Karim Abou Zeid <abouzeid@vision.rwth-aachen.de>

Exercise 1: Python Tutorial, Probability Density, GMM, EM

due **on** 2023-11-04

Important information regarding the exercises:

- Use the Moodle system to submit your solution. You will also find your corrections there.

- Due to the large number of participants, we require you to submit your solution to Moodle **in groups of 3 students**. You can use Moodle to form groups.

- Upload your pen & paper solutions as PDF. If your solutions are handwritten, ensure that your handwriting is legible, and that the pictures are not blurred and are taken under appropriate lighting conditions. All non-readable submissions will be discarded immediately.

- Upload your implementaion as '.zip' or '.tar.gz' file.

Question	Points	Bonus	Score
Python Tutorial	0	0	
Warming up!	4	0	
Maximum Likelihood	6	0	
Expectation Maximization (EM) Algorithm	10	4	
Σ Points	20	4	

Question 1: Python Tutorial .... $\left(\mathbf{\Sigma = 0}\right)$

To goal of this task is to get familiar with Python. **Please do not hand in a solution for this question.** A comprehensive Python tutorial can be found in `https://docs.python.org/3/tutorial/`.

(a) Install Python from `https://www.python.org/downloads/` Set the Python environment and run Python using the command `python` in your shell (see `https://docs.python.org/3/tutorial/interpreter.html`).

(b) Go through chapters 3, 4, and 5 in `https://docs.python.org/3/tutorial/`.

(c) Install NumPy and Matplotlib by running `pip install numpy matplotlib` in your shell.

(d) Walk through the basic NumPy tutorial (`https://numpy.org/doc/stable/user/quickstart.html`).

Note: We suggest you to use Visual Studio Code (`https://code.visualstudio.com/`) together with the Python extension, or the PyCharm IDE (`https://www.jetbrains.com/pycharm/`) for writing Python code.

Question 2: Warming up! .... $\left(\mathbf{\Sigma = 4}\right)$

Suppose that we have three colored boxes r (red), g (green), and b (blue). Box r contains 3 apples, 4 oranges, and 3 limes, box g contains 3 apples, 3 oranges, and 4 limes, and box b contains 1 apple, 1 orange, and 0 limes. A box is chosen at random with probabilities $p(r) = 0.2$, $p(g) = 0.6$, $p(b) = 0.2$, and a piece of fruit is removed from the box (with equal probability of selecting any of the items in the box), then

(a) What is the probability of selecting an apple? **(2 pts)**

(b) If we observe that the selected fruit is in fact an orange, what is the probability that **(2 pts)**
it came from the green box?

Question 3: Maximum Likelihood .... $(\Sigma = 6)$
Consider the following probablity density function

$$p(x|\theta) = \frac{1}{\sqrt{2\pi\theta}} \exp\left(-\frac{(x-3)^2}{2\theta}\right).$$

Given N measurements $x_1, ..., x_n > 0$ of x, what is the maximum likelihood estimate $\hat{\theta}$?

Question 4: Expectation Maximization (EM) Algorithm .... $(\Sigma = 10 + 4)$
In this exercise you shall implement the EM algorithm to solve a computer vision task
of image segmentation. In particular, you will use the EM algorithm to fit one Gaussian
Mixture Model (GMM) to samples of skin pixels (sdata) and one to samples of non-skin
pixels (ndata). Each sample is an RGB color value. These trained GMMs will be used to
segment an image into skin color and non-skin color regions.

Notes:

- For your reference, the equations for the D-dimensional GMMs can be found in the
  book of C. M. Bishop "Pattern Recognition and Machine Learning" (Chapter 9).
- You will need to install the following dependencies: `pip install scikit-learn`
  `numpy matplotlib imageio`
- You can use the provided 'apply.py' script to test your implementation. Comment out
  the unrelated parts to test the individual subexercises.

(a) Implement the function `getLogLikelihood` that computes the log-likelihood **(2 pts)**

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

of a mixture of Gaussian distributions with the signature

```
1 def getLogLikelihood(means, weights, covariances, X):
2     # Log Likelihood estimation
3     #
4     # INPUT:
5     # means            : Mean for each Gaussian KxD
6     # weights          : Weight vector 1xK for K Gaussians
7     # covariances      : Covariance matrices for each gaussian DxDxK
8     # X                : Input data NxD
9     # where N is number of data points
10    # D is the dimension of the data points
11    # K is number of gaussians
12    #
13    # OUTPUT:
14    # logLikelihood  : log-likelihood
15    return logLikelihood
```

(b) Implement the function `EStep` that performs the expectation step of the EM algorithm, **(2 pts)**
i.e. computes the responsibilities $\gamma_j(\mathbf{x}_n)$:

```
1 def EStep(means, covariances, weights, X):
2     # Expectation step of the EM Algorithm
3     #
```

```
4      # INPUT:
5      # means           : Mean for each Gaussian KxD
6      # weights         : Weight vector 1xK for K Gaussians
7      # covariances     : Covariance matrices for each gaussian DxDxK
8      # X               : Input data NxD
9      #
10     # N is number of data points
11     # D is the dimension of the data points
12     # K is number of gaussians
13     #
14     # OUTPUT:
15     # logLikelihood  : Log-likelihood (a scalar).
16     # gamma           : NxK matrix of responsibilities for N
          datapoints and K gaussians.
17     return [logLikelihood, gamma]
```

