Spring 2017 EECE 5644 Homework #5

Prof: Jaume Coll-Font TA: Aziz Kocanaogullari, Zhiqiang Tao

Make sure you read the problem statements and answer the questions. Submit the code online. This homework could be a bit challenging, so I suggest you start early.

5.1 (50 pts) **Bonus: Expectation Maximization for handwritten digit clustering.** In this problem, you will explore unsupervised learning with EM for a real application: handwritten digit recognition. The nice people at NIST have generated a dataset with many examples of handwritten digits (0-9). This dataset is provided as `digitsSmall.mat` with the data in a 3000 x 784 array. This array represents 3000 total samples of 28x28 (784) images of each digit. The label vector determines which digit corresponds to each sample. To plot sample 5 from this array, you can use `imagesc(reshape(inputData(5,:),28,[]));` The following figure shows the some example digits from this dataset.

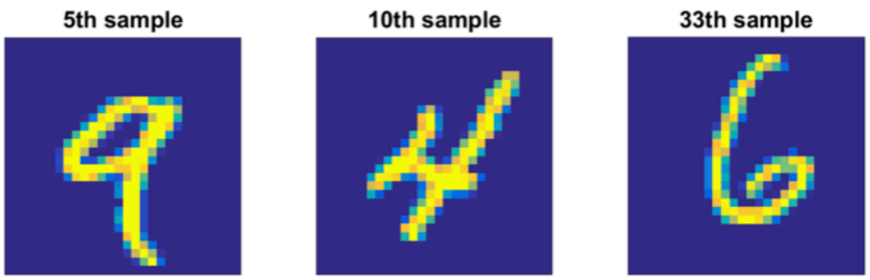

Figure 1: Example digits

You can also easily find the mean images by using the label comparison as logical `meanDigit2 = mean(inputData(labels==2,:));`. The following figure shows some mean images.

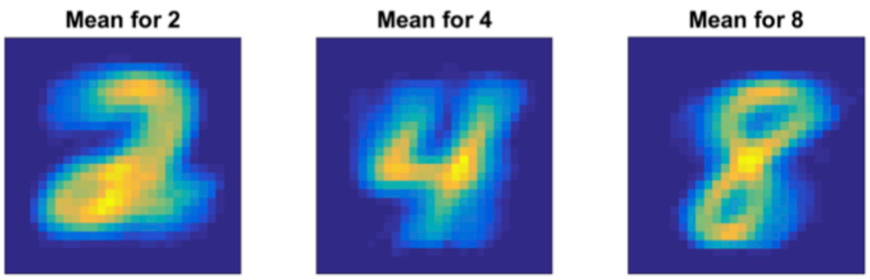

Figure 2: Example digits

We will perform clustering (unsupervised learning) using what you learned from EM; therefore, we won't be using the labels provided. Instead of using Gaussian mixtures, we will use Bernoulli mixtures and treat the digits as binary images.

(a) (5 pts) In this problem, we are not using the labels provided in the dataset to learn the cluster dependent parameters. In your words, why would it be useful to develop an unsupervised learning algorithm for this application? Can you think of other applications where labeling datasets is expensive?

(b) (10 pts) As mentioned before, we will treat is sample image as binary random variable modeled by a Bernoulli distribution (you can do this by thresholding the images with 0.5 i.e. `inputData = inputData>0.5;`). In other words, for a given $\mathbf{x} = [x_1 \cdots x_D]$, we will have its probability given by

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{d=1}^{D} \mu_d^{x_d}(1-\mu_d)^{1-x_d}$$

where $\boldsymbol{\mu} = [\mu_1 \cdots \mu_D]$ is the mean vector with the mean for each element. We are treating each dimension as independent Bernoulli distributions. However, for the mixture model, we will treat each potential digit as its own Bernoulli. Therefore, the distribution for a sample $\mathbf{x}$ drawn from our dataset will be given by:

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}|\boldsymbol{\mu}_k)$$

where $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_K\}$ (excuse my abuse of notation with $\boldsymbol{\mu}$) and $\boldsymbol{\pi} = [\pi_1 \cdots \pi_K]$ are each of the component's mean and prior, respectively, and

$$p(\mathbf{x}|\boldsymbol{\mu}_k) = \prod_{d=1}^{D} \mu_{kd}^{x_d}(1-\mu_{kd})^{1-x_d}$$

From this set of equations, we can observe that the log-likelihood for a dataset $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ is given by

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k)$$

Like in the Gaussian mixture case, we can introduce the binary K-dimensional variable $\mathbf{z} = [z_1 \cdots z_K]$ to indicate the component correspondence (only one element will be equal to 1 in this vector). Given this latent variable, we can write the distribution of $\mathbf{x}$ as:

$$p(\mathbf{x}|\mathbf{z}, \boldsymbol{\mu}) = \prod_{k=1}^{K} p(\mathbf{x}|\boldsymbol{\mu}_k)^{z_k}$$

$$p(\mathbf{z}|\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_k}$$

With some work, it is possible to write the expectation of the complete-data log-likelihood as:

$$E_{\mathbf{Z}}[\ln p(\mathbf{X}, \mathbf{Z}|\boldsymbol{\mu}, \boldsymbol{\pi})] = \sum_{n=1}^{N} \sum_{k=1}^{K} E[z_{nk}] \left\{ \ln \pi_k + \sum_{d=1}^{D} [x_{nd} \ln \mu_{kd} + (1-x_{nd}) \ln(1-\mu_{kd})] \right\}$$

where $\mathbf{Z} = \{\mathbf{z}_n\}$ and $\mathbf{X} = \{\mathbf{x}_n\}$. From this, the E-step will be given by

$$E[z_{nk}] = \frac{\pi_k p(\mathbf{x}_n|\boldsymbol{\mu}_k)}{\sum_{l=1}^{K} \pi_l p(\mathbf{x}_n|\boldsymbol{\mu}_l)}$$

and the M-step by

$$N_k = \sum_{n=1}^{N} E[z_{nk}]$$

$$\pi_k = \frac{N_k}{N} = \frac{1}{N}\sum_{n=1}^{N} E[z_{nk}]$$

$$\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_{n=1}^{N} E[z_{nk}]\mathbf{x}_n$$

Prove the formula for $\boldsymbol{\mu}_k$ by setting the derivative of the $E_{\mathbf{Z}}[\cdot]$ equation above to 0 (derivative with respect to $\boldsymbol{\mu}_k$ of course).

(c) (10 pts) Implement the Expectation Maximization (EM) algorithm for Bernoulli clustering with the steps as shown in (b). The implementation MUST be a function that takes AT LEAST the following inputs:

i. inputData: nSamples x dDimensions array with the data to be clustered
ii. numberOfClusters: number of cluster for algorithm
iii. stopTolerance: parameter for convergence criteria
iv. numberOfRuns: number of times the algorithm will run with random initializations

Notice that I say AT LEAST. You may want to add some optional parameters if you want to give the user more control over the algorithm. The function should output the results for best EM clustering. The output should be AT LEAST:

i. clusterParameters: numberOfClusters x 1 struct array with the Bernoulli mixture parameters:
   - .mu - mean of the Bernoulli component
   - .prior - prior of the cluster component

Notice that I say AT LEAST. I suggest you also have the following outputs since they will make your plotting easier I think:

i. estimatedLabels: nSamples x 1 vector with labels based on maximum probability. Since EM is a soft clustering algorithm, its output are just densities for each cluster in the mixture model
ii. logLikelihood: 1 x numberOfIterations vector with the log-likelihood as a function of iteration number

This algorithm is tricky to get right. I strongly suggest you run a simple tests firsts.

