Analyse et Traitement de l'Information

TP4: Clustering approaches and autoregression.

The MNIST dataset of handwritten digits [1] consists of 28×28 grayscale images. To download the dataset use this code:

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split

images, labels = fetch_openml('mnist_784', \
            return_X_y=True, as_frame=False, parser='auto')
train_images, test_images, train_labels, test_labels = \
            train_test_split(images, labels, random_state=42)
```

To filter images of the specific class you can use the following method:

```
import numpy as np

def select_with_label(images, labels, desired_labels):
    mask = np.isin(labels, desired_labels)
    return images[mask], labels[mask]

images_of_two, labels_of_two = \
        select_with_label(train_images, train_labels, desired_labels=['2'])
images_of_odd, labels_of_odd = \
        select_with_label(train_images, train_labels, \
        desired_labels=['1', '3', '5', '7', '9'])
```

You can copy-paste from this link.

[1] http://yann.lecun.com/exdb/mnist/

Clustering

Classification is a *supervised* problem, where labeled (training) data is used to label the unlabeled (testing) data.

Clustering is an *unsupervised* problem, where the objective is to group a given (unlabeled) dataset $\{\mathbf{x}\}_{i=1}^{n}$ into k clusters, so that similar samples appear in the same cluster, and dissimilar samples are in different clusters.

1 *k*-means

k-means clustering is a type of unsupervised learning that is employed to group data into K clusters. The primary steps involved in the k-means algorithm are as follows:

1. **Initialization**:
   - Choose the number of clusters K in advance.
   - Randomly initialize the centroids $\mu_1, \mu_2, ..., \mu_K$ of the clusters.
2. **Assignment Step**:
   - For each data point x_n, find the nearest centroid μ_i by comparing the Euclidean distance between x_n and all the centroids.
   - Assign each data point to the nearest centroid using the formula:

$$S_i^{(t)} = \left\{ x_n : \left\| x_n - \mu_i^{(t)} \right\|^2 \leq \left\| x_n - \mu_j^{(t)} \right\|^2 \ \forall j, 1 \leq j \leq k \right\} \tag{1}$$

     Here, $S_i^{(t)}$ represents the set of all data points assigned to the i-th cluster at iteration t.
   - Define z_{nk} as:

$$z_{nk} = \begin{cases} 1 & \text{if } x_n \text{ is assigned to cluster } k \\ 0 & \text{otherwise} \end{cases}$$

3. **Update Step**:
   - Recalculate the centroids by computing the mean of all data points assigned to each centroid's cluster:

$$\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{2}$$

4. **Convergence**:
    - Repeat the Assignment and Update steps until the centroids no longer change or change minimally.
5. **Objective Function Optimization**:
    - The algorithm aims to minimize the objective function L given by:

$$L = \sum_{i=1}^{N} \sum_{k=1}^{K} z_{nk} ||x_n - \mu_k||^2 \tag{3}$$

6. **Local Minima Issue**:
    - The k-means algorithm can converge to a local minimum, which is usually dependent on the initial positioning of centroids.
    - A common approach to mitigate this issue is to run the algorithm multiple times with different initial centroid positions, and then choose the clustering result that appears most frequently or has the lowest value of the objective function L.

These steps encapsulate the k-means algorithm's iterative process of assignment and update to achieve a clustering that minimizes the within-cluster variance.

```
class KMeans:
    def __init__(self, n_clusters, init="random", max_iter=300):
        raise NotImplementedError()

    def fit(self, X, y=None):
        # initialize centroids randomly
        # this function should update self.centroids coordinates
        # for given X
        raise NotImplementedError()

