MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.036—Introduction to Machine Learning
Spring 2016

Project 3: Rating movies with mixtures   Issued: Mon. 4/4 Due: Fri. 4/29 at 9am

Instructions: For this project, all of the skeleton functions that you will have to complete, as well as several other functions that we've provided for you, are in `project3.py`. You may wish to write helper functions. All other additional code that you will have to write to test and evaluate the methods explored in this project should be written in the relevant section of `main.py`.

For this project, we will make use of several Python packages. If you do not already have any of `numpy`, `scipy`, or `matplotlib` installed on your machine, please install them for this project.

Project Submission: Please submit two files—a *single* PDF file containing all your answers, code, and graphs, and a *second* .zip file called `project3.zip` containing `project3.py` and `main.py`, which should contain all of your code, to the Stellar web site by 9am, April 29th.

Validating your code is ready for submission: To check if your files and outputs are all in the right format for submission, please run the file `checker.py` with the your submission zip file. `checker.py` will use `toy_data.txt` to check your code, so please make sure that `toy_data.txt`, your submission zip file, and `checker.py` are all in the same folder.

Introduction

Your task is to build a mixture model for collaborative filtering. You are given a data matrix containing movie ratings made by users where the matrix is extracted from a much larger Netflix database. Any particular user has rated only a small fraction of the movies so the data matrix is only partially filled. The goal is to predict all the remaining entries of the matrix.

You will use mixtures of Gaussians to solve this problem. The model assumes that each user's rating profile is a sample from a mixture model. In other words, we have K possible types of users and, in the context of each user, we must sample a user type and then the rating profile from the Gaussian distribution associated with the type. We will use the Expectation Maximization (EM) algorithm to estimate such a mixture from a partially observed rating matrix. The EM algorithm proceeds by iteratively assigning (softly) users to types (E-step) and subsequently re-estimating the Gaussians associated with each type (M-step). Once we have the mixture, we can use it to predict values for all the missing entries in the data matrix.

1. **Part 1** Warm up (45 Points).

   For this part of the project you will compare clustering obtained via K-means to the (soft) clustering induced by EM. In order to do so, our K-means algorithm will differ a bit from the one you learned. Here, the means are estimated exactly as before but the algorithm returns additional information. More specifically, we use the resulting clusters of points to estimate a Gaussian model for each cluster. Thus, our K-means algorithm actually returns a mixture model where the means of the component Gaussians are the K centroids computed by the K-means algorithm. This is to make it such that we can now directly plot and compare solutions returned by the two algorithms as if they were both estimating mixtures.

(a) (5 Points) Read a 2D toy dataset using `X = readData('toy_data.txt')`. Your task is to run the K-means algorithm on this data using code `kMeans` we have provided. Initialize K-means by running `(Mu,P,Var) = init(X,K)` where K is the number of clusters. Note that `init(X,K)` returns a K-component mixture model with means, mixing proportions, and variances. The K-means algorithm will only care about the means, however, and returns a mixture that is retrofitted based on the K-means solution. Try $K = [1, 2, 3, 4]$ on this data, plotting each solution using our `plot2D` function. Since the initialization is random, make sure you try each K a few times to select the one that minimizes the total distortion cost (as explained in lecture). Submit the associated plots (best solution for each K). The code you used to do this should be written in the relevant section of `main.py`.

(b) (25 Points) Recall the Gaussian mixture model presented in class:

$$P(x|\theta) = \sum_{j=1}^{K} \pi_j N(x; \mu^{(j)}, \sigma_j^2 I),$$

where θ denotes all the parameters in the mixture (means $\mu^{(j)}$, mixing proportions π_j, and variances σ_j^2). The goal of the EM algorithm is to estimate these unknown parameters by maximizing the log-likelihood of the observed data $x^{(1)}, \ldots, x^{(n)}$. Starting with some initial guess of the unknown parameters, the algorithm iterates between E- and M-steps. The E-Step softly assigns each data point $x^{(i)}$ to mixture components. The M-step takes these soft-assignments as given and finds a new setting of the parameters by maximizing the log-likelihood of the weighted dataset (expected complete log-likelihood)

Implement the EM algorithm for the Gaussian mixture model desribed above. To this end, expand the skeleton functions `Estep`, `Mstep`, and `mixGauss` provided in `project3.py`. In our notation,

