DATA12001 Advanced Course in Machine Learning

Exercise 4 **Due April 23, 23:59**

In this exercise set we focus on unsupervised learning (lecture slides 8–10).

Submit pen&paper (as JPG or PDF) and application exercises (as PDF) to Moodle. Submit TMC exercises to TMC.

---

1 Gaussian mixture model with constraints

pen & paper, 4p

Consider the Gaussian Mixture Model (GMM) for $\mathbf{x} \in \mathbb{R}^D$

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad .$$

We will consider EM-maximization given N data points $x_1, \ldots, x_n$. The M-step maximizes

$$Q(\theta) = \sum_{n,k} r_{nk} \log(\pi_k p(\mathbf{x}_n \mid \theta_k)) = \sum_{n,k} r_{nk} \left(\log \pi_k - \frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k) - \frac{D}{2}\log(2\pi) - \frac{1}{2}\log\det \boldsymbol{\Sigma}_k \right),$$

where r_{nk} are $p(z_n = k \mid \mathbf{x}_n)$ are the probabilities of x_n belonging to kth cluster (computed using the model parameters from the previous round).

To find the optimal $\boldsymbol{\mu}_k$, the gradient

$$\nabla_{\boldsymbol{\mu}_k} Q = -\boldsymbol{\Sigma}_k^{-1} \sum_n r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k)$$

must be 0. This is only possible if $\boldsymbol{\mu}_k = \frac{1}{\sum_n r_{nk}} \sum_n r_{nk}\mathbf{x}_n$.

Finding optimal $\boldsymbol{\Sigma}_k$ is trickier and we will use Matrix cookbook here (see link in Moodle). We will use Eqs. 57 and 61 in Matrix cookbook and the fact that $\boldsymbol{\Sigma}$ is symmetric. Consequently, the gradient is equal to

$$\nabla_{\boldsymbol{\Sigma}_k} Q = \frac{1}{2} \sum_n r_{nk} \left(\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1} \right) \quad .$$

Now, $\nabla_{\boldsymbol{\Sigma}_k} Q = 0$ implies $\boldsymbol{\Sigma}_k = \frac{1}{\sum_n r_{nk}} \sum_n r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$.

The optimal priors π_k are given in the slides.

Here the covariances $\boldsymbol{\Sigma}_k$ are not constrained in any way – each cluster has its own covariance that is of full rank. We consider three alternatives for parameterization of the covariances:

(1) $\boldsymbol{\Sigma}_k = \mathrm{diag}(\mathbf{s_k})$ (diagonal covariances)

(2) $\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}$ (shared covariance for all clusters)

(3) $\boldsymbol{\Sigma}_k = \sigma^2 \mathbf{I}$ (shared spherical covariance for all clusters)

where $\mathbf{s_k} \in \mathbb{R}^{\mathbf{D}}$ is a vector of variances.

Derive the update rules (one part of the M-step) for the $\mathbf{\Sigma}$ parameters for each of the three cases. Does the parameterization of $\mathbf{\Sigma}$ have effect on the other updates of M-step (π and $\boldsymbol{\mu}_k$)?

2 Expectation Maximization

TMC, 5p

Complete the EM algorithm for Gaussian components (TMC exercise: `part4/01.em`).

Implement the following missing parts:

(a) `logsumrows(X)`, computes the sums of *rows* of an array $\mathbf{X}$. Both $\mathbf{X}$ and the output is in the log representation. In order to avoid overflow problems, a log-trick must be used, that is, if

$$s = \sum_i a_i$$

then

$$\log s = \log(\sum_i \exp(M + \log a_i)) - M$$

for any M. To avoid overflows, set M to be $-\max_i \log a_i$. Note that this needs to be done for every row of X.

