CS771 Minor Assignment 2

MLR-48 : Gradient Gang

1 Question-1

To train a generative multi-classification model where each class-conditional probability $\mathbb{P}[x|y = c]$ is a Gaussian Mixture Model (GMM), we need to estimate the parameters for each class's GMM and the class priors. The training is performed independently for each class.

Let's consider a single class c. We are given the data points belonging to this class, $D_c = \{x_1, x_2, \ldots, x_{N_c}\}$. The class-conditional distribution is:

$$\mathbb{P}[x|y = c] = \sum_{k=1}^{K} \pi_k^c \mathcal{N}(x|\mu_k^c, \Sigma_k^c)$$

where the parameters to be estimated for class c are $\theta_c = \{\pi_k^c, \mu_k^c, \Sigma_k^c\}_{k=1}^K$.

Maximizing the log-likelihood directly is intractable due to the sum inside the logarithm. Therefore, we use the Expectation-Maximization (EM) algorithm.

1.1 Estimating Class Priors

The class prior $\mathbb{P}[y = c]$ is estimated using Maximum Likelihood Estimation:

$$\hat{\mathbb{P}}[y = c] = \frac{N_c}{N}$$

where N_c is the number of training samples in class c, and N is the total number of training samples.

1.2 Estimating GMM Parameters with EM

For each class $c \in [C]$, we run the EM algorithm on its data D_c. The algorithm iterates between an Expectation (E) step and a Maximization (M) step.

1.2.1 Initialization

Initialize the parameters μ_k^c, Σ_k^c, and π_k^c for each component $k \in [K]$. This can be done using k-means clustering or by random selection from the data.

1.2.2 E-Step (Expectation)

In the E-step, we compute the posterior probability, or "responsibility," that component k has for generating data point x_i. This is calculated for every data point $x_i \in D_c$ and every component $k \in [K]$:

$$\gamma(z_{ik}^c) = \mathbb{P}(z_k^c = 1|x_i, \theta_c^{\text{old}}) = \frac{\pi_k^{c,\text{old}} \mathcal{N}(x_i|\mu_k^{c,\text{old}}, \Sigma_k^{c,\text{old}})}{\sum_{j=1}^{K} \pi_j^{c,\text{old}} \mathcal{N}(x_i|\mu_j^{c,\text{old}}, \Sigma_j^{c,\text{old}})}$$

Here, z_k^c is a latent variable indicating if component k of class c's GMM generated x_i.

1.2.3 M-Step (Maximization)

In the M-step, we re-estimate the model parameters using the responsibilities calculated in the E-step. We update the parameters to maximize the expected complete-data log-likelihood.

First, let's calculate the effective number of points assigned to component k:

$$N_k^c = \sum_{i=1}^{N_c} \gamma(z_{ik}^c)$$

Now, the parameter updates are:

- **Update mixture weights:** The new mixture weight for component k is the average responsibility of that component over all data points.

$$\pi_k^{c,\text{new}} = \frac{N_k^c}{N_c}$$

- **Update component means:** The new mean for component k is a weighted average of the data points, where the weights are the responsibilities.

$$\mu_k^{c,\text{new}} = \frac{1}{N_k^c} \sum_{i=1}^{N_c} \gamma(z_{ik}^c) x_i$$

- **Update component covariances:** The new covariance for component k is a weighted average of the outer products of the centered data points.

$$\Sigma_k^{c,\text{new}} = \frac{1}{N_k^c} \sum_{i=1}^{N_c} \gamma(z_{ik}^c)(x_i - \mu_k^{c,\text{new}})(x_i - \mu_k^{c,\text{new}})^T$$

1.2.4 Convergence

The E-step and M-step are repeated until the log-likelihood of the model converges, i.e., the change in log-likelihood between iterations falls below a small threshold. The log-likelihood for class c is given by:

$$\mathcal{L}(\theta_c) = \sum_{i=1}^{N_c} \log \left(\sum_{k=1}^{K} \pi_k^c \mathcal{N}(x_i | \mu_k^c, \Sigma_k^c) \right)$$

This process is repeated for every class $c \in [C]$ to train all the class-conditional GMMs.

