ECE1513: Introduction to Machine Learning
Programming Assignment 4
Assigned: Nov 16, 2022; Due: Dec 9, 2022 @ 11:59 p.m.

Objectives

In this assignment, you will implement the k-means and mixture of Gaussian (MoG) models, using PyTorch. The functions are provided and **you will need to fill-in after the `TODO` comment instead of coding them from scratch**. There are two parts in this assignment and we provide the dataset `data2D.npy` in the attached folder for both parts. In the first part, you will use PyTorch and gradient descent to implement the k-means clustering model in `KMEANS.py`. You will also compare the performance of your implementation to the one in scikit-learn. In the second part, we shift our focus to the MoG model and implement its learning and inference algorithms in `GMM.py`. Similar to the first part, instead of using EM algorithm, we will optimize the parameters using gradient descent. You will need to complete the training loop and explain in your report the functionality of some existing functions in `GMM.py`. Finally, for both parts, you will also be asked to answer several questions related to your implementations.

To avoid any potential installation issue, you are encouraged to develop your solution using Google Colab notebooks. You do not need to use GPU in this assignment.

Requirements

In your implementations, please use the function prototype provided (i.e. name of the function, inputs and outputs) in the detailed instructions presented in the remainder of this document. We will be testing your code using a test function that will evoke the provided function prototype. If our testing file is unable to recognize the function prototype you have implemented, you can lose significant portion of your marks. In the assignment folder, the following files are included in the `starter_code` folder:

- `KMEANS.py`
- `GMM.py`
- `data2D.npz`: this is a dataset and you can upload this file to Google Colab. We provided functions to read the data in both Python files.

These files contain the test function and an outline of the functions that you will be implementing. You also need to submit a separate `PA4_qa.pdf` file that answer questions related to your implementations (pdf version of your notebooks is also acceptable).

Abbreviations

Following the definition in our code, we use the following abbreivations in this document:

- `F` is an abbreviated import name of `torch.nn.functional`.
- `nn` is an abbreviated import name of `torch.nn`.

Preliminaries

In this part, we explain several helper functions/classes in the `KMEANS.py` and `GMM.py` file. **Do not modify any of these functions/classes**.

- Function 1: `def load_data()`. This function is available in `KMEANS.py` and `GMM.py`.
  - Inputs: None.
  - Output: `train_data, val_data`
    The outputs are training and testing data in the form of Numpy matrices. `train_data` and `val_data` are the training and testing set respectively.
  - Functionality:
    This function loads the `data2D.npy` dataset and splits it into training and test set.
- Function 2: `test_pytorch(train_data, test_data, k=5)` This function is available in `KMEANS.py` only.
  - Inputs: `test_data`, `k`
    The input `train_data`, `test_data` is our training and test data, which is a NumPy array of dimension `N x D` and `H x D` respectively. The scalar `k` is the number of clusters.
  - Outputs: This function performs k-means clustering. It uses the functions that you are about to implement in Part 1 to find the optimal cluster means (on `train_data`) and visualizes the results (on `test_data`).
- Function 3: `test_GMM(k = 5)` This function is available in `GMM.py` only.
  - Inputs: `k` The scalar `k` is the number of clusters.
  - Outputs: This function performs MoG clustering. It uses the functions that you are about to implement in Part 2 to find the optimal cluster parameters and visualizes the results.

Part 1: K-means Clustering

In this part, you will implement k-means clustering method using PyTorch. Recall that in k-means, we want to find the cluster means $\mu = \{\mu_1, \mu_2, ..., \mu_k\}$ (k is the number of clusters) that minimizes the following loss function:

$$\min_{\mu} \mathcal{L}(\mu) = \frac{1}{N} \sum_{n=1}^{N} \min_{i=1,...,k} ||x_n - \mu_i||_2^2$$

where N is the number of training data points. In Lloyd's algorithm, one can solve this problem by first initialize the cluster means $\mu = \{\mu_1, \mu_2, ..., \mu_k\}$ and then alternate between assignment step (assign each data points x_n to one of the k clusters) and update step (recalculate μ with the new assignments). Another way we can minimize $\mathcal{L}(\mu)$ is by using gradient descent. You may notice that the min operation is not differentiable, however, mathematically speaking, we can still calculate the sub-gradient for this function (actually, this is the same case for the derivative of ReLU activation at 0 or the max-pool operation). In this assignment, you will leverage the autodiff functionality in PyTorch to optimize the above function, which is similar to what you did in the Assignment 3. For this part, you need to complete the following functions:

- Function 1: `train_kmean_torch(train_data, k = 5, lr=0.1, epoch=150)`
  - Inputs: `train_data, k = 5, lr=0.1, epoch=150`
    The input `train_data` is our training data, which is a NumPy array of dimension `N x D` where `N` is the number of training points and `D` is the number of features (`D =2` in this assignment). `k`, `lr` and `epoch` are the number of clusters, learning rate and training epochs respectively.

