Ali Athar <athar@vision.rwth-aachen.de>
Sabarinath Mahadevan <mahadevan@vision.rwth-aachen.de>

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

due **before** 2019-10-24

Important information regarding the exercises:

- The exercises are not mandatory. Still, we strongly encourage you to solve them! All submissions will be corrected. If you submit your solution, please read on:
- Send your submission via mail with the subject `[ML] exercise solution 1` to <**athar@vision.rwth-aachen.de**> and <**mahadevan@vision.rwth-aachen.de**>.
- Due to the large number of participants, we require you to submit your solution to L^{2}P **in groups of 3 to 4 students**. You can use the **Discussion Forum** on L^{2}P to organize groups.
- If applicable submit your code solution as a zip/tar.gz file named `mn1_mn2_mn3.{zip/tar.gz}` with your **matriculation numbers** (`mn`).
- Please do **not** include the data files in your submission!
- Please upload your pen & paper problems as PDF. Alternatively, you can also take pictures (.png or .jpeg) of your hand written solutions. Please make sure your handwriting is legible, the pictures are not blurred and taken under appropriate lighting conditions. All non-readable submissions will be discarded immediately.

Question 1: Python Tutorial ..... $(\Sigma = 0)$

To goal of this task is to get familiar with Python. **Please do not hand in a solution for this task.** 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 by following the instructions given in `https://scipy.org/install.html`.

(d) Walk through the basic numpy tutorial (`https://docs.scipy.org/doc/numpy-1.15.1/user/quickstart.html`).

Note: We suggest you to use the PyCharm IDE (`https://www.jetbrains.com/pycharm/`) for writing python code.

Question 2: Warming up! ..... $(\Sigma = 2)$

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? **(1 pt)**

(b) If we observe that the selected fruit is in fact an orange, what is the probability that it came from the green box? **(1 pt)**

Question 3: Minimizing the Expected Loss ..... $(\Sigma = 4)$

In classification problems a sample x can be assigned to one of the N classes in $C = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_N\}$. If the class is unrecognizable it is also possible to assign neither of the classes, which is also known as rejection $\mathcal{C}_{\text{rej}}$. When the rejection risk is lower than the risk of assignment to each class $\mathcal{C}_k \in C$, rejection can be a desirable action. Let the true class of a sample x be $\mathcal{C}_k$, and the class it is assigned to during the classification be $\mathcal{C}_j$. For any $k \in \{1, 2, ..., N\}$ and $j \in \{1, 2, ..., N+1\}$, the loss matrix is defined as

$$L_{kj} = \begin{cases} 0, & \text{if } k = j \\ l_r, & \text{if } j = N+1 \\ l_s, & \text{otherwise} \end{cases}$$

where l_r is the loss incurred for choosing the $(N+1)^{th}$ action (rejection) and l_s is the loss incurred for making a substitution error.

(a) Show that the minimum risk (expected loss) is obtained when: **(2 pts)**

$$\text{classify}(x) :\rightarrow \begin{cases} \mathcal{C}_j, & \text{iff } \forall k : p(\mathcal{C}_j|x) \geq p(\mathcal{C}_k|x) \wedge p(\mathcal{C}_j|x) \geq 1 - \frac{l_r}{l_s} \\ \mathcal{C}_{\text{rej}}, & \text{otherwise} \end{cases}$$

(b) What happens if $l_r = 0$? **(1 pt)**

(c) What happens if $l_r > l_s$? **(1 pt)**

Question 4: Maximum Likelihood ..... $(\Sigma = 4)$

Consider the following function

$$p(x|\theta) = \theta^2 x \exp(-\theta x) g(x)$$

where $g(x)$ is the unit step function with

$$g(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

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

Question 5: Kernel / K-Nearest Neighborhood Density Estimators .. $(\Sigma = 4)$

In this exercise you will have to write functions which estimate the probability density function (pdf) of a given dataset of 100 data points (represented by a 100 dimensional vector). For representing the (continuous) pdf in discrete form in Python, create a linearly spaced vector using the command:

```
1 pos = numpy.arange(-5, 5.0, 0.1) # Returns a 100 dimensional vector
```

inside both the `kde` and the `knn` functions.

The output variable `estDensity` should be a matrix with 2 columns:

- The first column of the matrix will be the linearly spaced vector `pos` you have created.
- The second column contains the density values estimated at the corresponding positions in the first column.

You can then use the provided `apply` functions to visualize your results.

