^m)\}$, initial learning rate η_0, decay factor δ, maximum iterations T, tolerance ϵ
Output: Optimized parameters θ

1. Initialize θ to zeros or small random values.
2. Set iteration counter $t = 1$.
3. While $t < T$ and $\|\nabla L(\theta)\| > \epsilon$:
   (a) Shuffle the training set.
   (b) Select a random subset S_t of the training set.
   (c) For each (x_i, y_i) in S_t:
      i. Compute the gradient at the i-th example: $V_i = (y_i - \sigma(\theta^T x_i))x_i$
      ii. Update parameters: $\theta = \theta + \eta_t V_i$
   (d) Update learning rate: $\eta_t = \frac{\eta_0}{1+\delta t}$
   (e) Increment iteration count: $t = t + 1$
4. Return optimized parameters θ

Here, $\sigma(\theta^T x_i) = \frac{1}{1+e^{-\theta^T x_i}}$ is the sigmoid function.

2.3.1 Difference Between Gradient Descent and Stochastic Gradient Descent for Training Logistic Regression

Gradient Descent (GD): Computes the gradient of the log-likelihood function using the entire dataset to perform a single update per iteration. This method ensures a stable and accurate convergence but can be computationally expensive and slow on large datasets due to the need to process the entire dataset at each step.

- **Full Dataset vs. Random Samples:** GD uses the entire dataset to calculate the gradient of the log-likelihood and update the parameters in each iteration.
- **Convergence Behavior:** GD converges smoothly if the learning rate is properly set, as it uses the true gradient direction calculated from the entire dataset. This is beneficial for obtaining precise updates in logistic regression.
- **Computation Efficiency:** GD can be computationally expensive and slow, particularly with large datasets, because it requires computations over the entire dataset to make a single update.

Stochastic Gradient Descent (SGD): Updates parameters for each training example or a small mini-batch one at a time. This method is generally faster and can converge more quickly in practice, especially on large datasets, due to more frequent updates. However, the updates are noisier compared to full gradient descent, leading to more fluctuation in the optimization path. Despite this noise, SGD can often escape local minima better and reach convergence faster due to its stochastic nature.

- **Full Dataset vs. Random Samples:** In contrast to GD, SGD updates the parameters using only one or a small subset (mini-batch) of samples. This makes SGD faster per iteration and capable of handling very large datasets more efficiently.
- **Convergence Behavior:** Due to its stochastic nature, SGD has a noisier convergence path but can help in escaping local minima, which is useful in logistic regression where the loss surface can be complex.
- **Computation Efficiency:** SGD is generally more computationally efficient per iteration since it uses significantly less data for each update, making it suitable for large-scale logistic regression problems.
- **Implementation and Performance:** In practice, SGD can achieve similar or better performance compared to GD with less computation per update. For logistic regression, this means quicker adjustments and potentially faster convergence, although it may require more iterations overall. Techniques like learning rate schedules or momentum can enhance SGD's convergence in logistic regression.

2.4 Question 4: Show the Training Problem is Concave

Consider the logistic regression model where the log-likelihood function $\ell(\theta)$ is given by:

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

The gradient of $\ell(\theta)$ is:

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

To derive the Hessian matrix, we focus on the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$ and apply the quotient rule which states:

$$\frac{d}{dz}\left(\frac{u}{v}\right) = \frac{u'v - uv'}{v^2}$$

for $u(z) = 1$ and $v(z) = 1 + e^{-z}$, we find:

$$u'(z) = 0, \quad v'(z) = -e^{-z}$$

$$\sigma'(z) = \frac{0 \cdot (1 + e^{-z}) - 1 \cdot (-e^{-z})}{(1 + e^{-z})^2} = \frac{e^{-z}}{(1 + e^{-z})^2}$$

This simplifies to:

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Applying this to the derivative of the gradient with respect to θ:

$$\frac{\partial}{\partial \theta} \left(\frac{\exp(-\theta^T x^i) x^i}{1 + \exp(-\theta^T x^i)} \right) = \frac{\partial}{\partial \theta} \left(\sigma(-\theta^T x^i) x^i \right)$$