2 Question-2

Inference, or testing, involves classifying a new, unseen test point x_{test}. For a generative classifier, we use Bayes' theorem to find the class c that maximizes the posterior probability $\mathbb{P}[y = c | x_{test}]$.

The posterior probability is given by:

$$\mathbb{P}[y = c | x_{test}] = \frac{\mathbb{P}[x_{test} | y = c]\mathbb{P}[y = c]}{\mathbb{P}[x_{test}]}$$

The denominator, $\mathbb{P}[x_{test}] = \sum_{c'=1}^{C} \mathbb{P}[x_{test} | y = c']\mathbb{P}[y = c']$, is a normalization constant that is the same for all classes. Therefore, for classification, we can ignore it and find the class that maximizes the numerator.

The prediction rule is:

$$\hat{y} = \arg\max_{c \in [C]} \mathbb{P}[x_{test} | y = c]\mathbb{P}[y = c]$$

We use the parameters learned during the training phase:

1. **Class Priors $\mathbb{P}[y = c]$:** These were estimated during training as $\hat{\mathbb{P}}[y = c] = \frac{N_c}{N}$.

2. **Class-Conditional Probabilities $\mathbb{P}[x_{test}|y=c]$:** This is the likelihood of the test point under the GMM for class c, which we trained using the EM algorithm. The value is calculated using the learned parameters $\{\pi_k^c, \mu_k^c, \Sigma_k^c\}_{k=1}^K$:

   $$\mathbb{P}[x_{test}|y=c] = \sum_{k=1}^{K} \pi_k^c \mathcal{N}(x_{test}|\mu_k^c, \Sigma_k^c)$$

   where $\mathcal{N}(x_{test}|\mu_k^c, \Sigma_k^c)$ is the probability density function of the multivariate normal distribution:

   $$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

   with D being the dimension of the data vector x.

Combining these, the final classification rule is:

$$\hat{y} = \arg\max_{c\in[C]} \left(\hat{\mathbb{P}}[y=c]\sum_{k=1}^{K} \pi_k^c \mathcal{N}(x_{test}|\mu_k^c, \Sigma_k^c)\right)$$

For numerical stability, it is often better to work with log-probabilities. The decision rule becomes:

$$\hat{y} = \arg\max_{c\in[C]} \left(\log(\hat{\mathbb{P}}[y=c]) + \log\left(\sum_{k=1}^{K} \pi_k^c \mathcal{N}(x_{test}|\mu_k^c, \Sigma_k^c)\right)\right)$$

To perform inference, we simply compute this value for each class $c = 1, \ldots, C$ and assign x_{test} to the class that yields the highest value.

3 Question-3

When we transitioned from the single Gaussian model to the Gaussian Mixture Model (GMM) for each class, a key difference in the training process using the EM algorithm is how the parameters, especially the covariance matrix, are updated.

3.1 Changes introduced

1. **Modeling each class**:
   Instead of one Gaussian per class, we fit a mixture of K Gaussians per class. Multiple mean vectors (μ_k), multiple covariance matrices (Σ_k), and multiple mixing coefficients (π_k) for each class are learned. The learning process for these parameters involves the EM algorithm, which iteratively refines the estimates of the means, covariances, and mixing coefficients to best fit the data for that class as a combination of K Gaussians. The key difference here is that we are now learning K sets of parameters per class instead of just one, allowing for more complex and multi-modal distributions within a class.

2. **Update to Covariance Matrix**:
   In the original single Gaussian case (*learnClassConditionalDist*), we simply calculated the empirical mean and covariance for all data points in a class once. The covariance matrix was a fixed value after this initial calculation.

   However, in the GMM training using the EM algorithm, the M-step involves updating the parameters of each of the K Gaussian components based on the responsibilities calculated in the E-step. The update equations for the means (μ_k) and mixing coefficients (π_k) are calculated in each M-step. Crucially, for a full GMM, the covariance matrix (Σ_k) for each component k is also updated in every iteration of the M-step. The formula for updating the covariance matrix for component k is:

   $$\Sigma_k = \frac{\sum_{i=1}^{N_c} q_{ik}(\mathbf{x}*i - \mu_k)(\mathbf{x}*i - \mu_k)^T}{\sum *i = 1^{N_c} q * ik}$$

where q_{ik} is the responsibility of component k for data point $\mathbf{x}_i$, $\mathbf{x}_i$is a data point in the current class, μ_k is the updated mean for component k in the current M-step, and N_c is the number of data points in the current class.

