

**SYDE675: Introduction to Pattern Recognition****Assignment 2****Due: 11:59 PM (EST), Feb 27, 2024**, submit on LEARN.

Include your name and student number!

Submit your write-up in pdf and all source code in a zip file (with proper documentation). Write a script for each programming exercise so that the TAs can easily run and verify your results. Make sure your code runs!

[Text in square brackets are hints that can be ignored.]

**NOTE** For all the exercises, you are only allowed to use the basic Python, Numpy, and matplotlib (for plotting) libraries, unless specified otherwise.

**Exercise 1: Parametric and Non-Parametric Estimation (35 pts)**

MENTION THAT -VE VALUES CAN BE DISCARDED FOR THE EXPONENTIAL DISTRIBUTION

In this exercise, you will be using the **MNIST** dataset for image classification. To work with this dataset, you will first need to flatten your images from  $28 \times 28$  to  $784 \times 1$  vectors. Next, use the **PCA** in scikit learn to convert the  $784 \times 1$  vectors to  $1 \times 1$  vectors (or real numbers). **Note** that the dataset consists of the training and the test sets. Use the training set for implementing the classifiers in the exercise. **Also**, use only two classes in the dataset i.e. the two classes representing numbers 3 and 4.

1. (10 pts) We derived the expressions for the mean and standard deviation of the Gaussian distribution in class. Now, let's assume the probability distribution of your dataset is not just a single Gaussian distribution but a product of a Gaussian distribution and an exponential distribution:

$$p(X|\theta) = \frac{1}{2}[\mathcal{N}(X|\theta_1, \theta_2) \times f(x; \lambda)] \quad (1)$$

In this case, you will be estimating three parameters ( $\theta_1$ ,  $\theta_2$ , and  $\lambda$ ), where  $\theta_1$  and  $\theta_2$  are the mean and variance of the Gaussian distribution. Using MLE, derive an expression for finding the three parameters of your probability distribution defined by eq. (1).

2. (5 pts) Using the formula derived for the above distribution and the parameters derived for a Gaussian distribution in class, develop two ML classifiers using the training data of the two MNIST classes. Test the two ML classifiers on the test set of the MNIST dataset. Which of the two ML classifiers is the best? Why?
3. (15 pts) Now let's move on to the non-parametric estimation. Use kernel-based density estimation with a Gaussian kernel of  $\sigma = 20$  to estimate the probability distributions of the two MNIST classes. Note that  $\sigma$  represents the scaling factor for the Gaussian kernel. Use an ML-based classifier using this probability estimation, and report the test accuracy on the MNIST dataset.
4. (5 pts) Comparing the non-parametric and the two parametric estimation methods, which one is the best? Explain.

**Exercise 2: K-means clustering (35 pts)**

**NOTE:** You are allowed to compare your results with the Scikit implementation of k-means, but you cannot use the Scikit implementation as your own solution.

In this exercise you will be using the MNIST dataset as in the previous exercise. You will also be using  $784 \times 1$  directly without any PCA. You can use the training data of the MNIST dataset for this exercise. Use all the classes in the dataset.

1. (10 pts) In the class we looked at the K-means clustering algorithm with the Euclidean distance. Now, let's implement the k-means clustering algorithm with two different distance metrics, the cosine distance and the Mahalanobis distance. You can use the scikit implementation of the cosine distance. For the Mahalanobis distance, you do not need to calculate the inverse of the covariance using eigenvalue decomposition.
2. (5 pts) Apply your k-means implementation to the MNIST dataset. Use  $k = \{5, 10, 20, 40, 200\}$ . Also, you

will not be using class labels during the clustering process as it is an unsupervised learning problem.