Using the chain rule and noting x^i is constant w.r.t. θ:

$$\frac{\partial}{\partial \theta} \left(\sigma(-\theta^T x^i) x^i \right) = x^i \sigma'(-\theta^T x^i) \frac{\partial}{\partial \theta} (-\theta^T x^i) = x^i \sigma(-\theta^T x^i)(1 - \sigma(-\theta^T x^i))(-x^i)$$

From this, the Hessian matrix $H(\ell(\theta))$ of the log-likelihood function $\ell(\theta)$ is given by the sum of these second derivatives for all samples:

$$H(\ell(\theta)) = -\sum_{i=1}^{m} \sigma(-\theta^T x^i)(1 - \sigma(-\theta^T x^i)) x^i (x^i)^T$$

The Hessian matrix $H(\ell(\theta))$ is the sum of matrices of the form:

$$-\sigma(-\theta^T x^i)(1 - \sigma(-\theta^T x^i)) x^i (x^i)^T$$

Since $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function, $\sigma(z)$ is always between 0 and 1. Thus, $\sigma(-\theta^T x^i)(1 - \sigma(-\theta^T x^i))$ is always non-negative.

Each term $x^i (x^i)^T$ is a positive semi-definite matrix (rank-one matrix). The negative sign in front of the summation indicates that $H(\ell(\theta))$ is a sum of negative semi-definite matrices, making $H(\ell(\theta))$ negative semi-definite.

A function with a negative semi-definite Hessian matrix is concave. Therefore, the log-likelihood function $\ell(\theta)$ is concave.

The gradient descent update rule for logistic regression is:

$$\theta^{t+1} = \theta^t + \eta \nabla \ell(\theta^t)$$

where $\nabla \ell(\theta)$ is the gradient of the log-likelihood function.

Since $\ell(\theta)$ is concave, the optimization problem is well-behaved, and the gradient descent method will efficiently converge to the unique global optimizer. Proper selection of the learning rate η ensures convergence and prevents overshooting the maximum.

In conclusion, the concavity of $\ell(\theta)$ guarantees that the logistic regression training problem can be solved efficiently using gradient descent, and the solution will be the unique global optimizer.

3 Bayes Classifier for Spam Filtering

3.1 Question 1: Calculate Class Priors

Given the training data, the class priors are calculated as follows:

$$P(y=0) = \frac{\text{Number of Spam Messages}}{\text{Total Number of Messages}} = \frac{3}{7} \approx 0.4286$$

$$P(y=1) = \frac{\text{Number of Non-Spam Messages}}{\text{Total Number of Messages}} = \frac{4}{7} \approx 0.5714$$

3.1.1 Spam Feature Vectors:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

1. "million dollar offer for today"

$$\mathbf{x}^{(1)} = [0,1,0,0,0,0,1,1,1,0,0,1,0,0,0]$$

2. "secret offer today"

$$\mathbf{x}^{(2)} = [1,1,0,0,0,0,1,0,0,0,0,0,0,0,0]$$

3. "secret is secret"

$$\mathbf{x}^{(3)} = [2,0,0,0,0,0,0,0,0,0,1,0,0,0,0]$$

3.1.2 Non-Spam Feature Vectors:

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

1. "low price for valued customer today"

$$\mathbf{x}^{(4)} = [0,0,1,1,1,1,1,0,0,0,0,1,0,0,0]$$

2. "play secret sports today"

$$\mathbf{x}^{(5)} = [1,0,0,0,0,0,1,0,0,1,0,0,1,0,0]$$

3. "sports is healthy"

$$\mathbf{x}^{(6)} = [0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0]$$

4. "low price pizza today"

$$\mathbf{x}^{(7)} = [0,0,1,1,0,0,1,0,0,0,0,0,0,0,1]$$

3.2 Question 2: Derivation of Maximum Likelihood Estimates

3.2.1 Formulate the Lagrangian

The log-likelihood function for our training data is:

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

Since $\sum_{k=1}^{d} \theta_{c,k} = 1$, we introduce Lagrange multipliers λ_0 and λ_1:

$$\mathcal{L}(\theta_{0,1}, \ldots, \theta_{0,d}, \lambda_0) = \sum_{i=1}^{m_0} \sum_{k=1}^{d} x_k^{(i)} \log \theta_{0,k} + \lambda_0 \left(1 - \sum_{k=1}^{d} \theta_{0,k}\right)$$

$$\mathcal{L}(\theta_{1,1}, \ldots, \theta_{1,d}, \lambda_1) = \sum_{i=1}^{m_1} \sum_{k=1}^{d} x_k^{(i)} \log \theta_{1,k} + \lambda_1 \left(1 - \sum_{k=1}^{d} \theta_{1,k}\right)$$

Take Partial Derivatives and Set to Zero

For spam messages ($y = 0$):

$$\frac{\partial \mathcal{L}}{\partial \theta_{0,k}} = \sum_{i=1}^{m_0} x_k^{(i)} \frac{1}{\theta_{0,k}} - \lambda_0 = 0$$

Rearranging,

$$\theta_{0,k} = \frac{\sum_{i=1}^{m_0} x_k^{(i)}}{\lambda_0}$$

Summing over all k,

$$\sum_{k=1}^{d} \theta_{0,k} = 1 = \frac{\sum_{k=1}^{d} \sum_{i=1}^{m_0} x_k^{(i)}}{\lambda_0}$$

Let $N_0 = \sum_{i=1}^{m_0} n_i$ be the total number of words in all spam messages, then,

$$\lambda_0 = N_0$$

Thus,

$$\theta_{0,k} = \frac{\sum_{i=1}^{m_0} x_k^{(i)}}{N_0}$$

For non-spam messages ($y = 1$):

$$\frac{\partial \mathcal{L}}{\partial \theta_{1,k}} = \sum_{i=1}^{m_1} x_k^{(i)} \frac{1}{\theta_{1,k}} - \lambda_1 = 0$$

Rearranging,

$$\theta_{1,k} = \frac{\sum_{i=1}^{m_1} x_k^{(i)}}{\lambda_1}$$

Summing over all k,

$$\sum_{k=1}^{d} \theta_{1,k} = 1 = \frac{\sum_{k=1}^{d} \sum_{i=1}^{m_1} x_k^{(i)}}{\lambda_1}$$

Let $N_1 = \sum_{i=1}^{m_1} n_i$ be the total number of words in all non-spam messages, then,

$$\lambda_1 = N_1$$

Thus,

$$\theta_{1,k} = \frac{\sum_{i=1}^{m_1} x_k^{(i)}}{N_1}$$

3.2.2 Theta Values for Spam and Non-Spam Classes

The calculated θ values for both the Spam class ($\theta_{0,k}$) and the Non-Spam class ($\theta_{1,k}$) provide the probabilities of each word appearing in messages of each class. These values are essential for determining the likelihood of a message being spam or non-spam based on its content.

Spam Class ($\theta_{0,k}$) Total words in spam messages = 5 + 3 + 3 = 11

$$\theta_{0,k} = \frac{\text{count of word } k \text{ in spam}}{11}$$

The θ values for the Spam class ($\theta_{0,k}$) are:

$$\theta_{0,k} = [0.2727, 0.1818, 0, 0, 0, 0, 0.1818, 0.0909, 0.0909, 0, 0.0909, 0.0909, 0, 0, 0]$$

This translates to the following probabilities for each word in the vocabulary:

- "secret": $3 \div 11 = 0.2727$
- "offer": $2 \div 11 = 0.1818$
- "low": 0
- "price": 0
- "valued": 0
- "customer": 0
- "today": $2 \div 11 = 0.1818$

- "dollar": $1 \div 11 = 0.0909$
- "million": $1 \div 11 = 0.0909$
- "sports": 0
- "is": $1 \div 11 = 0.0909$
- "for": $1 \div 11 = 0.0909$
- "play": 0
- "healthy": 0
- "pizza": 0

Words like "secret" and "offer" have higher probabilities in spam messages, indicating that these words are strong indicators of spam.

