Exercises
Computational Intelligence Lab
SS 2011

Machine Learning Laboratory
Dept. of Computer Science, ETH Zürich
Prof. Dr. Joachim M. Buhmann
Web http://ml2.inf.ethz.ch/courses/cil

Series 5, Mar 31th, 2011 (Mixture Models and Sampling)

Problem 1 (Rejection Sampling):

Consider Rejection sampling and answer the following questions:

1. What is the distribution over original values of x?
2. For the sample x drawn from this distribution, what is the probability to be accepted?
3. According to the previous parts, what is the probability that a sample is accepted?
4. Discuss what happens if c is chosen too small or too large.

Problem 2 (GMM for Image Compression):

This assignment is concerned with the application of Gaussian mixture models to image compression. The exercise uses the dataset and setup as in Series 1, so you should already feel comfortable with the application. Here we use the Gaussian Mixture Model (GMM) algorithm to compress the image.

Again, we begin with setting up the environment on your local machine, then we go step by step through the EM algorithm.

Submission system environment setup:

1. The web page for the image compression application can be found here:

   http://cil.inf.ethz.ch/applications/compression.

2. Download the provided templates for compression and decompression functions, the evaluation script and the training data into a local folder.
3. Unzip the training images into the *same* folder.
4. Try running the evaluation script `EvaluateCompression.m` – it should output two numbers: 0 1.0001. These are the mean squared error and the compression rate of the template compression/decompression scripts.

The `Compress.m` script defines a function `I_comp = Compress(I)`, which takes an image `I` as an input and outputs a structure `I_comp` that contains a compressed version of `I`. The template provided simply stores the input image in the field of the output structure (providing no compression, whatsoever). Please place your compression code in this script.

Open the `Decompress.m` script. It defines a function `I_rec = Decompress(I_comp)`, which takes as its input the output of the `Compress` function, and returns the decompressed image. Please place your decompression code in this script.

To submit your solution to the online evaluation system, we require you to prepare a ".zip" file containing:

1. A function file called `Compress.m` which implements

   `I_comp = Compress(I),`

   which takes a color image scaled to $[0, 1]$ as an input and outputs a structure `I_comp` (the design of this structure is up to you).

2. A function file called `Decompress.m` which implements

   `I_rec = Decompress(I_comp),`

   which takes a structure returned by `Compress` as its input and returns a reconstructed image.

Your submission is evaluated according to the following criteria:

1. Compression ratio: size of the `I_comp` structure in bytes, relative to the size of original image `I`.
2. Approximation quality: mean squared error of the reconstructed image `I_rec` w.r.t. the original image `I`.
3. Run time of both the compression and decompression functions.

Representation of images:

The image is represented as a matrix $M_1 \times M_2$ of pixel values. If the image has color, then there is a third dimension, representing color components – red, green and blue (RGB) – and the matrix is three dimensional now $M_1 \times M_2 \times 3$.

Clustering for image compression:

The pipeline for a given image is:

1. Perform the feature extraction on the image.
2. Perform GMM clustering on the feature vectors.
3. Present the compressed image, decompress the image and visualize the quality of the compression.

1. **Feature Extraction**
   Write a function

   ```
   X = extract(I),
   ```

   which transforms a color or grayscale image from either an $M_1 \times M_2 \times 3$ (or $M_1 \times M_2$) array to a $3 \times N$ (or $1 \times N$) matrix with $N = M_1 \cdot M_2$. The commands `reshape` and `shifdim` might prove to be useful.

2. **GMM Clustering**
   Given the data set `X` of D features and N samples, we now compute a clustering of the data. Using the template from the website write a function

   ```
   [z, U, loglike] = gmm(X, nClusters),
   ```

   that returns the mean parameters `U` of the mixture model and the assignments `z` of the GMM clustering. Furthermore the log-likelihood `loglike` should be returned. The parameter `nClusters` stands for the number of clusters used, which you should choose manually. We give some more remarks about the implementation of the GMM at the end of the assignment description.

3. **Image Compression**
   Write a function, which uses `extract(I)` and `gmm(X, nClusters)` to compress an image:

   ```
   [I_comp] = Compress(I, nClusters),
   ```

   where `I_comp` is a structure containing everything that is needed for reconstructing the original image. The size of this structure will be evaluated by the submission system.

4. **Image Reconstruction**
   Write a function to reconstruct the image:

   ```
   [I_rec] = Decompress(I_comp),
   ```

   `I_rec` should be an image with the same dimensions as the original image in double precision in the $[0, 1]$ interval. Notice, that for the assignment of pixels to clusters you do not need double precision.

Possible extensions:

All of the extensions suggested for the K-means case, can be applied to this problem, too. The following lists some further possible extensions:

1. Propose some heuristics to avoid the singularity problem in the mixture of Gaussians.
2. As introduced in the lecture, you can use the K-means to initialize the GMM algorithm. This approach can overcome initialization problems that may occur in the GMM.
3. In the algorithm you could also estimate the covariance matrix from the data. The update would be given by

   $$\boldsymbol{\Sigma}_k^{\mathrm{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{k,n})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\mathrm{new}})(\mathbf{x}_n - \boldsymbol{\mu}_k^{\mathrm{new}})^T,$$

   where we use the notation from the lecture slides.

Remarks about the implementation of the GMM: As in the previous week with the K-means algorithm, we provide you with a fairly complete template of the GMM implementation. The only part missing is the implementation of the M-step. In the M-step the model parameters are reestimated based on the responsibilities. Please also study the E-step where the responsiblities $\gamma(z_{k,n})$ are computed. The responsibility $\gamma(z_{k,n})$ is the probability of a data point $\mathbf{x}_n$ being assigned to the k-th cluster. You can use the script `demoGMM.m` in order to test your GMM implementation on synthetic data. Note that you can modify the parameters of the GMM method to gain a better result.