3. (5 pts) As we do not use class labels for clustering we cannot use test error to evaluate the k-means algorithm. Instead, we can test our clustering implementation using cluster consistency: all data points in a cluster should belong to the same class. For example, if your cluster number 1 has all the data points belonging to class “5” then this cluster has perfect consistency. You can find the cluster consistency  $Q_i$  for each cluster  $i$  as follows:

$$Q_i = \frac{m_i}{N_i} \quad (2)$$

where,  $N_i$  is the total number of data points in cluster  $i$ , and  $m_i$  represents the total number of data points belonging to the most common class in cluster  $i$ . You can find  $m_i$  by first counting the total number of data points belonging to each of the 10 classes in cluster  $i$ , and then taking the max of this list. To find the overall clustering consistency:

$$Q = \frac{1}{k} \sum_{i=1}^k Q_i \quad (3)$$

Report the cluster consistency of your k-means clustering on the MNIST dataset for all four values of  $k$ .

4. (5 pts) Which  $k$  value produces the best results? Explain. Can the results from cluster consistency be misleading? Explain. [HINT Intuitively, what  $k$  value should produce the best results on the MNIST dataset?]
5. (10 pts) What can you do to further validate your results if the cluster consistency metric is not working? Can you use the objective function defined in the class to find out the internal cluster distance of the data points from the mean? How can this objective help determine any misleading clustering results? Explain and demonstrate this method on the clustering results of the previous steps.

### Exercise 3: Gaussian Mixture Model (GMM) (30 pts)

For a **diagonal matrix**  $\text{diag}(\mathbf{s})$ ,  $|\text{diag}(\mathbf{s})| = \prod_i s_i$ .

**Note:** For 3.1, you only need to implement (or analyze) your algorithm without testing it on any dataset. Implementations in 3.1 will be tested in 3.2. Therefore, any questions related to the testing of your implementation in 3.1 on a dataset, can be answered in 3.2.

**Algorithm 1: EM for GMM.**


---

**Input:**  $X \in \mathbb{R}^{n \times d}$ ,  $K \in \mathbb{N}$ , initialization for *model*  
 // *model* includes  $\pi \in \mathbb{R}_+^K$  and for each  $1 \leq k \leq K$ ,  $\mu_k \in \mathbb{R}^d$  and  $S_k \in \mathbb{S}_+^d$   
 //  $\pi_k \geq 0$ ,  $\sum_{k=1}^K \pi_k = 1$ ,  $S_k$  symmetric and positive definite.  
 // random initialization suffices for full credit.  
 // alternatively, can initialize  $r$  by randomly assigning each data to one of the  $K$  components  
**Output:** *model*,  $\ell$

```

1 for iter = 1 : MAXITER do
  // step 2, for each  $i = 1, \dots, n$ 
2   for  $k = 1, \dots, K$  do
3      $r_{ik} \leftarrow \pi_k |S_k|^{-1/2} \exp[-\frac{1}{2}(\mathbf{x}_i - \mu_k)^\top S_k^{-1}(\mathbf{x}_i - \mu_k)]$  // compute responsibility
  // for each  $i = 1, \dots, n$ 
4    $r_{i.} \leftarrow \sum_{k=1}^K r_{ik}$ 
  // for each  $k = 1, \dots, K$  and  $i = 1, \dots, n$ 
5    $r_{ik} \leftarrow r_{ik} / r_{i.}$  // normalize
  // compute negative log-likelihood
6    $\ell(\text{iter}) = -\sum_{i=1}^n \log(r_{i.})$ 
7   if iter > 1 &&  $|\ell(\text{iter}) - \ell(\text{iter} - 1)| \leq \text{TOL} * |\ell(\text{iter})|$  then
8     break
  // step 1, for each  $k = 1, \dots, K$ 
9    $r_{.k} \leftarrow \sum_{i=1}^n r_{ik}$ 
10   $\pi_k \leftarrow r_{.k} / n$ 
11   $\mu_k = \sum_{i=1}^n r_{ik} \mathbf{x}_i / r_{.k}$ 
12   $S_k \leftarrow (\sum_{i=1}^n r_{ik} \mathbf{x}_i \mathbf{x}_i^\top / r_{.k}) - \mu_k \mu_k^\top$ 

```

---

- (15 pts) Derive and implement the EM algorithm for the **diagonal** Gaussian mixture model (dGMM), where all covariance matrices are constrained to be diagonal. Note that the above algorithm does not make any assumptions about the covariance matrices, however, in our class, we assumed the covariances to be identity matrices. Algorithm 1 recaps all the essential steps and serves as a hint rather than a verbatim instruction. In particular, you must change the highlighted steps accordingly (with each  $S_k$  being a diagonal matrix), along with formal explanations..