There are a couple of details you need to be mindful:

i. **Initialization**: your algorithm can end up in a local maximum. This is why one of the inputs is the number of times you will run EM with random initializations. By random initial conditions, I mean, select the **initial K mean vectors by drawing from a uniform distribution between 0.25 and 0.75**. You can make the initial priors to be uniform. Then, run your algorithm until convergence starting from each of these initial conditions. Finally, pick the one that results in the highest log-likelihood.

ii. **Convergence**: from a practical viewpoint, you need to decide when it is no longer worth iterating. You could stop if the increase in log-likelihood between the last two iterations is less than 0.001% of the change in likelihood between the current value and the log-likelihood value after the very first iteration of the algorithm. For these data sets, you can also impose a maximum number of iterations to halt the algorithm (e.g., 50) if it gets that far and still has not converged.

iii. **Computational performance**: computing $E[z_{nk}]$ can be tricky due to the size of the dataset. Think really hard how to vectorize this operation. As a hint, you could use `bsxfun` or the $\exp(\log(\cdot))$ trick to convert powers to multiplications.

(d) (10 pts) Run the EM algorithm on `digitsSmall.mat` **only for digits 0,1,4** with K=3. Make a 2x3 figure (with subplot) to generate the following plots: on the top row, the means for the clusters (they should look like a 0, 1, and 4) and on the bottom row, the difference between these cluster means and the corresponding true means. The true mean for comparison can be computed as `meanDigit0 = mean(inputData(labels==0,:)>0.5);` How do the cluster means compare it to the true ones?

(e) (15 pts) Run the EM algorithm on `digitsSmall.mat` for all digits for K=5,10,20. Make 3 figures (1x5,2x5,4x5) total, one for each K, with the images of the cluster means as found by your EM algorithm. Make sure you run enough initializations (50 for example). What can you say about the quality of the means as K increases? Did K=10 do a good job? What about K=20? What do you think could be the problem? Would a Gaussian mixture model improve these results? Would that be a good model for these images?

5.2 (50 pts) **Support Vector Machines for handwritten digit classification**. In this problem, you will use MATLAB's SVM tool `fitcsvm`. Please, read the documentation carefully to learn its functionality. This experience will teach you how to use tools that you haven't developed yourself. We will explore the multi-class extension to SVMs. We will use the smaller dataset to learn and use the bigger dataset to test the generalization capability of the classifier.

(a) (10 pts) Describe 3 ways to extend standard SVMs to multi-class classification. State the strengths and weaknesses of each approach

(b) (10 pts) For the next items, we will focus on the one-vs-all approach to multi-class SVM. I suggest you build a class for multiclass SVM and write Learn and Predict methods. Alternatively, you could write 2 functions: a "learn" one that outputs the set of SVM models that will be passed to the "predict" function. To learn the digits' models, you will need to call **`fitcsvm`** 10 times with the appropriate positive and negative labels. To predict, use the score that MATLAB's **`predict`** function outputs for the 10 models and pick the label with the maximum score. Make sure you use the correct score in the output i.e. read the fun manual (RTFM).

(c) (10 pts) Use the **`digitsSmall.mat`** to learn and **`digitsLarge.mat`** to test. Make a figure with an image of the confusion matrix (predicted labels vs true labels) with the total accuracy on the title. Use default options

(d) (10 pts) Use the **`digitsSmall.mat`** to learn and **`digitsLarge.mat`** to test. Make a figure with an image of the confusion matrix (predicted labels vs true labels) with the total accuracy on the title. Use the radial basis function kernel for kernel-SVM with the kernel scale parameter as automatic (if not, you will perform poorly). Compare this to (c). Does RBF improve results? What does the automatic kernel scale do?

(e) (10 pts) Use the **`digitsSmall.mat`** to learn and **`digitsLarge.mat`** to test. Make a figure with an image of the confusion matrix (predicted labels vs true labels) with the total accuracy on the title. Use the polynomial kernel with order 3. Compare this to (c).

(f) (10 pts) Bonus: explain why the RBF kernel implies an infinite dimensional feature space.