Machine Learning 1, SS24
Homework 4
K-means. Expectation-Maximization Algorithm. Maximum Likelihood Estimation.

Thomas Wedenig, thomas.wedenig@tugraz.at

Tutor:	Sofiane Correa de Sa, correadesa@student.tugraz.at
Points to achieve:	25 pts
Bonus points:	5* pts
Deadline:	07.07.2024 23:59
Hand-in procedure:	Use the **cover sheet** that you can find in the TeachCenter. Submit `main.py`, `k_means.py`, `em.py`, `utils.py` and **a report (PDF)**. Do not change the file name of the Python files. Do not upload a zip file.
Submissions after the deadline:	Each missed day brings a (-5) points penalty.
Plagiarism:	If detected, 0 points for all parties involved. If this happens twice, we will grade the group with "Ungültig aufgrund von Täuschung"
Course info:	TeachCenter, https://tc.tugraz.at/main/course/view.php?id=1648

Contents

General remarks

- **Do not add any additional `import` statements** anywhere in the code.
- **Do not modify the function signatures** of the skeleton functions (i.e., the function names and inputs).
- **Do not change the file name** of any of the Python files.
- Include **all plots and results** in your PDF report.

Failing to conform to these rules will lead to point deductions. Further, your submission will be graded based on:

- Correctness (Is your code doing what it should be doing? Is your derivation correct?)
- The depth of your interpretations (Usually, only a couple of lines are needed.)
- The quality of your plots (Is everything clearly readable/interpretable? Are axes labeled? …)
- Your submission should run with Python 3.9+.

1 K-means. Expectation-Maximization Algorithm [25 points]

In this task, we will implement the K-means and Expectation-Maximization algorithms, and evaluate them using the *Mickey Mouse dataset*. Functions for loading the dataset, plotting the original data, and plotting the clusters are already provided.

1.1 K-means Algorithm [8 points]

Suppose we are given a dataset $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}\}$ (with $\mathbf{x}^{(i)} \in \mathbb{R}^D$ for all $i \in \{1, \dots, N\}$), we aim to partition the dataset into K clusters $\{\mathcal{D}_1, \dots, \mathcal{D}_K\}$. Each cluster $\mathcal{D}_k$ is represented by a *centroid vector* $\boldsymbol{\mu}^{(k)} \in \mathbb{R}^D$. Recall that the number of clusters K is a hyperparameter that we need to choose.

One approach to clustering is to solve the k-means problem, i.e., minimize the *within-cluster sum-of-squares* (WCSS):

$$\min_{\boldsymbol{\mu}^{(1)}, \dots, \boldsymbol{\mu}^{(K)}, Z} \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ik} \|\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k)}\|_2^2 \tag{1}$$

with $z_{ik} \in \{0, 1\}$ and $\sum_k z_{ik} = 1$ for all $i \in \{1, \dots, N\}$. Note that z_{ik} is a binary indicator variable that is 1 if datapoint $\mathbf{x}^{(i)}$ belongs to cluster k, and 0 else. More formally,

$$z_{ik} = \begin{cases} 1 & \text{if } k = \text{argmin}_j \, ||\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(j)}||_2^2 \\ 0 & \text{else} \end{cases} \tag{2}$$

This means, to each data point a one-hot vector of length K is assigned – there is exactly one 1 in the vector at the index of the assigned cluster, and the remaining entries of the vector are zeros.

The centroids (cluster means) are calculated as follows:

$$\boldsymbol{\mu}^{(k)} = \frac{1}{\sum_i z_{ik}} \sum_i z_{ik} \mathbf{x}^{(i)} \tag{3}$$

Tasks:

1. In the code (file `k_means.py`), in the function **wcss()** implement the *WCSS* objective function from Equation 1.
2. In the code (file `k_means.py`), implement the function **closest_centroid()**. This function takes a single sample (data point), calculates the distances to all centroids, and returns the integer index of the closest centroid. This implementation corresponds to the argmin part of Equation 2.
3. In the code (file `k_means.py`) implement the function **compute_Z()** by using Equation 2. Construct a Z matrix with the corresponding one-hot vectors on its rows. Note that you should use the function **closest_centroid()** that returns the index of the closest centroid.
4. In the code (file `k_means.py`), in the function **recompute_centroids()** implement Equation 3.
5. In the code (file `k_means.py`), in the function **kmeans()** add function calls for assigning samples to clusters, then evaluate the objective function, recompute the centroids, and calculate the objective function again.
6. In `main.py`, in the function **task_kmeans()**, choose the appropriate number of clusters K and maximum iterations `max_iter`. There is already a function call to **kmeans()** provided.
7. There is already a function call to plot the Mickey Mouse dataset after clustering. Include the plot in the report. Compare it with the plot of the original data (in one or two sentences).
8. Set $K = 5$. Include the plot of the clustered dataset in the report. Compare it with the plot of the original data (in one or two sentences).

