Machine Learning (CS771A)
Homework 3 (Due date: Nov 2, 2016, 11:59pm)

Instructions

- Each late submission will receive a 10% penalty per day for up to 2 days. No submissions will be accepted after the 2nd late day.
- We will only accept electronic submissions and the main writeup must be as a PDF file. If you are handwriting your solutions, please scan the hard-copy and convert it into PDF. Your name and roll number should be clearly written at the top. In case you are submitting multiple files, all files must to be zipped and **submitted as a single file** (named: your-roll-number.zip). Please do not email us your submissions. Your submissions have to be uploaded here: `http://tinyurl.com/zyyfrq3`.

Problem 1 (20 marks)

(Implementing Kernel Perceptron) Your task is to implement kernel Perceptron for binary classification. The data you will work with is a subset of the original MNIST data (classes: digits 4 vs 9) and is given in the file `mnist_binary.mat`. For a description of the kernel Perceptron and the form of updates needed to implement it, please refer to the mid-sem exam problem (Part 3, Q4) and its solution. The provided skeleton code `Kernel_Perceptron.m` should also give you more idea about how to implement it. As usual, in the file `Kernel_Perceptron.m`, you only need to complete the parts marked "???". The skeleton code also takes care of various other things, such as splitting the data into training/test, computing accuracies, plotting, etc.

As you can see from the skeleton code, you have to try your implementation using polynomial kernels of various degrees (1,2,4,8,10,20). Note that polynomial kernel with degree 1 is akin to a linear kernel.

For each choice of the kernel, the provided kernel Perceptron (skeleton) code does 10 passes over the training data (our stopping criteria). The code also computes and prints the classification accuracies on test data (but you have to complete the part that predicts the test labels; see the skeleton code for more details) and produces a plot of test accuracy vs number of mistakes made by the kernel Perceptron during training.

For this submission, you have to submit the completed code, and the plot of test accuracy vs iterations. Also answer the following: *(1) Which kernel gives the best test accuracy? (2) Using which kernel, the Perceptron makes the smallest number of mistakes on the training data (in its 10 passes)?*

Problem 2 (20 marks)

(Implementing Matrix Factorization and Matrix Completion) Your task is to implement matrix factorization, i.e., given the observed (i.e., training) entries of an $N \times M$ matrix $\mathbf{X}$, learn a low-rank factorization $\mathbf{X} \approx \mathbf{U}\mathbf{V}^\top$ (using the same model we discussed in the class) where $\mathbf{U}$ is $N \times K$ and $\mathbf{V}$ is $M \times K$, and use the latent factors to predict the missing (i.e., test) entries in $\mathbf{X}$.

The skeleton code (`matrix_factorization.m`) is provided which takes care of various things such as initialization and evaluation. You only need to complete the parts that update the latent factors $\mathbf{U}$ and $\mathbf{V}$. The skeleton code provides more guidelines regarding the implementation.

The provided toy-ish (but real!) data set `movielens100k.mat` (originally taken from `http://grouplens.org/datasets/movielens/`) consists of 100k ratings by 943 users on 1682 movies (each rating between 1-5). The data is further split into 80k ratings that you will use to train the model and 20k ratings that you will predict using the learned latent factors (the "masks" specify which entries are to be used for training vs test). Loading the provided data file in MATLAB will show you these details. For this submission, you have to submit the completed code and the plot of mean-absolute-error (MAE) vs iterations.

Problem 3 (30 marks)

(Implementing a Probabilistic Cousin of K-means) Your task will be to implement EM for a simplified version of GMM and answer a few questions (basically, interpreting the results). Unlike the more general GMM we looked at in the class, here we will assume that all Gaussians have a shared spherical diagonal covariance matrix $\sigma^2\mathbf{I}_D$ (making it like a probabilistic cousin of K-means!). In order to implement this model, you will first need to derive the expression for σ^2. To do so, you need to write the expected complete data log likelihood for this model (with shared $\sigma^2\mathbf{I}$) and do MLE for σ^2. **This exercise itself will be worth 10 marks**.

Again, you are provided some skeleton code and you only need to complete the missing pieces in the `gmm.m` file; specifically, places that have "???" where you need to estimate $\mathbf{Z}$, GMM parameters (means, shared variance, and the mixing proportions), and the Bayesian Information Criteria (BIC) which is one of the metrics to select the "optimal" number of clusters. Note that the BIC is defined as $BIC = -2 * ILL + num\_params * \log(N)$ where, in the case of GMM, "ILL" is the incomplete data log likelihood, "num_params" is the number of free parameters in the model, and N is the number of data points.

Again, remember that you would only need to touch `gmm.m`. Other files are basically helper codes.

Once you complete `gmm.m`, to test your implementation, you may run `test_gmm.m` which will perform clustering on some synthetic 2D data. This script is only meant to be a sanity check for your implementation of `gmm.m`. If you don't see reasonable clusters coming out, something is probably broken.