    def predict(self, X):
        # this function for each X returns the cluster
        # it belongs to
        raise NotImplementedError()

```

Questions 1

1.1 Create a subset of MNIST which consists of 2000 randomly sampled instances of "3"s and "7"s.

1.2 Implement k-means from scratch (do not use *sklearn*) with random initialization of the clusters. Perform the algorithm with $K = 2, 3, \ldots, 10$. Compute L defined in Equation 3 for each K. Plot L varying with K. Describe the effect of K over L.

1.3 *(Bonus, 20 points) Implement k++ assignment explained in the lecture. Compare the results of several runs with the previous implementation and k++ initialization.*

2 Gaussian Mixture Models

Definition

Mixture models assume that the data is generated from a mixture of c "basis" functions (components), each belonging to a family F. The density $f(x)$ is given by:

$$f(x) = \sum_{j=1}^{c} \alpha_j \phi(x; \theta_j)$$

where:

- α_j are the mixture coefficients, such that $\sum_j \alpha_j = 1$.
- $\phi(x; \theta_j)$ are functions controlled by parameters θ_j.

Assumptions

1. The number of components c is known.
2. The family of functions F is known.
3. Labels (class labels) are unknown.

Density Representation

Density $f(x)$ represents a random process where x is drawn from a set of states ω_j with prior probability $\Pi(\omega_j)$:

$$f(x) = \sum_{j=1}^{c} \alpha_j \phi(x; \theta_j) = \Pi(x|\theta) = \sum_{j=1}^{c} \Pi(\omega_j)\Pi(x|\omega_j; \theta_j)$$

Identifications:

- $\alpha_j = \Pi(\omega_j)$ such that $\sum_j \alpha_j = 1$.
- $\phi(x; \theta_j) = \Pi(x|\omega_j; \theta_j)$ is the conditional probability that x is generated by state ω_j.

Given $X = \{x_1, ..., x_N\}$ unlabeled samples generated by the mixture, the parameters to infer are $\theta = [\alpha_j, \theta_j]_{j \in [c]}$.

Likelihood and Log-likelihood

Likelihood:

$$\Pi(X|\theta) = \prod_{i=1}^{N} \Pi(x_i|\theta)$$

Log-likelihood:

$$\mathbb{L}(\theta; X) = \sum_{i=1}^{N} \log \left(\sum_{j=1}^{c} \alpha_j \phi(x_i; \theta_j) \right)$$

Gaussian Mixture Model

In a Gaussian Mixture Model (GMM), the basis functions are chosen to be Gaussian:

$$f(x) = \sum_{j=1}^{c} \alpha_j \mathcal{N}(x|\mu_j, \Sigma_j)$$

Expectation (E-step)

Given an initial value for $\theta^0 = [\alpha_j^0, \mu_j^0, \Sigma_j^0]_{j \in [c]}$, compute the responsibility γ_{ij} of every data x_i to every component θ_j:

$$\gamma_{ij}(\theta^0) = \frac{\alpha_j^0 \mathcal{N}(x_i|\mu_j^0, \Sigma_j^0)}{\sum_{k=1}^{c} \alpha_k^0 \mathcal{N}(x_i|\mu_k^0, \Sigma_k^0)}$$

Maximization (M-step)

Estimate the parameters of every component by (weighted) maximum likelihood:

$$\begin{aligned}
\hat{\alpha}_j &= \frac{1}{N} \sum_{i=1}^{N} \gamma_{ij} \\
\hat{\mu}_j &= \frac{\sum_{i=1}^{N} \gamma_{ij} x_i}{\sum_{i=1}^{N} \gamma_{ij}} \\
\hat{\Sigma}_j &= \frac{\sum_{i=1}^{N} \gamma_{ij} (x_i - \hat{\mu}_j)(x_i - \hat{\mu}_j)^T}{\sum_{i=1}^{N} \gamma_{ij}}
\end{aligned}$$

EM Algorithm for c-component Mixtures

1. Initialize $\theta^0 = [\alpha_j^0, \mu_j^0, \Sigma_j^0]_{j \in [c]}$.
2. **E-step**: Compute responsibilities γ_{ij}.
3. **M-step**: Update $\alpha_j, \mu_j, \Sigma_j$ using the equations above.
4. Iterate steps 2 and 3 until convergence.

Over-parameterized Problem

In high-dimensional spaces, estimating the covariance matrices becomes challenging due to the large number of parameters ($O(D^2)$). Solutions include:

- Spherical models: $\Sigma = \sigma^2 I$, 1 parameter.
- Diagonal models: $\Sigma = \text{diag}(\sigma_1, ..., \sigma_D)$, D parameters.
- Full models: Σ is a full covariance matrix, $O(D^2)$ parameters.

Thus, dimensionality reduction is needed (PCA).

```python
class GMM:
    def __init__(self, n_components, max_iter = 100):
        self.n_componets = n_components
        self.max_iter = max_iter
        # pi list contains the fraction of the dataset for every cluster
        # self.pi = [1/self.n_componets] * self.n_componets
        self.pi = [1/self.n_componets for comp in range(self.n_componets)]


    def fit_predict(self, X, y=None):
        #this function returns a cluster number to each element of X
        raise NotImplementedError()

```

Questions 2

2.1 Use a subset of MNIST consisting of 2000 randomly sampled instances of "3"s and "7"s from the previous task. Apply PCA to reduce dimensionality. Implement GMM model from scratch (do not use *sklearn*) with random initialization of the parameters. Perform the algorithm with k = 2, 3, . . . , 10.

2.2 Build confusion matrices using *sklearn* for k-means and GMM.

2.3 Describe the differences between k-means and GMM (deterministic/probabilistic, parameters, functions to optimize, updates).

2.4 Describe how k-means can be seen from the perspective of the EM-algorithm (*hint:* distance instead of the probability function).

Submission

Please archive your report and codes in "Prénom Nom.zip" (replace "Prénom" and "Nom" with your real name), and upload to "`Upload TP4 - Clustering approaches and autoregression.`" on `https://moodle.unige.ch` before Monday, November 13 2023, 21:59 PM. Note, that the assessment is mainly based on your report, which should include your answers to all questions and the experimental results. *Importance is given on the mathematical explanations of your works and your codes should be commented*

Please use the template from the first TP.