By updating the covariance matrix at each iteration, the GMM can more accurately capture the intricate correlations between pixels within the different substructures (components) of each digit class, leading to a more refined model and potentially better performance.

3. **Prediction**:

   For a given test data point, the function calculates the probability (log-likelihood) of that data point under the GMM learned for each class. This involves summing the probabilities from all K Gaussian components within that class's GMM, weighted by their mixing coefficients.

   The function then predicts the class whose GMM assigns the highest probability to the test data point. The core change here is that the class score is now an aggregate of scores from multiple Gaussians, reflecting the mixture model.

3.2 Results

1. **Training Accuracy**: As the number of components (K) increases from 1 to 30, the training accuracy generally increases. It starts at around 88.22% for K=1 (which is equivalent to the single Gaussian model's accuracy) and reaches its highest points around K=20, 25, and 30, exceeding 94%. This suggests that with more components, the GMM is better able to fit the training data.
2. **Test Accuracy**: The test accuracy shows a different trend. It increases as K goes from 1 up to 10, reaching a peak of 89.9% at K=10. However, as K increases beyond 10, the test accuracy starts to decrease, dropping to around 85.98% for K=30.

In conclusion, while increasing the number of GMM components (K) helps the model fit the training data better, it eventually leads to a decrease in performance on unseen test data for K values greater than 10. This indicates that the model starts to overfit the training data when K is too large. As shown in Figure 1, the best test performance was achieved with K=10, suggesting it was the optimal balance between model complexity and generalization ability on this dataset.

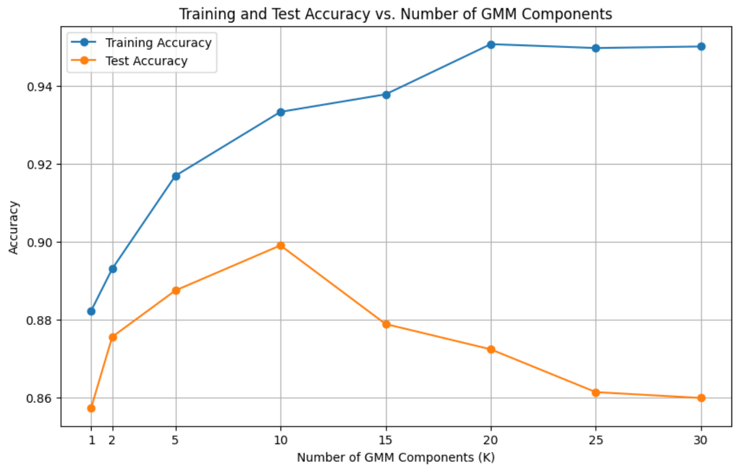

Figure 1: Training and Test Accuracy vs. Number of GMM Components

4 Question 4

4.1 Changes introduced

1. **Prediction**:

   For the GMM per class, the log-likelihood of the observed pixels under each class's GMM is calculated. This involves considering only the observed dimensions of the means and covariance matrices for each component within the GMM. The class prediction is then made based on which class's GMM (using only the observed features) gives the highest probability to the censored test image.

2. **Reconstruction**:

   The function estimates the values of the missing pixels based on the observed pixels and the learned Gaussian parameters (mean and covariance). It uses the conditional distribution of the missing features given the observed features, which for a Gaussian is another Gaussian. For a censored image, the function identifies the component within the predicted class's GMM that best explains the observed pixels and then uses that component's parameters to reconstruct the missing pixels.

4.2 Results

Similar to the uncensored results, the test accuracy on censored images generally increases as K increases from 1 up to a certain point, peaking around K=10 or K=20 (with a slight dip at K=15). The accuracy then decreases as K increases further (K=25 and 30). As expected, the prediction accuracy on censored images is lower than on the uncensored images for all values of K. This is because the model has less information to work with when a portion of the image is missing.