Once things look reasonable, your main task is to experiment on the (surprise, surprise!) MNIST data using the provided `run.m` script (which, in turn, calls `gmm.m`). You don't have to modify `run.m`.

Running `run.m` will produce a number of figures. Please answer the associated questions with each figure:

- Fig 1 displays a plot of the *complete* log likelihood (red xs) and incomplete log likelihood (blue os) as a function of iteration. As a verification of implementation, you should see that the incomplete log likelihood is always increasing. Moreover, the complete should lower-bound the incomplete (i.e., the blue line should always be above the red line). *Is there a significant difference between the ILLs and CLLs? Do the CLLs seem to converge roughly to the same value across all runs?*
- Fig 2 displays the Bayesian Information Ctriterion (BIC) as a function of the number of clusters (I have tried 2,5,10,15,20,25,30). *What is the shape of this graph; does this make sense?*
- Fig 3-9 display the means of each of the clusters (for each choice of K), together with the $\pi_k's$ ("pk" in the code), i.e., the cluster proportions. Futher, they are sorted by π_k values. At what point do you start actually seeing "real" digits? At what point do you see at least one example of every digit? For the "blurry" cases, what do they seem to be mixtures of? Is there a trend between quality of mean and π_k values?

Answer all the above questions in your writeup and also include your plots (in the write-up, not as separate files). Also submit your main code `gmm.m`. Note that you do not have to submit/discuss anything related to the results on the synthetic 2D data set.

Problem 4 (15 marks)

(EM for Poisson Mixture Model) Recall that a Poisson distribution is a distribution over positive count values; for a count k with parameter λ, the Poisson has the form $p(k \mid \lambda) = \frac{1}{e^\lambda}\frac{\lambda^k}{k!}$. We saw (mid-sem exam problem) that the MLE for λ given a sequence of counts $k_1, \ldots, k_N$ was simply $\frac{1}{N}\sum_n k_n$ – the mean of the counts.

Let's consider an generalization of this: the Poisson mixture model. Believe it or not, this is actually used in web server monitoring. The number of accesses to a web server in a minute typically follows a Poisson distribution.

Suppose we have N web servers we are monitoring and we monitor each for M minutes. Thus, we have $N \times M$ counts; call $k_{n,m}$ the number of hits to web server n in minute m. Our goal is to *cluster* the web servers according to their hit frequency. Construct a Poisson mixture model for this problem and derive the EM algorithm by working out the expectation and maximization steps for this model. Note that here we are basically clustering data where each observation is an M dimensional vector of counts.

Hint: Suppose we want to cluster the data into L clusters; let z_n be the latent variable telling us which cluster web server n belongs to (from one to L). Let λ_l denote the parameter for the Poisson for cluster l. Then, the complete data likelihood for each point n should look pretty close to the Gaussian case, but with a product of Poissons, rather than a multivariate Gaussian. For all the observations, this looks something like:

$$p(\boldsymbol{k}, \boldsymbol{z} \mid \boldsymbol{\lambda}, \boldsymbol{\pi}) = \prod_{n=1}^{N}\prod_{l=1}^{L}\left[\prod_{m=1}^{M}\text{Poisson}(k_{n,m} \mid \lambda_l)\right]^{\mathbf{1}[z_n=l]}$$

Here, $\mathbf{1}[z_n = l]$ is one if $z_n = l$ and zero otherwise (if using the one-hot notation for z_n, we will have $z_{nl} = 1$ and all other components of the vector z_n will be zero), $\boldsymbol{k} = \{\boldsymbol{k}_n\}_{n=1}^N$, is the observed data with the n-th observation denoted as $\boldsymbol{k}_n = \{k_{n1}, \ldots, k_{nM}\}$, $\boldsymbol{z} = \{z_1, \ldots, z_N\}$ denotes the cluster ids of each of the n observations, $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_L\}$ are the parameters of each of the L Poisson distributions in the mixture, and $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_L\}$ denote the mixing proportions (these sum to 1). The goal is to estimate $\boldsymbol{z}$, $\boldsymbol{\lambda}$, and $\boldsymbol{\pi}$ using EM. Using a similar recipe as we used for GMM, you are basically required to do the following:

- Write down the complete data log likelihood for the model.
- Estimate each z_{nl} in the E step (unnormalized expression for z_{nl} is fine).
- Estimate each π_l and λ_l in the M step by maximizing the expected complete data log-likelihood.

Problem 5 (5+10 marks)

(Embeddings) K-means, GMM, and PCA/Factor Analysis, despite solving seemingly different problems (hard-clustering, soft-clustering, dimensionality reduction, respectively), in the end can be seen as learning a K-dimensional latent variable $\boldsymbol{z}_n$ for each observation $\boldsymbol{x}_n$, which can be treated as an "embedding" as $\boldsymbol{x}_n$. How do the embeddings learned by these methods differ from each other? (answer in 50 words or less)

(Expectation Maximization) Explain the Expectation Maximization (EM) algorithm intuitively. When do we need EM and how using EM helps in parameter estimation? (answer in 100 words or less)