(b) `computeresponsibilities(X, prior, mu, C)`, computes the probabilities $p(z_i = j \mid \mathbf{x}_i)$, that is the probability of $\mathbf{x}_i$ of belonging to jth cluster. In order to compute this, first use Bayes' formula

$$p(z_i = j \mid \mathbf{x}_i) = \frac{p(z_i = j)p(\mathbf{x}_i \mid z_i = j)}{p(\mathbf{x}_i)}.$$

Note that the normalization term is equal to

$$p(\mathbf{x}_i) = \sum_j p(z_i = j, \mathbf{x}_i) = \sum_j p(z_i = j)p(\mathbf{x}_i \mid z_i = j).$$

Also, the term $p(\mathbf{x}_i \mid z_i = j)$ is the Gaussian pdf,

$$p(\mathbf{x}_i \mid z_i = j) = \frac{1}{\sqrt{(2\pi)^m \det \mathbf{C}_j}} \exp(-1/2(\mathbf{x}_i - \boldsymbol{\mu}_j)^T \mathbf{C}_j^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_j))$$

In order to avoid overflow issues you should first compute the probabilities in the log-representation

$$\log p(z_i = j \mid \mathbf{x}_i) = \log p(z_i = j) + \log p(\mathbf{x}_i \mid z_i = j) - \log p(\mathbf{x}_i)$$

and the exponentiate the quantities. This should be done by first computing an array `L`, where `L[i, j]` is $\log p(z_i = j) + \log p(\mathbf{x}_i \mid z_i = j)$. Then we can use `logsumrows(L)` in order to obtain $\log p(\mathbf{x}_i)$.

(c) `computeparameters(R, X)`, computes the priors π, means $\boldsymbol{\mu}$, and the covariance matrices $\mathbf{C}$ for each component from the data $\mathbf{X}$ and probabilities $p(z_i = j \mid \mathbf{x}_i)$, which are given in the array `R`. Hint: You can use `np.cov` with `aweights` and `ddof = 0` to compute the weighted covariance matrix.

(d) `computeparametersdiagonal(R, X)`, computes the priors π, means $\boldsymbol{\mu}$, and the diagonal covariance matrices $\mathbf{C}$ for each component from the data $\mathbf{X}$ and probabilities $p(z_i = j \mid \mathbf{x}_i)$, which are given in the array `R`. Here, the model requires that the covariance matrices are diagonal.

(e) `computeparameterssame(R, X)`, computes the priors π, means $\boldsymbol{\mu}$, and the joint covariance matrix C for each component from the data $\mathbf{X}$ and probabilities $p(z_i = j \mid \mathbf{x}_i)$, which are given in the array `R`. Here, the model requires that the covariance matrices are all equal.

(f) `computeparametersspherical(R, X)`, computes the priors π, means $\boldsymbol{\mu}$, and the joint covariance matrix C for each component from the data $\mathbf{X}$ and probabilities $p(z_i = j \mid \mathbf{x}_i)$, which are given in the array `R`. Here, the model requires that the covariance matrices are all equal, and a shape of $\sigma\mathbf{I}$.

(g) `em(X, R, itercnt, stats)`, the main loop of the EM algorithm. The loop should run `itercnt` times, first calling the function `stats` in order to compute the parameters based on the probabilities stored in `R`, and then recomputing `R` using the parameters.

Hints: This is a long exercise with many parts. It is recommended to do them in the given order; the automated tests will test the individual parts separately. Keep in mind that the data vectors are rows in the data matrix.

Submit your final code in TMC.

3 Two gaussians

application, 2p

Use the code from the previous exercise to model the given in `2gaussians.txt`. Use two clusters, iterate 100 times, and initiate R with random.

Model the data with the standard mixture model (`computeparameters`) and the model with equal covariance matrices (`computeparameterssame`).

Construct two scatter plots, and use the color to indicate the cluster membership (the function `scatter` has a named parameter `c` that can be used to assign the color to a point). Don't use discretized memberships, use responsibilities.

Explain shortly what and why is happening in the plots.

Include the figures and the code, and submit your answer to moodle.

4 Alternate Least Squares

TMC, 5p

Complete the ALS algorithm (TMC exercise: `part4/02.als`).

The description of the algorithm is given in the lecture slides.

Implement the following missing parts:

(a) `error(X, W, H, reg)`, computes the regularized loss,

$$\mathcal{L} = \|\mathbf{X} - \mathbf{W}\mathbf{H}\|^2 + \lambda \|\mathbf{W}\|^2 + \lambda \|\mathbf{H}\|^2,$$

where the first norm ignores all the missing entries in $\mathbf{X}$. *Hint*: use `~np.isnan(X)` to find indices of all entries in $\mathbf{X}$ that are present.