Estimate the pdf of the data samples using:

(a) a kernel density estimation method with a Gaussian kernel with standard deviation h in `kde`. **(2 pts)**

```
def kde(samples, h):
    # compute density estimation from samples with KDE
    # Input
    # samples    : DxN matrix of data points
    # h          : (half) window size/radius of kernel
    # Output
    # estDensity : estimated density in the range of [-5,5]
    return estDensity
```

(b) a `k`-Nearest Neighbor density estimator in `knn`. **(2 pts)**

```
def knn(samples, k):
    # compute density estimation from samples with KNN
    # Input
    # samples    : DxN matrix of data points
    # k          : number of neighbors
    # Output
    # estDensity : estimated density in the range of [-5, 5]
    return estDensity
```

Question 6: Expectation Maximization (EM) Algorithm ..... $(\mathbf{\Sigma = 10 + 4})$

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).

(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

```
def getLogLikelihood(means, weights, covariances, X):
    # Log Likelihood estimation
    #
    # INPUT:
    # means          : Mean for each Gaussian KxD
    # weights        : Weight vector 1xK for K Gaussians
    # covariances    : Covariance matrices for each gaussian DxDxK
    # X              : Input data NxD
    # where N is number of data points
    # D is the dimension of the data points
    # K is number of gaussians
    #
    # OUTPUT:
    # logLikelihood  : log-likelihood
```

```
15      return logLikelihood
```

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

```
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 algorithm. i.e. $\hat{\pi}_j^{new}, \hat{\boldsymbol{\mu}}_j^{new}, \hat{\boldsymbol{\Sigma}}_j^{new}$: **(2 pts)**

```
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 EM to work robustly. One possibility is to add a small value to the diagonal entries of all covariance matrices: $\boldsymbol{\Sigma}_{\text{reg}} = \boldsymbol{\Sigma} + \epsilon\mathbf{I}$. This ensures that the covariance matrix has a low condition number, which makes the computation of the inverse more stable. Implement the function **(1 pt)**

```
1  def regularize_cov(covariance, epsilon):
```

```
    # regularize a covariance matrix, by enforcing a minimum
    # value on its singular values. Explanation see exercise sheet
    #
    # INPUT:
    # covariance     : matrix
    # epsilon        : minimum value for singular values
    #
    # OUTPUT:
    # regularized_cov: reconstructed matrix
    return regularized_cov
```

that regularizes a covariance matrix as described above.

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

```
def estGaussMixEM(data, K, n_iters, epsilon):
    # EM algorithm for estimation gaussian mixture mode
    #
    # INPUT:
    # data           : input data, N observations, D dimensional
    # K              : number of mixture components (modes)
    #
    # OUTPUT:
    # weights        : mixture weights - P(j) from lecture
    # means          : means of gaussians
    # covariances    : covariancesariance matrices of gaussians
    # logLikelihood  : log-likelihood of the data given the model
    return [weights, means, covariances]
```

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

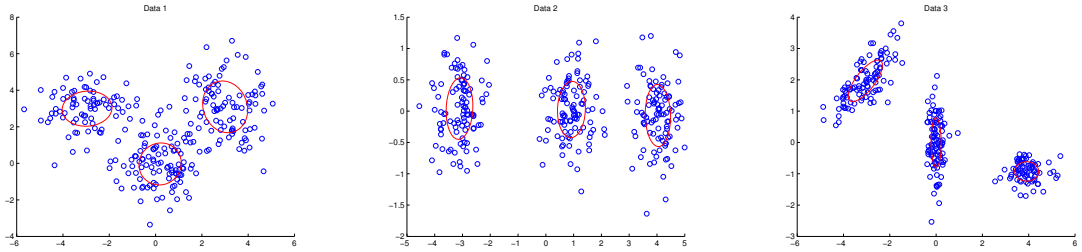
```
weights = numpy.ones(K) / K
```

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

```
  kmeans = KMeans(n_clusters = K, n_init = 10).fit(data)
    cluster_idx = kmeans.labels_
    means = kmeans.cluster_centers_

    # Create initial covariance matrices
    for j in range(K):
        data_cluster = data[cluster_idx == j]
        min_dist = np.inf
        for i in range(K):
            # compute sum of distances in cluster
            dist = np.mean(euclidean_distances(data_cluster, [
                means[i]], squared=True))
            if dist < min_dist:
                min_dist = dist
        covariances[:, :, j] = np.eye(n_dim) * min_dist
```

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



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

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

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)` 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 *Likelihood Ratio:* **(2 bonus)**

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

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 not work? Give some possible explanations. **(2 bonus)**

Optionally, you can capture images from your webcam directly for testing purposes. Have a look at the OpenCV library (`https://opencv.org/`) for more details on this.