(c) Implement the function `MStep` that performs the maximization step of the EM algo-   **(2 pts)**
    rithm. i.e. $\hat{\pi}_j^{new}, \hat{\boldsymbol{\mu}}_j^{new}, \hat{\boldsymbol{\Sigma}}_j^{new}$:

```
1  def MStep(gamma, X):
2      # Maximization step of the EM Algorithm
3      #
4      # INPUT:
5      # gamma           : NxK matrix of responsibilities for N
          datapoints and K gaussians.
6      # X               : Input data (NxD matrix for N datapoints of
          dimension D).
7      #
8      # N is number of data points
9      # D is the dimension of the data points
10     # K is number of gaussians
11     #
12     # OUTPUT:
13     # logLikelihood  : Log-likelihood (a scalar).
14     # means           : Mean for each gaussian (KxD).
15     # weights         : Vector of weights of each gaussian (1xK).
16     # covariances     : Covariance matrices for each component(
          DxDxK).
17     return [weights, means, covariances, logLikelihood]
```

(d) As mentioned in the lecture, it is often necessary to introduce a regularization for   **(1 pt)**
    EM to work robustly. One possibility is to add a small value to the diagonal entries
    of all covariance matrices: $\boldsymbol{\Sigma}_{\text{reg}} = \boldsymbol{\Sigma} + \varepsilon\mathbf{I}$. This ensures that the covariance matrix
    has a low condition number, which makes the computation of the inverse more stable.
    Implement the function

```
1  def regularize_cov(covariance, epsilon):
2      # regularize a covariance matrix, by enforcing a minimum
3      # value on its singular values. Explanation see exercise sheet
4      #
5      # INPUT:
6      # covariance      : matrix
7      # epsilon         : minimum value for singular values
8      #
9      # OUTPUT:
```

```
10        # regularized_cov: reconstructed matrix
11        return regularized_cov
```

that regularizes a covariance matrix as described above.

(e) Implement the function `estGaussMixEM` for performing EM for estimating Gaussian **(1 pt)**
Mixture Models of D-dimensional data.

```
1  def estGaussMixEM(data, K, n_iters, epsilon):
2      # EM algorithm for estimation gaussian mixture mode
3      #
4      # INPUT:
5      # data             : input data, N observations, D dimensional
6      # K                : number of mixture components (modes)
7      #
8      # OUTPUT:
9      # weights          : mixture weights - P(j) from lecture
10     # means            : means of gaussians
11     # covariances      : covariancesariance matrices of gaussians
12     return [weights, means, covariances]
```

Use the functions from the previous subquestions for the implementation of this function. Initialize the weights uniformly

```
1  weights = numpy.ones(K) / K
```

and the means and the covariances using the K-Means algorithm:

```
1    kmeans = KMeans(n_clusters = K, n_init = 10).fit(data)
2      cluster_idx = kmeans.labels_
3      means = kmeans.cluster_centers_
4
5      # Create initial covariance matrices
6    for j in range(K):
7        data_cluster = data[cluster_idx == j]
8        min_dist = np.inf
9        for i in range(K):
10           # compute sum of distances in cluster
11           dist = np.mean(euclidean_distances(data_cluster, [
                 means[i]], squared=True))
12           if dist < min_dist:
13               min_dist = dist
14       covariances[:, :, j] = np.eye(n_dim) * min_dist
```

(f) Test your EM implementation on the provided synthetic data `data1`, `data2`, `data3` **(2 pts)**
and visualize your fitted model. Submit screen-shots of the visualizations. They should
look similar to this:

Now, repeat the experiment for different number of mixture models (e.g. $K = 4, 5, 6, \ldots$
etc). Examine how the likelihood of the computed model changes with K. Are these
likelihoods comparable? Which is the best K and why?

(g) Use the provided datasets of RGB pixel values for skin (`sdata`) and non-skin (`ndata`) **(2 bonus)**
regions for a skin detection experiment. First, train a Gaussian Mixture Model for each
dataset using your implementation from previous part. Based on these two Mixture
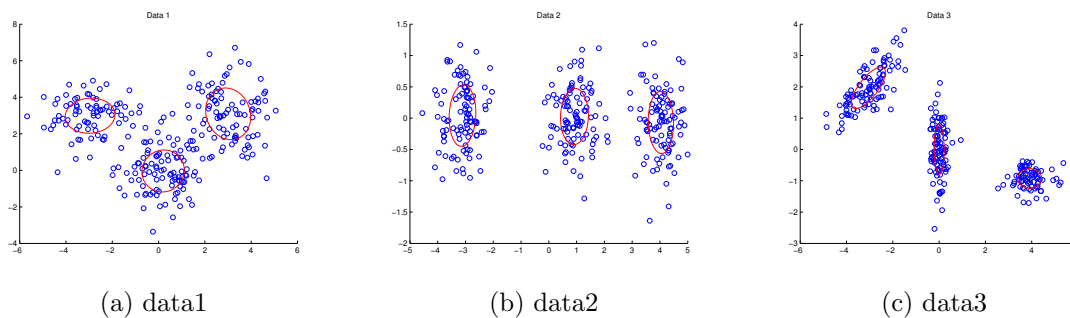Models, classify the pixels in the provided image `faces.png` according to the *Likeli-*

(a) data1   (b) data2   (c) data3

Figure 1: This figure shows the visualisation for the fitted models.

hood Ratio:

$$\frac{p(\text{color}|\text{skin})}{p(\text{color}|\text{non-skin})} > \theta$$

Implement this in the function `skinDetection`. Use different θ and compare your results to the groundtruth `faces_groundtruth.png`. Which parameters θ, `K`, etc. work well? Give possible explanations.

(h) Try your algorithm on some images. Can you find examples where it works or does **(2 bonus)** not work? Give some possible explanations.