1.2 Expectation-Maximization (EM) Algorithm [12 points]

A Gaussian Mixture Model (GMM) with K components is given as:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{k=1}^{K} w_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \Sigma_k) \tag{4}$$

with $\boldsymbol{\theta} = \{w_k, \boldsymbol{\mu}_k, \Sigma_k\}_{k=1}^{K}$ where w_k represent the component weights (with the constraints that $\sum_{k=1}^{K} w_k = 1$ and $0 \leq w_k$ for all $k \in \{1, \ldots, K\}$), $\boldsymbol{\mu}_k$ are the means, and Σ_k are the covariance matrices.

Given a dataset $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, the steps of the EM algorithm are:

- Initialize the parameters $\boldsymbol{\theta} = \{w_k, \boldsymbol{\mu}_k, \Sigma_k\}_{k=1}^{K}$ using some initialization strategy.
- **Expectation step (E-step):** For each sample $\mathbf{x}^{(i)}$, calculate the cluster responsibilities (i.e., the probability that the sample was caused by the k-th component of the GMM):

$$\gamma_{ik} = \frac{w_k \mathcal{N}(\mathbf{x}^{(i)}; \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{k'=1}^{K} w_{k'} \mathcal{N}(\mathbf{x}^{(i)}; \boldsymbol{\mu}_{k'}, \Sigma_{k'})} \tag{5}$$

  We can think of this as a soft version of Equation 2.

- **Maximization step (M-step):** Given the soft cluster assignments γ_{ik}, we can update $\boldsymbol{\theta}$:

$$\boldsymbol{\mu}_k \leftarrow \frac{1}{\sum_{i=1}^{N} \gamma_{ik}} \sum_{i=1}^{N} \gamma_{ik} \mathbf{x}^{(i)} \tag{6}$$

$$\Sigma_k \leftarrow \frac{1}{\sum_{i=1}^{N} \gamma_{ik}} \sum_{i=1}^{N} \gamma_{ik} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)(\mathbf{x}^{(i)} - \boldsymbol{\mu}_k)^T \tag{7}$$

$$w_k \leftarrow \frac{1}{N} \sum_{i=1}^{N} \gamma_{ik} \tag{8}$$

- Evaluate the log-likelihood function using the new parameters $\boldsymbol{\theta}$:

$$\log\left(p_{\boldsymbol{\theta}}(\mathcal{D})\right) = \sum_{i=1}^{N} \log\left(\sum_{k=1}^{K} w_k \mathcal{N}(\mathbf{x}^{(i)}; \boldsymbol{\mu}_k, \Sigma_k)\right) \tag{9}$$

  and check if the log-likelihood is the same as in the previous step (up to a small constant $\varepsilon > 0$). If this is the case, the algorithm has converged. Otherwise, repeat the EM steps until convergence.

Tasks:

1. In the code (file `em.py`), in the function **calculate_responsibilities()** implement the E-Step of EM algorithm, namely, Equation 5 that calculates responsibilities γ_{ik} for each sample $\mathbf{x}^{(i)} \in \mathcal{D}$ and each cluster $k \in \{1, \ldots, K\}$.
2. In the code (file `em.py`), in the function **update_parameters()** implement the M-Step of EM algorithm, that is, Equation 6, Equation 7, and Equation 8.
3. In the code (file `em.py`), in the function **em()** implement the steps of EM algorithms that are missing. A strategy for initializing $\boldsymbol{\theta}$ is already provided.
4. In **task_em()** (file `main.py`), set K to the value that you used in K-means (in order to compare it with the results from the previous task), and choose an appropriate value for `max_iter`. The function **plot_objective_function()** will plot the log-likelihood over the iterations. Include the plot in the report.
5. There is already a function call to plot the Mickey Mouse after clustering. Include the plot in the report. Compare it with the plot of the original data (in one or two sentences). Include in the report the initial values of $(w_1, \ldots, w_K)$ and the final values (after the algorithm converged).