Non-Spam Class ($\theta_{1,k}$) Total words in non-spam messages = 6 + 4 + 3 + 4 = 17

$$\theta_{1,k} = \frac{\text{count of word } k \text{ in spam}}{17}$$

The θ values for the Non-Spam class ($\theta_{1,k}$) are:

$$\theta_{1,k} = [0.0588, 0, 0.1176, 0.1176, 0.0588, 0.0588, 0.1765, 0, 0, 0.1176, 0.0588, 0.0588, 0.0588, 0.0588, 0.0588]$$

This translates to the following probabilities for each word in the vocabulary:

- "secret": $1 \div 17 = 0.0588$
- "offer": 0
- "low": $2 \div 17 = 0.1176$
- "price": $2 \div 17 = 0.1176$
- "valued": $1 \div 17 = 0.0588$
- "customer": $1 \div 17 = 0.0588$
- "today": $3 \div 17 = 0.1765$
- "dollar": 0
- "million": 0
- "sports": $2 \div 17 = 0.1176$
- "is": $1 \div 17 = 0.0588$
- "for": $1 \div 17 = 0.0588$

- "play": $1 \div 17 = 0.0588$
- "healthy": $1 \div 17 = 0.0588$
- "pizza": $1 \div 17 = 0.0588$

Words like "today" and "low" have higher probabilities in non-spam messages, indicating these words are more commonly found in legitimate emails.

3.2.3 For Spam Messages ($y = 0$)

Total words in spam messages (N_0):

$$N_0 = 5 + 3 + 3 = 11$$

$$\theta_{0,1} = \frac{\sum_{i=1}^{3} x_1^{(i)}}{N_0} = \frac{3}{11}$$

$$\theta_{0,7} = \frac{\sum_{i=1}^{3} x_7^{(i)}}{N_0} = \frac{2}{11}$$

3.2.4 For Non-Spam Messages ($y = 1$)

Total words in non-spam messages (N_1):

$$N_1 = 6 + 4 + 3 + 4 = 17$$

$$\theta_{1,1} = \frac{\sum_{i=1}^{4} x_1^{(i)}}{N_1} = \frac{1}{17}$$

$$\theta_{1,15} = \frac{\sum_{i=1}^{4} x_{15}^{(i)}}{N_1} = \frac{1}{17}$$

Summary

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

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

$$\theta_{1,1} = \frac{1}{17}$$

$$\theta_{1,15} = \frac{1}{17}$$

These values are the maximum likelihood estimates for $\theta_{0,1}$, $\theta_{0,7}$, $\theta_{1,1}$, and $\theta_{1,15}$.

3.2.5 Conclusion

The θ values clearly show the difference in word distributions between spam and non-spam messages. Words such as "secret", "offer", "dollar", and "million" are strong indicators of spam, while words like "low", "price", and "today" are more indicative of non-spam.

These θ values can be used to calculate the likelihood of a new message being spam or non-spam. For example, to classify a new message "today is secret", you would:

1. Calculate the likelihood of the message being spam and non-spam using the θ values.
2. Compare the likelihoods to determine which class the message most likely belongs to.

3.3 Question 3: Classifying "today is secret" using Naive Bayes Classifier

Given the test message "today is secret", we will calculate the posterior probabilities and decide whether it is spam or not spam.

Step 1: Convert the message to a feature vector
The feature vector for "today is secret" is:

$$\text{"today is secret"} \rightarrow [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]$$

Step 2: Calculate the likelihood of the message being spam
The θ values for the Spam class ($\theta_{0,k}$) are:

$$\theta_{0,k} = [0.2727, 0.1818, 0, 0, 0, 0, 0.1818, 0.0909, 0.0909, 0, 0.0909, 0.0909, 0, 0, 0]$$

Thus, the likelihood of the message being spam is:

$$P(\text{"today is secret"}|\text{spam}) = 0.2727 \times 0.1818 \times 0.0909 = 0.004497$$

