MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.036—Introduction to Machine Learning
Spring 2015

Project 3: Which movies do Gaussian likes? Issued: Thursday., 4/10 Due: Fri., 4/24 at 9am

Project Submission: Please submit two files—a *single* PDF file containing all your answers, code, and graphs, and a *second* .zip file containing all the code you wrote for this project, to the Stellar web site by 9am, April 25th.

Introduction

Your task is to build a mixture model for collaborative filtering. You are given a data matrix containing movie ratings made by users; we have sampled this matrix from the Netflix database. Not all movies have been rated by all users, and the goal of this project is to use mixture modeling to predict the missing ratings. You will explore this task by using the Expectation Maximization (EM) algorithm that uses the hidden structure across different users, and with the help of this hidden structure, you will be able to predict the missing ratings. In other words, we have a partially observed rating matrix, and we will use the EM algorithm to complete (fill in) the missing entries.

1. **Part 1** Warm up example.

   For this part of the project you will compare clustering obtained via K-means to the (soft) clustering induced by EM.

   (a) Use the toy data set ($toy\_data.txt$) and the K-means code (function **kMeans** in $project3\_student.py$) provided to plot different clusters for cluster sizes $K = [5, 10, 15]$. Notice that when using K-means, each data point is fully assigned to a single cluster, that is, each point can have only one cluster label.

   (b) Let x be a data point. Recall the mixture model presented in class:

   $$P(x|\theta) = \sum_{j=1}^{K} p(j|\theta)p(x|j,\theta),$$

   where θ denotes the parameters of the model. A data point may be generated by first choosing a cluster, and then choosing a data point x according to that cluster's distribution. Once you have learned the centroids and the clusters given by K-means, how would the data generation process based on K-means differ from the data generation process given by a mixture model. [Hint: K-means partitions the space into clusters, whereas a mixture model allows weighted memberships across different clusters].

   (c) Consider a mixture model that uses a Gaussian as the conditional distribution given the hidden label. That is, $p(x|j,\theta) = N(x|\mu^{(j)}, \sigma_j^2 I)$, where $\mu^{(j)}$ and $\sigma_j^2 I$ are the unknown parameters for mixture component of type j.

   The goal of the EM algorithm is to estimate these unknown parameters by making use of observed data, say $x^{(1)}, \ldots, x^{(N)}$. Starting with some initial guess at the unknown parameters, the E-Step keeps the model fixed (i.e., for each component j, its parameters $\mu^{(j)}$ and the σ_j^2 are held fixed; this idea is similar to K-means holding the centroids fixed in one of its steps), and computes the soft-assignments for each data point. Thus, for each data point $x^{(t)}$ $(1 \leq t \leq N)$ the

E-step computes the posterior probabilities $p(j|x^{(t)})$. The M-step takes these soft-assignments as fixed, and computes the maximum-likelihood estimates of the parameters $\mu^{(j)}$ and $\sigma_j^2 I$ for each component j ($1 \leq j \leq K$) (notice the analogy to K-means, which, given assignment of data points to their clusters, computes the centroids of each cluster).

Task: Implement the EM algorithm using a Gaussian mixture model (as recalled above). Write a python function **mixGauss** in $project3_student.py$. Your inputs should be

i. X: an $n \times d$ Numpy array of n data points, each with d features
ii. K: number of mixtures; Mu: K*d Numpy array, each row corresponds to a mixture mean vector;
iii. P: $K \times 1$ Numpy array, each entry corresponds to the weight for a mixture;
iv. Var: $K \times 1$ Numpy array, each entry corresponds to the variance for a mixture;

Your outputs should be

i. output: Mu: $K \times d$ matrix, each row corresponds to a mixture mean;
ii. P: $K \times 1$ Numpy array, each entry corresponds to the weight for a mixture;
iii. Var: $K \times 1$ Numpy array, each entry corresponds to the variance for a mixture;
iv. post: $n \times K$ Numpy array, each row corresponds to the soft counts for all mixtures for an example
v. LL: Numpy array, which records the loglikelihood value for each iteration.

(d) The EM algorithm has a chicken an egg problem. Before it can evaluate a better gaussian model, it needs soft-clustering assignments and for re-evaluating soft-clustering assignments it needs some new but fixed gaussian model to evaluate the soft-clusterings. Hence, we need to break this chicken and egg problem by providing some initialization for the parameters so that EM can start working.