Figures 3 and 4 present several censored images along with their reconstructed counterparts.

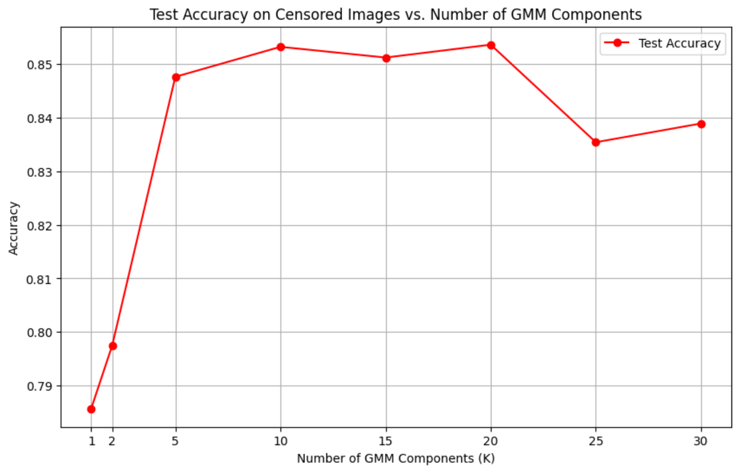

Figure 2: Test Accuracy on Censored Images vs. Number of GMM Components

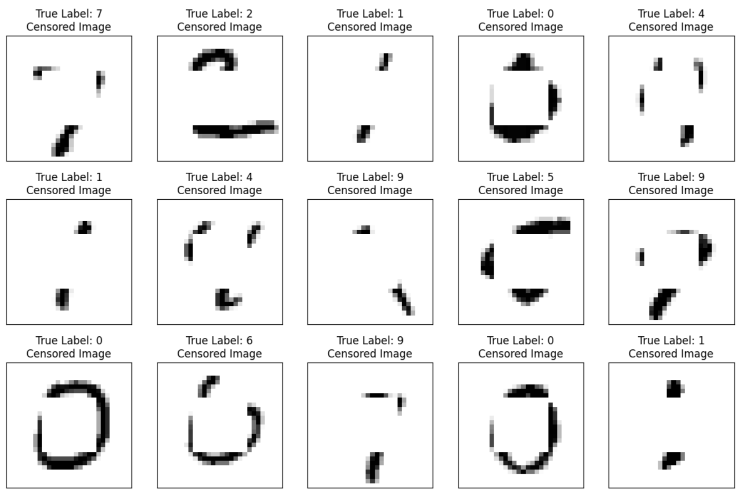

Figure 3: Sample censored images.

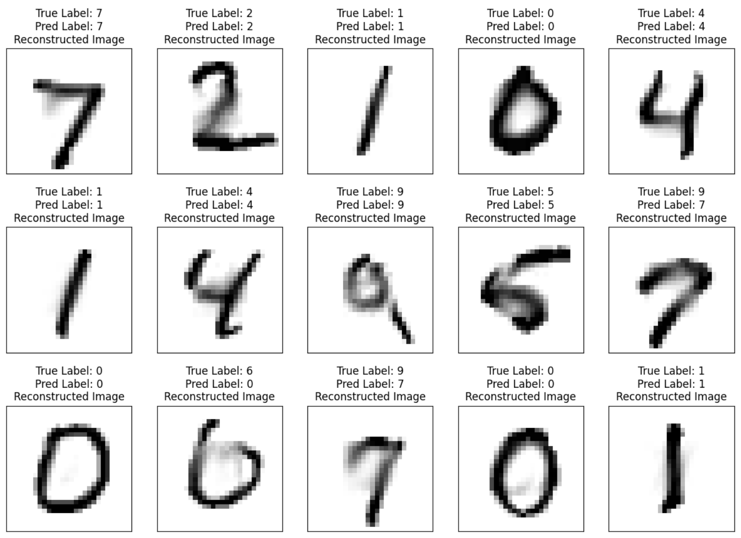

Figure 4: Reconstructed images with their predicted and true class labels