1.3 Summary and comparison of two algorithms [5 points]

Tasks:

1. Which algorithm works better for the Mouse data set? Why? Explain by comparing the plots of the original data, after K-means clustering, and after EM clustering using the GMM.
2. Are there noisy data points (outliers) in the original data? Is K-means robust to noise? Is EM robust to noise?
3. What is **the main difference** between the K-means and EM algorithm? Briefly discuss the connection between z_{ik} (K-means) and γ_{ik} (in EM).
4. In the EM algorithm, the covariance matrices are updated in each iteration. However, the K-means algorithm does not have these parameters. What shape of clusters does K-means tend to produce? Hence, what are the assumed (implicit) covariance matrices for clusters in K-means?
5. In Task 1.2, we have learned parameters of a Gaussian Mixture Model, which we used to generate soft cluster assignments. Learning parameters of a probability distribution over our training data is called **density estimation** and the resulting model can do more than just soft clustering our data points: For example, it is common to use such distributions to draw **samples** from this density (this is called **generative modeling**). Thus, assume we wish to sample a new data point $\mathbf{x}$ from our trained GMM $p_{\boldsymbol{\theta}}$. Briefly explain how we can interpret the component weights $(w_1, \ldots, w_K)$ as prior probabilities over a categorical latent variable and how this interpretation can be used to sample from $p_{\boldsymbol{\theta}}$. Write down the definition of $p_{\boldsymbol{\theta}}$ using this interpretation and state all steps needed to draw a sample from $p_{\boldsymbol{\theta}}$.

2 Bonus: Maximum Likelihood Estimation [5* points]

Assume we are given a dataset $\mathcal{D} = (\mathcal{X}, \mathcal{Y})$ with $\mathcal{X} = \{(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)})\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^D$, and $\mathcal{Y} = \{y^{(1)}, \dots, y^{(N)}\}$ with $y^{(i)} \in \mathbb{R}$ for all $i \in \{1, \dots, N\}$. We want to solve a supervised regression problem using a model $f_{\boldsymbol{\theta}}$ that tries to predict y from $\mathbf{x}$. For example, $f_{\boldsymbol{\theta}}$ could represent a neural network.

We will now adopt a probabilistic perspective on this problem and show the equivalence between *Maximum Likelihood Estimation* and minimizing popular loss functions for training regression models.

Tasks:

1. Assume that the likelihood of a point $y^{(i)}$ given the features $\mathbf{x}^{(i)}$ is Gaussian with mean $f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$ and variance σ^2 (which does not depend on $\mathbf{x}^{(i)}$), i.e.,

$$p_{\boldsymbol{\theta}}(y^{(i)} \mid \mathbf{x}^{(i)}) = \mathcal{N}(y^{(i)}; f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \sigma^2) \quad \text{where } \mathcal{N}(y^{(i)}; f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2}(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)})^2\right)$$

   If we assume that all $y^{(i)} \in \mathcal{Y}$ were independently drawn from their corresponding conditional Gaussian distribution, we can factorize the likelihood of $\mathcal{Y}$ as follows:

$$p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X}) = \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(y^{(i)} \mid \mathbf{x}^{(i)})$$

   Recall that we have often tried to find parameters of $f_{\boldsymbol{\theta}}$ by minimizing the mean squared error:

$$\boldsymbol{\theta}^*_{\text{MSE}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \left(f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$

   Show that under the assumptions above, we have $\boldsymbol{\theta}^*_{\text{MSE}} = \boldsymbol{\theta}^*_{\text{MLE}}$ where $\boldsymbol{\theta}^*_{\text{MLE}}$ is the *maximum likelihood estimate* given by

$$\boldsymbol{\theta}^*_{\text{MLE}} = \operatorname*{argmax}_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X}) = \operatorname*{argmax}_{\boldsymbol{\theta}} \log\left(p_{\boldsymbol{\theta}}(\mathcal{Y} \mid \mathcal{X})\right)$$

   Provide all steps of your derivation.

   Hints: Use the following generic facts about minimizers/maximizers:

   - $\operatorname{argmin}_{\mathbf{z}} g(\mathbf{z}) = \operatorname{argmin}_{\mathbf{z}} \log(g(\mathbf{z}))$
   - $\operatorname{argmax}_{\mathbf{z}} g(\mathbf{z}) = \operatorname{argmin}_{\mathbf{z}} -g(\mathbf{z})$
   - $\operatorname{argmin}_{\mathbf{z}} \alpha g(\mathbf{z}) + \beta = \operatorname{argmin}_{\mathbf{z}} g(\mathbf{z})$

   where g is some function and α, β are constants w.r.t. $\mathbf{z}$. Also, recall that $\forall a, b \in \mathbb{R} : \log(a \cdot b) = \log(a) + \log(b)$.