Task: Think of some adverbial way to initialize the EM algorithm and **explain** why that initialization might not be infer the hidden labels in a good way.

(e) Now that you have considered an initialization for EM, run your implementation of the EM algorithm using **init** function we provided. Compute and report the log-likelihood of the parameters you learned after T=5, 10, 20, 50, 100 steps.

(f) Your next task is to understand how K-means clustering differs from the soft-clustering induced by learning a mixture model. Using the parameters estimated in part (c) and the function `plot2D`, **plot** the clusters each of the 2D Gaussians and include it in your write up. **Explain** why in the mixture model it does not make sense to deterministically assign any data point to a Gaussian and hence highlight its difference from K-means clustering. Do points that are very far away from cluster centroids still have a chance to be assigned to any cluster in EM? What about in K-means?

(g) Now we will try to choose the number of mixture components (K) that EM should learn. **Explain** why choosing a value of K that achieves the highest log-likelihood might not be the best criterion for selecting K.

(h) One way to avoid the issues addressed in part (e) is to penalize a high number of parameters. **Explain** how the Bayesian Information Criterion (BIC) addresses the issue brought up in part (e) and why it might be a better function for choosing K.

(i) Implement the Bayesian Information Criterion (BIC) for selecting the number of mixture components. Choose the best value of K from the choices $\{5, 10, 15, 20, 30\}$.
Write a python function `BICmix`. The inputs are:

i. X: an $n \times d$ Numpy array of n data points, each with d features
ii. P: $K \times 1$ Numpy array, each entry corresponds to the weight for a mixture;
iii. Var: $K \times 1$ Numpy array, each entry corresponds to the variance for a mixture;

The output should be:

i. K: number of mixtures; Mu: $K \times d$ Numpy array, each row corresponds to a mixture mean vector;

2. **Part 2** EM for predicting movie ratings via *matrix completion*.

In this part of the project we will use the EM algorithm for matrix completion. Let X denote the $N \times D$ data matrix. The rows of this matrix correspond to users and the columns correspond to movies. A single entry $x_j^{(i)}$ (or in matrix indexing notation x_{ij}) indicates the rating person user i gave to movie j, and this rating is a single number that lies in the set $\{1, 2, 3, 4, 5\}$.

In a realistic setting, most of the entries of X are missing, because a user will not have watched most of the movies so he/she would have not rated the unwatched movies. Thus, we use the set C_u to denote the collection of movies (column indices) that user u has rated. Also, let H_u denote the set of movie indices that a user has not watched. Notice that $C_u \cup H_u = \{1, \ldots, D\}$. To denote a subset of the movies a particular user has watched we write $x_{C_u}^{(u)}$, which is a vector with $|C_u|$ entries. Similarly $x_{H_u}^{(u)}$ denotes the vector of hidden / unknown entries (the ratings we wish to estimate).

For example, if user 1 has the ratings vector $x^{(1)} = (5, 4, ?, ?, 2)$, then $C_1 = \{1, 2, 5\}$ and $H_1 = \{3, 4\}$ and $x_{C_1}^{(1)} = (5, 4, 2)$.

Our goal is to use a mixture model to generate the missing entries of the matrix X (thus the name "matrix completion"). We will estimate the parameters of the mixture model using EM.

(a) The mixture model from Part 1 assigns the probability density $P(x^{(u)}|\theta) = \sum_{j=1}^{K} p_j N(x^{(u)}; \mu^{(j)}, \sigma_j^2 I)$ to the vector $x^{(u)}$. However, since we have missing entries, i.e., not all entries of $x^{(u)}$ are known, we will just use only the observed data and compute $P(x_{C_u}^{(u)}|\theta)$. **Argue** that the correct expression for $P(x_{C_u}^{(u)}|\theta)$ is:

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

Make sure to mention why the covariance matrix has an identity matrix $|C_u| \times |C_u|$ and not $d \times d$.

[Hint: note that the covariance matrix is a multiple of the identity].

(b) Now that you have a mixture model for each user, provide a possible interpretation to what this mixture model could mean in this application. Specifically, mention what the clustering type could mean.