- Output: This function returns the objective $\mathcal{L}(\mu)$ and the cluster means μ.

  The output `L_train` is a scalar that measures the average distance between each data points to its assigned cluster's mean. `m` is a NumPy array of size `k x D` (where `k` and `D` are defined above) that contains the cluster means (`k` of them).

- Function implementation considerations:

  1. This function first converts the `train_data` into a torch tensor. It then initializes the cluster means as a trainable tensor. You then need to specify the Adam Optimizer, using the learning rate `lr`.
  2. Next, you will calculate the squared distance between each data points `X[i]` and **every cluster mean** `m[j]` and put the distance into the list `list_mse`. Note that `list_mse` contains `k` tensor arrays (each has the size of `N`), where the `jth` array is the list of distance between each data point to the cluster mean `m[j]`. You can use broadcast operation https://numpy.org/doc/1.20/user/theory.broadcasting.html in NumPy/PyTorch to complete this calculation.
  3. Finally, you will calculate the objective function by averaging the distances between each data points to its nearest cluster mean. To do this, first you will use `torch.stack` to turn `list_mse` list into a tensor `list_mse_torch` of size `k x N`. Then you will use `torch.min` to get the tensor of closest distance `list_mse_torch_min`. Finally, calculate `L_train` by using `torch.mean` to get the loss function on the training set.
  4. The function will then perform back-propagation and returns the NumPy version of the training loss `L_train` and cluster means `m`.

- Function 2: `evaluate(test_data, m)`

  - Inputs: `test_data`, `m`

    The input `test_data` is our testing data, which is a NumPy array of dimension `H x D` where `H` is the number of testing points and `D` is the number of features (2 in this assignment). `m` is a NumPy array of size `k x D` that contains `k` clusters means, each of size `1 x D`.

  - Outputs: This function returns the objective $\mathcal{L}(\mu)$ w.r.t the cluster means μ for the test set.

    The output is a scalar that measures the average distance between each data points to its assigned cluster's mean.

  - Function implementation considerations: The loss function is the same (with a different set of data) as the one you implemented in `train_kmean_torch` (but using NumPy instead of PyTorch).

- Function 3: `get_association(test_data, m)`

  - Inputs: `test_data`, `m`

    The input `test_data` is our data, which is a NumPy array of dimension `H x D` where `H` is the number of data points and `D` is the number of features (2 in this assignment). `m` is a NumPy array of size `k x D` that contains `k` clusters means, each of size `1 x D`.

  - Outputs: Recall that to determine which cluster a data point x_n belongs to, we use the following equation:

    $$j = \arg\min_{i=1,\ldots,k} ||x_n - \mu_i||_2^2$$

    where the cluster means μ is the input `m` and j is the index of the closest cluster. As a result, this function returns a NumPy array `index` of size `H`. This array contains the assigned cluster index for `H` data points.

  - Function implementation considerations:

    You can use `numpy.argmin, numpy.sum` to complete the function.

- Function 4: `test_sckitlearn(train_data, test_data, k=5)`

  - Inputs: `test_data`, `m`

    The input `train_data`, `test_data` is our training and test data, which is a NumPy array of dimension `N x D` and `H x D` respectively. The scalar `k` is the number of clusters.

  - Outputs: This function uses scikit-learn to find the optimized cluster means (on `train_data`) and visualizes the results (on `test_data`).

  - Function implementation considerations:

    Use the `KMeans` class from sckitlearn to complete this function. Specifically, for the parameters, you will use `k` clusters, `5000` maximum iterations and `"auto"` algorithm. Note that in scikit-learn, "auto" algorithms refers to the Lloyd's algorithm.