- X: an $n \times d$ Numpy array of n data points, each with d features
- K: number of mixture components
- M: $K \times d$ Numpy array where the j^{th} row is the mean vector $\mu^{(j)}$
- P: $K \times 1$ Numpy array of mixing proportions π_j, $j = 1, \ldots, K$
- Var: $K \times 1$ Numpy array of variances σ_j^2, $j = 1, \ldots, K$

The convergence criteria that you should use is that the improvement in the log-likelihood is less than or equal to 10^{-6} multiplied by the absolute value of the new log-likelihood. In slightly more algebraic notation: new log-likelihood $-$ old log-likelihood $\leq 10^{-6} \cdot |$new log-likelihood$|$

Your code will output updated versions of M, P, and Var, as well as an $n \times K$ Numpy array `post`, where post$[i, j]$ is the posterior probability $p(j|x^{(i)})$, and `LL` which is a list of the log-likelihood scores evaluated at the beginning of each EM iteration (returned by `Estep`). To validate that your `Estep` is correct, with $K = 3$, a single call to `Estep` after initializing using `(Mu,P,Var) = init(X,K,fixedmeans=True)` should return a log-likelihood of -1331.67489. Once you finish implementing `Mstep` and `mixGauss`, the next step below will provide an additional benchmark to test your full implementation to ensure that it's working correctly.

(c) Now that you have a working implementation of EM, let's check that it is indeed correct. First, EM should monotonically increase the log-likelihood of the data. Initialize and run the EM algorithm on the toy dataset as you did earlier with K-means. You should check that the `LL` values that the algorithm returns after each run are indeed always monotonically increasing (non-decreasing). As another validation, let's compare your algorithm to our implementation.

Set $K = 3$ and let `(Mu,P,Var) = init(X,K,fixedmeans=True)`. If you run the EM algorithm with this initialization, as we observed above you should get -1331.67489 as the initial log-likelihood. You should also get -1138.89248 as the final log-likelihood when the algorithm converges. Finally, as a runtime guideline, in your testing on the toy dataset, calls of `mixGauss` using the values of K that we are testing should run in on the order of seconds (i.e. if each call isn't fairly quick, that may be an indication that something is wrong). The code you used to do this should be written in the relevant section of `main.py`.

(d) (5 Points) Generate analogous plots to K-means using your EM implementation. Note that the EM algorithm can also get stuck in a locally optimal solution. For each value of K, you should run the EM algorithm multiple times with different initializations returned by `init` and select the solution that achieves the highest log-likelihood. Compare the K-means and mixture solutions for $K = [1, 2, 3, 4]$. Briefly explain when, how, and why they differ. The code you used to do this should be written in the relevant section of `main.py`.

(e) (10 Points) So far we have simply set the number of mixture components K but this is also a parameter that we must estimate from data. How does the log-likelihood of the data vary as a function of K assuming we avoid locally optimal solutions? To compensate, we need a selection criterion that penalizes the number of parameters used in the model. Fill in the missing Bayesian Information Criterion (BIC) calculation in `BICmix(X,Kset)`. Find the best K using `Kset=[1,2,3,4]` and the toy dataset (the code you used to do this should be written in the relevant section of `main.py`). This will be the K that produces the optimal BIC score (highest or lowest, depending on how you define BIC; e.g. it would be the maximal BIC score if you use the definition provided in HW5). Report the best K and the corresponding BIC score. Does the criterion select the correct number of clusters for the toy data?

2. **Part 2** Mixture models for matrix completion (55 Points)

We can now extend our Gaussian mixture model to predict actual movie ratings. Let X again denote the $n \times d$ data matrix. The rows of this matrix correspond to users and columns specify movies so that `X[u,i]` gives the rating value of user u for movie i (if available). Both n and d are typically quite large. The ratings range from one to five stars and are mapped to integers $\{1, 2, 3, 4, 5\}$. We will set `X[u,i]=0` whenever the entry is missing.