Step 3: Calculate the likelihood of the message being non-spam
The θ values for the Non-Spam class ($\theta_{1,k}$) are:

$$\theta_{1,k} = [0.0588, 0, 0.1176, 0.1176, 0.0588, 0.0588, 0.1765, 0, 0, 0.1176, 0.0588, 0.0588, 0.0588, 0.0588, 0.0588]$$

Thus, the likelihood of the message being non-spam is:

$$P(\text{"today is secret"}|\text{non-spam}) = 0.1765 \times 0.0588 \times 0.0588 = 0.000609$$

Step 4: Calculate the posterior probabilities using Bayes' theorem and class priors

Using Bayes' theorem:

$$P(y = c|z) = \frac{P(z|y = c)P(y = c)}{P(z|y = 0)P(y = 0) + P(z|y = 1)P(y = 1)}$$

Calculate Spam Posterior $P(y = 0|\text{"today is secret"})$:

$$P(y = 0|\text{"today is secret"}) = \frac{P(\text{"today is secret"}|y = 0)P(y = 0)}{P(\text{"today is secret"}|y = 0)P(y = 0) + P(\text{"today is secret"}|y = 1)P(y = 1)}$$

Substituting the values:

$$P(y = 0|\text{"today is secret"}) = \frac{0.004497 \times \frac{3}{7}}{0.004497 \times \frac{3}{7} + 0.000609 \times \frac{4}{7}}$$

$$P(y = 0|\text{"today is secret"}) = \frac{0.004497 \times 0.4286}{0.004497 \times 0.4286 + 0.000609 \times 0.5714}$$

$$P(y = 0|\text{"today is secret"}) = \frac{0.001927}{0.001927 + 0.000348} = \frac{0.001927}{0.002275} \approx 0.847$$

Calculate Non-Spam Posterior $P(y = 1|\text{"today is secret"})$:

$$P(y = 1|\text{"today is secret"}) = \frac{P(\text{"today is secret"}|y = 1)P(y = 1)}{P(\text{"today is secret"}|y = 0)P(y = 0) + P(\text{"today is secret"}|y = 1)P(y = 1)}$$

Substituting the values:

$$P(y = 1|\text{"today is secret"}) = \frac{0.000609 \times \frac{4}{7}}{0.004497 \times \frac{3}{7} + 0.000609 \times \frac{4}{7}}$$

$$P(y = 1|\text{"today is secret"}) = \frac{0.000609 \times 0.5714}{0.004497 \times 0.4286 + 0.000609 \times 0.5714}$$

$$P(y = 1|\text{"today is secret"}) = \frac{0.000348}{0.001927 + 0.000348} = \frac{0.000348}{0.002275} \approx 0.153$$

Step 5: Apply Bayes Decision Rule

$$\text{If } q_1(z) > q_0(z), \text{ then } y = 1 \text{ (non-spam)}$$
$$\text{otherwise } y = 0 \text{ (spam)}$$

Where:

$$q_0(z) = P(y = 0|\text{"today is secret"}) \approx 0.847$$
$$q_1(z) = P(y = 1|\text{"today is secret"}) \approx 0.153$$

Since $q_0(z) > q_1(z)$, we classify the message "today is secret" as **spam**.

4 Comparing Classifiers: Divorce Classification/Prediction

4.1 Question 1: Report Testing Accuracy

Performance Analysis:

- **KNN (K-Nearest Neighbors) and Gaussian Naive Bayes** performed best with a testing accuracy of approximately 0.970588. This can be attributed to several factors:
  - **KNN:** This model's success might be due to its ability to handle complex patterns in the data without assuming any specific form of the data distribution. Given the diverse scales and potential non-linear relationships (suggested by different ranges and spread in data), KNN can adapt well by effectively capturing local similarities among samples.
  - **Gaussian Naive Bayes:** Despite the assumption of a normal distribution for each feature, Gaussian Naive Bayes might perform well here due to its robustness in cases where the actual distributions of features do not deviate drastically from normality. The effectiveness might also be boosted by its capacity to handle features independently, mitigating the effect of multicollinearity among features.
- **Logistic Regression:** Although it has a perfect training accuracy, its testing accuracy drops to 0.941176. This could be indicative of overfitting, where the model learns the noise and specific details in the training data at the expense of losing generalizability. Logistic Regression assumes linear relationships and feature independence, both of which might be compromised in this dataset due to the presence of highly correlated features and potentially non-linear relationships.