[You might want to review the steps we took in class to get the updates in Algorithm 1 and adapt them to a bit more complex case here. The solution should look like  $s_j = \frac{\sum_{i=1}^n r_{ik} (x_{ij} - \mu_j)^2}{\sum_{i=1}^n r_{ik}} = \frac{\sum_{i=1}^n r_{ik} x_{ij}^2}{\sum_{i=1}^n r_{ik}} - \mu_j^2$  for the  $j$ -th diagonal. Multiplying an  $n \times p$  matrix with a  $p \times m$  matrix costs  $O(mnp)$ . Do not maintain a diagonal matrix explicitly; using a vector for its diagonal suffices.]

To stop the algorithm, set a maximum number of iterations (say  $\text{MAXITER} = 500$ ) and also monitor the change of the negative log-likelihood  $\ell$ :

$$\ell = -\sum_{i=1}^n \log \left[ \sum_{k=1}^K \pi_k (2\pi)^{-d/2} |S_k|^{-1/2} \exp[-\frac{1}{2}(\mathbf{x}_i - \mu_k)^\top S_k^{-1}(\mathbf{x}_i - \mu_k)] \right], \quad (4)$$

where  $\mathbf{x}_i$  is the  $i$ -th column of  $X^\top$ . As a debug tool, note that  $\ell$  should decrease from step to step, and we can stop the algorithm if the decrease is smaller than a predefined threshold, say  $\text{TOL} = 10^{-5}$ .

- (15 pts) Next, we apply (the adapted) Algorithm 1 in Ex 2.1 to the **MNIST** dataset (that you already experimented on before). For each of the 10 classes (digits), we can use its (and only its) training images to estimate its (class-conditional) distribution by fitting a GMM (with say  $K = 5$ , roughly corresponding to 5 styles of writing this digit). This gives us the density estimate  $p(\mathbf{x}|y)$  where  $\mathbf{x}$  is an image (of some

digit) and  $y$  is the class (digit). We can now classify the test set using the Bayes classifier:

$$\hat{y}(\mathbf{x}) = \arg \max_{c=0,\dots,9} \underbrace{\Pr(Y=c) \cdot p(X=\mathbf{x}|Y=c)}_{\propto \Pr(Y=c|X=\mathbf{x})}, \quad (5)$$

where the probabilities  $\Pr(Y=c)$  can be estimated using the training set, e.g., the proportion of the  $c$ -th class in the training set, and the **density**  $p(X=\mathbf{x}|Y=c)$  is estimated using GMM for each class  $c$  separately. Report your error rates on the test set as a function of  $K$ . Use 5, 10, 20, 30 as the values for  $K$ . Note that if time is a concern, using  $K=5$  will receive full credit.

[Optional: Reduce dimension by **PCA** may boost accuracy quite a bit. Your running time should be on the order of minutes (for one  $K$ ), if you do not introduce extra for-loops in Algorithm 1.]

[In case you are wondering, our classification procedure above belongs to the so-called plug-in estimators (plug the estimated densities to the known optimal Bayes classifier). However, note that estimating the density  $p(X=\mathbf{x}|Y=c)$  is actually harder than classification. Solving a problem (e.g. classification) through some intermediate harder problem (e.g. density estimation) is almost always a bad idea. **Extra: 5 pts** Do you think this will be true for harder classification problems (such as **indoor scene classification**)? Explain.]