In a realistic setting, most of the entries of X are missing. For this reason, we define C_u as the set of movies (column indexes) that user u has rated and H_u as its complement (the set of remaining unwatched/unrated movies we wish to predict ratings for). From the point of view of our mixture model, each user u is an example $x^{(u)} =$ `X[u,:]`. But since most of the coordinates of $x^{(u)}$ are missing, we need to focus the model during training on just the observed portion. To this end, we use $x^{(u)}_{C_u} = \{x^{(u)}_i : i \in C_u\}$ as the vector of only observed ratings. If columns are indexed as $\{0, \dots, d-1\}$, then a user u with a rating vector $x^{(u)} = (5, 4, 0, 0, 2)$, where zeros indicate missing values, has $C_u = \{0, 1, 4\}$, $H_u = \{2, 4\}$, and $x^{(u)}_{C_u} = (5, 4, 2)$.

In this part, we will extend our mixture model in two key ways.

- First, we are going to estimate a mixture model based on partially observed ratings. We will be maximizing the marginal log-likelihood:

$$\ell(X; \theta) = \sum_{u=1}^{n} \log \left[\sum_{j=1}^{K} \pi_j N(x^{(u)}_{C_u} | \mu^{(j)}_{C_u}, \sigma_j^2 I_{|C_u| \times |C_u|}) \right] \quad (1)$$

$|C_u|$ is the number of observed rating values from user u. Not the differences from the log-likelihood that we maximize in the EM algorithm that was presented in class.

- Second, since we will be dealing with a large, high-dimensional data set, we will need to be more mindful of numerical underflow issues. To this end, you should perform most of your computations in the log domain. Remember, $\log(a \cdot b) = \log(a) + \log(b)$. This can be useful to remember when a and b are very small – in these cases, addition should result in fewer numerical underflow issues than multiplication.

  An additional numerical optimization trick that you will find useful is the LogSumExp trick. Assume that we wish to evaluate $y = \log(\exp(x_1) + ... \exp(x_n))$. We define $x* = \max\{x_1..x_n\}$. Then, $y = x * + \log(\exp(x_1 - x*) + ... \exp(x_n - x*))$. This is just another trick to help ensure numerical stability.

Let's get started:

(a) (5 Points) If $x^{(u)}$ were a complete rating vector, the mixture model from Part 1 would simply say that $P(x^{(u)}|\theta) = \sum_{j=1}^{K} \pi_j N(x^{(u)}; \mu^{(j)}, \sigma_j^2 I)$. In the presence of missing values, we must use the marginal probability $P(x_{C_u}^{(u)}|\theta)$ that is over only the observed values. This marginal corresponds to integrating the mixture density $P(x^{(u)}|\theta)$ over all the unobserved coordinate values. In our case, this marginal can be computed easily. Provide a brief explanation for why

$$P(x_{C_u}^{(u)}|\theta) = \sum_{j=1}^{K} \pi_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{|C_u| \times |C_u|}). \tag{2}$$

Hint: a multivariate spherical Gaussian is simply a product of univariate Gaussians pertaining to each coordinate. In other words, $N(x; \mu, \sigma^2 I) = \prod_i N(x_i; \mu_i, \sigma_i^2)$ where $N(x_i; \mu_i, \sigma_i^2)$ is just a univariate (1-dimensional) Gaussian distribution.

(b) We need to update our EM algorithm a bit to deal with the fact that the observations are no longer complete vectors. We use Bayes' rule to find an updated expression for the posterior probability $p(j|u) = P(y = j|x_{C_u}^{(u)})$:

$$p(j \mid u) = \frac{p(u|j) \cdot p(j)}{p(u)} = \frac{p(u|j) \cdot p(j)}{\sum_{j=1}^{K} p(u|j) \cdot p(j)} = \frac{p_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{C_u \times C_u})}{\sum_{j=1}^{K} p_j N(x_{C_u}^{(u)}; \mu_{C_u}^{(j)}, \sigma_j^2 I_{C_u \times C_u})}$$

This is the soft assignment of cluster u to data point j.

To minimize numerical instability, you will be re-implementing the E-step in the log-domain, so you should calculate the values for the log of the posterior probability, $\ell(j, u) = \log(p(j|u))$ (though the actual output of your E-step should include the non-log posterior).