(c) Using the mixture density from part (a) for a partial data point $x_{C_u}^{(u)}$, we are ready to write the incomplete log-likelihood and maximize it to derive the M-step of the EM algorithm. To that end, we will maximize the following incomplete log-likelihood:

$$l(\theta) = \sum_{u=1}^{N} \left[\sum_{j=1}^{K} p(j|u) \log\big(p_j N(x_{C_u}^{(u)}|\mu_{C_u}^{(j)}, \sigma_j^2 I_{|C_u| \times |C_u|})\big) \right]$$

where the posterior probability $p(j|u)$ can be interpreted as the soft-assignment of the data point $x_{C_u}^{(u)}$ to the mixture component j. To maximize $l(\theta)$, we keep the probabilities $p(j|u)$ (the soft-assignments) fixed, and we maximize over the model parameters. In this part we will **derive** the M-step that results after maximizing the above likelihood with respect to the model parameters.

i. First **derive** what the update equation should be for $\mu_{C_u}^{(j)}$ by considering each movie in the mean vector. In other words take the partial derivative with respect $\mu_l^{(j)}$ and setting the derivative to zero.
[Hint: notice that you are deriving the update for a single movie, therefore, make sure you only consider users that rated the particular movie being considered]

ii. Now that we have updated what the new means of our gaussian should be we will update what the spread of them should be. To do that take the partial derivative of the soft-counts log-likelihood function $l(\theta)$ with respect to σ_j^2 for each different type of user j. Notice that we are taking the derivative with respect to the variance not the standard deviation
[Hint: these facts might come in handy: The determinant of a diagonal matrix is the product of its diagonal entries and the inverse of diagonal matrix is the reciprocal of each entry]

iii. The last parameter to update will be the mixing proportions p_j. This quantity must satisfy the constraint $\sum_{j=1}^{k} p_j = 1$, hence it needs to take a little more care to optimize it. Show that the correct update in this case is:

$$p_j = \frac{\sum_1^n p(j|u)}{n}$$

by incorporating the constraint in the optimization problem through the Lagrange multipliers.
[Hint 1: write the expression of the Lagrangian using $\lambda(1 - \sum_j^k p_j)$.]
[Hint 2: take the derivative with respect to p_j and λ and then manipulate the equation to not have λ in any of the expressions.]

(d) In the E-Step of the EM algorithm, one uses the updated model parameters to re-estimate the soft-assignments $p(j|u)$. **Write** down the formula for the E-Step that shows how to update $p(j|u)$ [Hint: the parameters $\mu_{C_u}^{(j)}$, σ_j^2, p_j are to be held fixed in this step].

(e) Next, **implement** the EM algorithm for running on the partially observed model. Use the E- and M- steps you derived in parts (b) and (c) above.
Write a python function **mixGaussMiss**. Your inputs should be

i. X: an $n \times d$ Numpy array of n data points, each with d features
ii. K: number of mixtures; Mu: $K \times d$ Numpy array, each row corresponds to a mixture mean vector;
iii. P: $K \times 1$ Numpy array, each entry corresponds to the weight for a mixture;
iv. Var: $K \times 1$ Numpy array, each entry corresponds to the variance for a mixture;

Your outputs should be

i. output: Mu: $K \times d$ matrix, each row corresponds to a mixture mean;
ii. P: $K \times 1$ Numpy array, each entry corresponds to the weight for a mixture;
iii. Var: $K \times 1$ Numpy array, each entry corresponds to the variance for a mixture;
iv. post: $n \times K$ Numpy array, each row corresponds to the soft counts for all mixtures for an example
v. LL: Numpy array, which records the loglikelihood value for each iteration.

(f) Finally, **run** your EM algorithm using the initialization (function **init**) to learn the model parameters on the incomplete data set (`1000mat.txt`). And once you have learned the parameters using EM, you have at your hands a *generative model*, which you should apply to complete the matrix (generate the missing entries from H_u for all users).

(g) You are given both the complete matrix (`1000mat.txt`) and the uncompleted matrix ($1000mat\_complete.txt$). Now that you have filled the missing entries, we are ready to compare it with original complete matrix. Report the Frobenius norm of the difference between your recovered matrix and the true matrix. [Reference: You could find the definition of Frobenius norm at `http://mathworld.wolfram.com`