Conclusion: KNN and Gaussian Naive Bayes are likely performing better in this setting because their inherent model characteristics align better with the underlying data structure and distribution in the `marriage.csv` dataset. Their ability to model complex and non-linear relationships without stringent assumptions about data distribution makes them more suited for this dataset as compared to Logistic Regression.

Classifier	Training Accuracy	Testing Accuracy
Naive Bayes	0.977941	0.970588
Logistic Regression	1.000000	0.941176
K-Nearest Neighbors	0.977941	0.970588

Table 1: Accuracy of classifiers before PCA

Classifier	Training Accuracy after PCA	Testing Accuracy after PCA
Naive Bayes	0.977941	0.970588
Logistic Regression	0.977941	0.970588
K-Nearest Neighbors	0.985294	0.970588

Table 2: Accuracy of classifiers after PCA

4.2 Question 2: PCA and Decision Boundaries

To visualize the decision boundaries of the classifiers, we performed PCA to reduce the feature space to two dimensions. The decision boundaries for Naive Bayes, Logistic Regression, and K-Nearest Neighbors are shown in the figure below.

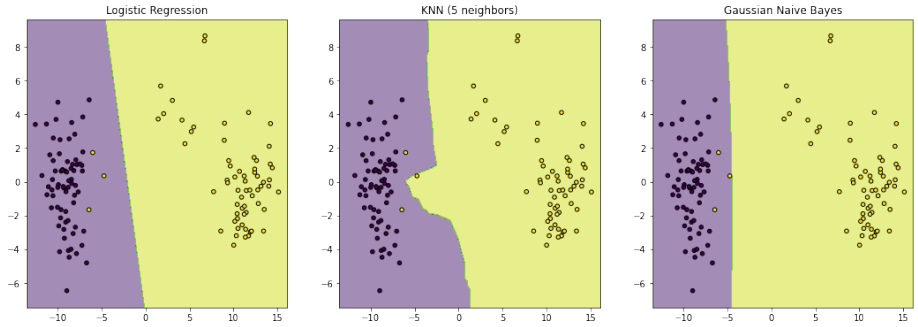

Figure 4: Decision boundaries of classifiers after PCA

4.2.1 Visual Analysis of Decision Boundaries

- **Logistic Regression:**
  - **Boundary Characteristics:** The decision boundary is a straight line, indicating a linear model approach.
  - **Classification Zones:** Divides the space into two clear zones, potentially oversimplifying the data structure.
- **KNN (5 neighbors):**
  - **Boundary Characteristics:** Exhibits an irregular and non-linear boundary, adapting closely to the local distribution of data points.
  - **Classification Zones:** Forms detailed pockets of classification, showing high sensitivity to local data density.
- **Gaussian Naive Bayes:**

- **Boundary Characteristics:** Mostly linear but with some curvature, reflecting the assumption of normal distribution within each class.
- **Classification Zones:** More generalized than KNN, fitting broader areas rather than specific data clusters.

4.2.2 Differences in Decision Boundaries

1. **Adaptability and Complexity:**
   - Logistic Regression shows the least adaptability with a purely linear decision boundary.
   - KNN demonstrates high adaptability with boundaries that contour to local data arrangements, ideal for datasets with complex patterns.
   - Gaussian Naive Bayes provides a balance, offering some adaptability without extreme sensitivity to data noise.
2. **Generalization Capability:**
   - **Logistic Regression** might underfit complex data structures due to its rigid boundary.
   - **KNN** might overfit, interpreting noise as patterns, especially in datasets with intricate structures.
   - **Gaussian Naive Bayes** is less likely to overfit compared to KNN and may perform well if the data distribution approximates normality.

4.2.3 Conclusion

The decision boundaries illustrated in the PCA-reduced two-dimensional space highlight the critical importance of model selection based on underlying data characteristics and specific needs of the problem, aiming to balance bias and variance for optimal generalization.