Let $f(u, i) = \log(\pi_i) + \log\left(N(x_{C_u}^{(u)}; \mu_{C_u}^{(i)}, \sigma_i^2 I_{C_u \times C_u})\right)$. Then, in terms of f, the log posterior is:

$$\begin{aligned}
\ell(j|u) = \log(p(j \mid u)) = \log\left(\frac{p_j N(x^{(u)}_{C_u}; \mu^{(j)}_{C_u}, \sigma_j^2 I_{C_u \times C_u})}{\sum_{j=1}^K p_j N(x^{(u)}_{C_u}; \mu^{(j)}_{C_u}, \sigma_j^2 I_{C_u \times C_u})}\right) \\
= \log\left(p_j N(x^{(u)}_{C_u}; \mu^{(j)}_{C_u}, \sigma_j^2 I_{C_u \times C_u})\right) - \log\left(\sum_{j=1}^K p_j N(x^{(u)}_{C_u}; \mu^{(j)}_{C_u}, \sigma_j^2 I_{C_u \times C_u})\right) \\
= \log(p_i) + \log\left(N(x^{(u)}_{C_u}; \mu^{(i)}_{C_u}, \sigma_i^2 I_{C_u \times C_u})\right) - \log\left(\sum_{j=1}^K \exp(\log(p_j N(x^{(u)}_{C_u}; \mu^{(j)}_{C_u}, \sigma_j^2 I_{C_u \times C_u})))\right) \\
= f(u,j) - \log\left(\sum_{j=1}^K \exp(f(u,j))\right)
\end{aligned}$$

(Implementation hint: use the LogSumExp trick.)

(c) Once we have evaluated $p(j|u)$ in the E-step, we can proceed to the M-step. We wish to find the parameters π, μ, and σ that maximize $\ell(X;\theta)$, the expected complete log-likelihood:

$$\ell(X;\theta) = \sum_{u=1}^n \left[\sum_{j=1}^K p(j|u) \log\bigl(\pi_j N(x^{(u)}_{C_u} | \mu^{(j)}_{C_u}, \sigma_j^2 I_{|C_u| \times |C_u|})\bigr)\right],$$

To maximize $\ell(X;\theta)$, we keep $p(j|u)$ (the soft-assignments) fixed, and maximize over the model parameters. Some of the parameters can be updated exactly as before with complete example vectors. For example,

$$\hat{\pi}_j = \frac{\sum_{u=1}^n p(j|u)}{n}$$

But we must be more careful in updating $\mu^{(j)}$ and σ_j^2. This is because the parameters appear differently in the likelihood depending on how incomplete the observation is. Notice that some coordinates of $\mu^{(j)}$ do not impact observation $x^{(u)}_{C_u}$ at all. But we can proceed to separately update each coordinate of $\mu^{(j)}$.

The update equation for $\mu^{(j)}_i$ (the i^{th} coordinate of $\mu^{(j)}$) is derived as follows:

Decomposing the multivariate spherical Gaussians into univariate spherical Gaussians as before,

$$\begin{aligned}
\ell(X;\theta) &= \sum_{u=1}^n \left[\sum_{j=1}^K p(j|u) \log\bigl(p_j \prod_{i=1}^D \delta(i, C_u) N(x_i^{(u)} | \mu_i^{(j)}, \sigma^2_{i,(j)})\bigr)\right] \\
&= \sum_{u=1}^n \left[\sum_{j=1}^K p(j|u) \log\bigl(p_j \prod_{i=1}^D \delta(i, C_u) \frac{1}{\sqrt{2\pi}\sigma_{i,(j)}} \exp(-\frac{1}{2\sigma^2_{i,(j)}}(x_i^{(u)} - \mu_i^{(j)})^2)\bigr)\right]
\end{aligned}$$

where $\delta(i, C_u)$ is an indicator function: 1 if $i \in C_u$ and zero otherwise.

Transforming the log of a product into a sum of logs and differentiating with respect to the lth movie coordinate for cluster j yields

$$\begin{aligned}
\frac{\partial \ell(X;\theta)}{\partial \mu_l^{(j)}} &= \sum_{u=1}^{n} \left[p(j|u)\delta(l, C_u)(-\frac{1}{2\sigma^2_{l,(j)}}(-2x_l^{(u)} + 2\mu_l^{(j)})) \right] = 0 \\
0 &= \sum_{u=1}^{n} p(j|u)\delta(l, C_u)(x_l^{(u)} - \mu_l^{(j)}) \\
\hat{\mu}_l^{(j)} &= \frac{\sum_{u=1}^{n} \delta(l, C_u)p(j \mid t)x_l^{(t)}}{\sum_{u=1}^{n} \delta(l, C_u)p(j \mid t)}
\end{aligned}$$

We do **not** compute the mean update in the log domain; we use $p(j|u)$ instead of $\ell(j, u)$. When you set $\mu_i^{(j)}$ and σ_j^2 in the implementation, it will be easier, and not lead to numerical underflow issues, to use $p(j|u)$ instead of the logarithm $\ell(j, u)$.