Include this in your report. Compare the clustering performance between our PyTorch implementation and the scikit-learn one. You can do this by including in your report the visualization results between two implementations for `k=1,2,3,4,5`. Make sure to run them multiple times to see if the results are different for different runs. You can use `test_sckitlearn` and `test_pytorch` for this purpose. Comment your results.

The following is the mark breakdown for Part 1:

- Test file successfully runs implemented function: 25 marks
- Questions are answered correctly: 25 marks

Part 2: Mixture of Gaussians (MoG)

In this part, we will cluster our data by using MoG model. Recall that the objective function that we opt to maximize the log-likelihood $\log \mathrm{P}(x_1, x_2, ..., x_N)$. Assuming that our data is generated by a MoG model, we can write the log-likelihood as:

$$\log \mathrm{P}(x_1, x_2, ..., x_N) = \sum_{n=1}^{N} \log \mathrm{P}(x_n) = \sum_{n=1}^{N} \log \sum_{i=1}^{k} \mathrm{P}(z_n = i)\mathrm{P}(x_n | z_n = i) = \sum_{n=1}^{N} \log \sum_{i=1}^{k} \pi^i \mathcal{N}(x_n | \mu^i, \Sigma^i)$$

where π^i is the prior probability of the cluster i, μ^i and Σ^i are the mean and covariance of the Gaussian distribution i respectively. We will refer $\{\pi^i, \mu^i, \Sigma^i\}_{i=1}^k$ as the MoG parameters that we would like to find. To complete this part, you will first understand some implemented functions in `GMM.py` before completing the training loop for estimating the MoG's parameters. After that, you need to attach the visualization results in your report. To simplify the implementation, for each 2D Gaussian, we only assign a mean and a **single variance** (instead of a whole 2x2 covariance matrix).

Include this in your report Explains the functionality of these functions in your report.

1. `distanceFunc(X, MU)`: what are `X` and `MU`? What is the purpose of this function? Explains the output.
2. `log_GaussPDF(X, mu, sigma)`: what are `X` and `mu` and `sigma`? What is the purpose of this function? Explains the output.
3. `log_posterior(log_PDF, log_pi)`: what are `log_PDF` and `log_pi`? What is the purpose of this function? Explains the output.

Complete the following function After understanding the functionality of the above functions, you now can complete the training loop in `train_gmm(train_data, test_data, k = 5, epoch=1000)`.

- Function 1: `train_gmm(train_data, test_data, k = 5, epoch=1000)`

- Inputs: `train_data, test_data, k = 5, epoch=1000`
  The input `train_data` is a training data, which is a NumPy array of dimension `N x D` where `N` is the number of training points and `D` is the number of features (2 in this assignment). Similarly, `test_data` is the NumPy array for test data. `k` and `epoch` are the number of clusters and training epochs respectively.
- Output: This function returns the log-likelihood on test set, the MoG parameters as well as the cluster-association. The output `test_loss` measures the average log-likelihood on `test_data`. `log_joint_test` is a matrix of size `N x k` where N is the number of data points and k is the number of clusters. Each index `i,j` in `log_joint_test` is the likelihood of data point i with respect to the cluster j (the posterior). `pi` is an array of size `k`, which are the weights (or prior probability) for each cluster. `mu` is an array of size `k x D`, which are the means for the `k` clusters. Finally `sigma` is an array of size `k`, where each value in the array is the standard-deviation for the associated clusters.
- Function implementation considerations:
  This function first converts the `train_data` and `test_data` into a torch tensor. It then initializes the MoG parameters as trainable torch variables. In the training loop, the function computes and maximizes the average log-likelihood across the whole training data (using Adam optimizer). **You need to complete this loop by fill-in after the TODO comment**. Finally, after training, the model evaluates its performance on `test_data` and returns the parameters as well as the cluster-association. **You need to complete the evaluation process by fill-in after the TODO comment**.

Include this in your report. We provided the `test_GMM(k)` function, which optimize the MoG parameters and visuaize the cluster. This function only requires `k`, which is the number of clusters you want to specify.

1. For each `k=1,2,3,4,5`, runs `test_GMM(k)` multiple times and visualize the results.
2. Include the visualization for `k=1,2,3,4,5` in your report. Comment on the results.

The following is the mark breakdown for Part 2:

- Test file successfully runs implemented function: 25 marks
- Questions are answered correctly: 25 marks