(b) `solve(X, W, reg)`, finds H that minimizes

$$\mathcal{L} = \|\mathbf{X} - \mathbf{WH}\|^2 + \lambda \|\mathbf{H}\|^2 ,$$

where the first norm ignores all the missing entries in $\mathbf{X}$.

To find $\mathbf{H}$, let $\mathbf{x}_i$ be the ith column of $\mathbf{X}$ containing only the non-missing entries. Write $\mathbf{W}_i$ to be a sub-matrix containing only the rows that appear in $\mathbf{x}_i$. Let $\mathbf{h}_i$ be the ith column of $\mathbf{H}$. Then we can decompose the sum as

$$\mathcal{L} = \sum_i \|\mathbf{x}_i - \mathbf{W}_i\mathbf{h}_i\|^2 + \lambda \|\mathbf{h}_i\|^2 \quad .$$

This is a standard regularized least squares problem, and it has an analytic solution

$$\mathbf{h}_i = (\mathbf{W}_i^T\mathbf{W}_i + \lambda\mathbf{I})^{-1}\mathbf{W}_i^T\mathbf{x}_i \quad .$$

(c) `als(X, W, reg, itercnt)`, the main loop of the algorithm.

The algorithm should loop `itercnt` times, update *first* $\mathbf{H}$ and then update $\mathbf{W}$ using the new $\mathbf{H}$. In both cases use `solve`.

The algorithm should return the new $\mathbf{W}$ and $\mathbf{H}$, and an error array `err` of length `itercnt`. The value `err[i]` is the error after $i + 1$ iterations.

Hint: When solving W you need to transpose the matrices.

Hints: This is a long exercise with many parts. It is recommended to do them in the given order; the automated tests will test the individual parts separately.

Submit your final code in TMC.

5 Non-negative Matrix Factorization

TMC, 5p

Implement the NMF algorithm (TMC exercise: `part4/03.nmf`).

The description of the algorithm is given in the lecture slides.

The function `nmf(X, W, H, itercnt)` is given the data matrix $\mathbf{X}$, and initial factor matrices $\mathbf{W}$ and $\mathbf{H}$.

The algorithm should loop `itercnt` times, *first* update H and then update $\mathbf{W}$ using the new $\mathbf{H}$.

The algorithm should return the new $\mathbf{W}$ and $\mathbf{H}$, and an error array `err` of length `itercnt` + 1. The value `err[i]` is the L_2 error between $\mathbf{X}$ and the factorization $\mathbf{WH}$ after ith iteration. The 0th entry `err[0]` is the error using the initial $\mathbf{W}$ and $\mathbf{H}$.

Submit your final code in TMC.

6 Sammon Projection

TMC, 5p

Implement the SammonProjection algorithm (TMC exercise: `part4/04.sammon`).

Given a distance matrix $\mathbf{d}'$, Sammon projection seeks points X such minimizing

$$\mathcal{L}(X) = \frac{1}{\sum_{i<j} d'_{ij}} \sum_{i<j} \frac{(d_{ij} - d'_{ij})^2}{d'_{ij}},$$

where $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$.

To find the points, the algorithm performs a gradient descent. The initial points are typically obtained with PCA / Classical MDS.

The initialization is already provided in the template but the gradient descent is missing.

The gradient of $\mathcal{L}$ can be written as

$$\nabla \mathcal{L} = \mathbf{C}\mathbf{X},$$

where $\mathbf{C}$ can be expressed using only distances $\mathbf{d}'$ and $\mathbf{d}$.

To find $\mathbf{C}$ use the chain rule,

$$\frac{\partial \mathcal{L}}{\partial x_{ik}} = \sum_{j \neq i} \frac{\partial \mathcal{L}}{\partial d_{ji}} \frac{\partial d_{ji}}{\partial x_{ik}}$$

and re-arrange the terms.

As a gradient step during $r+1$th iteration, use $\frac{\eta}{1+\tau\eta r}$. That is, during the first iteration the step size should be η. In summary, the new value for $\mathbf{X}$ should be

$$\mathbf{X} - \frac{\eta}{1 + \tau\eta r}\mathbf{C}\mathbf{X} \quad .$$

Hints: Use `euclidean_distances` to compute the distance matrix from $\mathbf{X}$. The matrix $\mathbf{C}$ will have different looking equations for the diagonal and non-diagonal entries.

Submit your final code in TMC.