Finally, the update equation for the variance is not too different from before:

$$\hat{\sigma}_j^2 = \frac{1}{\sum_{u=1}^{n} |C_u| p(j|u)} \sum_{u=1}^{n} p(j|u) \| x_{C_u}^{(u)} - \hat{\mu}_{C_u}^{(j)} \|^2$$

Implementation hints:

- You may find LogSumExp useful. But remember that your M-step should return the new $P = \hat{\pi}$, not the log of $\hat{\pi}$.
- The following will not affect the update equation above, but will affect your implementation: since we are dealing with incomplete data, we might have a case where most of the points in cluster j are missing the i-th coordinate. If we are not careful, the value of this coordinate in the mean will be determined by a small number of points, which leads to erratic results. Instead, we should only update the mean when $\sum_{u=1}^{n} p(j|u)\delta(i, C_u) \geq 1$. Since $p(j|u)$ is a soft probability assignment, this corresponds to the case when at least one full point supports the mean.
- To also avoid the variances of clusters going to zero due to a small number of points being assigned to them, in the M-step you will need to implement a minimum variance for your clusters. We recommend a value of 0.25, though you are free to experiment with it if you wish. Note that this issue, as well as the thresholded mean update in the point above, are better dealt with through regularization; however, to keep things simple, we do not do regularization here.

(d) (30 Points) Fill in the skeleton functions `Estep_part2` and `Mstep_part2` so that these versions work with partially observed vectors where missing values are indicated with zeros, and perform the computations in the log domain to help with numerical stability. Similarly, implement `mixGauss_part2` using `Estep_part2` and `Mstep_part2`. As in Part 1, the convergence criteria that you should use is that the improvement in the log-likelihood is less than or equal to 10^{-6} multiplied by the absolute value of the new log-likelihood.

To debug your EM implementation, you may use the script `matrixCompletionText.py` and the file containing expected outputs, `matrixCompletionTest_Solutions.txt`. To use `matrixCompletionText.py`, please replace the filename `project3` in the imports section at the top with the name of your project3 code file. The script runs your E and M steps for Part 2 separately on a test matrix completion data set (`matrixCompletionTest_incomplete.txt` and matrixCompletionTest_complete.txt), with a fixed initialization. The script also tests the mixGauss and fillMatrix functions and finds the Root Mean Squared Error between the filled matrix and the true complete matrix for this test dataset. The correct outputs for this test-case are provided in matrixCompletionTest_Solutions.txt. This test case if provided for you to debug your implementation – you do not need to turn in any of the code associated with this debugging. **From here on out, make sure you are calling the EM and mixture model functions you wrote in Part 2 (e.g. mixGauss_part2, Estep_part2, and Mstep_part2), and not those you wrote in Part 1!**

(e) (5 Points) To validate your mixture model implementation, estimate a Gaussian mixture with $K = 12$ from the incomplete data matrix. As before, you will need to try several different initializations, and select the best one. Report the log-likelihood value of the best mixture. To debug, you can try different values of K on the incomplete data X = readData('netflix_incomplete.txt') and make sure that the log-likelihood values are indeed monotone. If you set $K = 1$, the resulting log-likelihood of data should be -1521060.95399. This may take on the order of a couple of minutes to run with $K = 12$.

(f) (5 Points) Now that we have a mixture model, how do we use it to complete a partially observed rating matrix? Provide an expression for completing a particular row, say x_C where the observed values are $i \in C$.

(g) (10 Points) Expand the function fillMatrix that takes as input an incomplete data matrix X as well as a mixture model, and outputs a completed version of the matrix Xpred.You can test the accuracy of your predictions against actual target values by loading the complete matrix Xc = readData('netflix_complete.txt') and measuring the root mean squared error between the two matrices rmse(Xpred,Xc). Please use your best mixture for $K = 12$ from part (e) to generate all the results and provide it in your write up.