Statistical Machine Learning GU4241/GR5241
Spring 2017
https://courseworks.columbia.edu/

Linxi Liu
ll3098@columbia.edu
Shuaiwen Wang
sw2853@columbia.edu
Haolei Weng
hw2375@columbia.edu

Homework 5

Due: Tuesday, April 11th, 2017

Homework submission: We will collect your homework **at the beginning of class** on the due date. You also need to submit **your code** for Problem 2 and 3 through Canvas. If you cannot attend class that day, you can leave your solution in the homework dropbox 904 at the Department of Statistics, 9th floor SSW, at any time before then.

Problem 1 (Hierachical clustering, 10 points)

Perform single-linkage hierarchical clustering on the following 2-dimensional observations. Use the Manhattan distance between a pair of points: $d(A, B) = |X_{1A} - X_{1B}| + |X_{2A} - X_{2B}|$. Show your results after each step.

Obs.	X_1	X_2
A	1	4
B	1	3
C	3	4
D	5	2
E	3	2
F	3	0

Problem 2 (Multinomial Clustering, 30 points)

In this exercise, we will apply the EM algorithm and a finite mixture of multinomial distributions to the image segmentation problem. Image segmentation refers to the task of dividing an input image into regions of pixels that "belong together" (which is about as formal and precise a definition as you will be able to find in the literature.) One way to approach image segmentation is to extract a suitable set of features from the image and apply a clustering algorithm to them. The clusters output by the algorithm can then be regarded as the image segments. The features we will use are histograms drawn from small regions in the image.

Histogram clustering: The term *histogram clustering* refers to grouping methods whose input features are histograms. A histogram can be regarded as a vector; if all input histograms have the same number of bins d, a set of input histograms can be regarded as a set of vectors in $\mathbb{R}^d$. We can therefore perform a simple histogram clustering by running a k-means algorithm on this set of vectors. The method we will implement in this problem is slightly more sophisticated: Since histograms are represented by multinomial distributions, we describe the clustering problem by a finite mixture of multinomial distributions, which we estimate with the EM algorithm.

The EM algorithm for multinomial mixtures. The input of the algorithm is:

- A matrix of histograms, denoted `H`, which contains one histogram vector in each row. Each column corresponds to a histogram bin.
- An integer K which specifies the number of clusters.
- A threshold parameter τ.

The variables we need are:

- The input histograms. We denote the histogram vectors by $\mathbf{H}_1, ..., \mathbf{H}_n$.
- The *centroids* $\mathbf{t}_1, ..., \mathbf{t}_K$. These are $\mathbb{R}^d$ vectors (just like the input features). Each centroid is the parameter vector of a multinomial distribution and can be regarded as the center of a cluster; they are computed as the weighted averages of all features assigned to the cluster.
- The *assignment probabilities* $\mathbf{a}_1, ..., \mathbf{a}_n$. Each of the vectors $\mathbf{a}_i$ is of length K, i. e. contains one entry for each cluster k. The entry a_{ik} specifies the probability of feature i to be assigned to cluster k. The matrix which contains the vectors $\mathbf{a}_i$ in its rows will be denoted P.

The algorithm iterates between the computation of the assignment probabilities $\mathbf{a}_i$ and adjusting the cluster centroids $\mathbf{t}_k$.

The algorithm:

1. Choose K of the histograms at random and normalize each. These are our initial centroids $\mathbf{t}_1, ..., \mathbf{t}_K$.
2. Iterate:

   (a) E-step: Compute the components of each $\mathbf{a}_i$, the assignment probabilities a_{ik}, as follows:

   $$\begin{aligned} \phi_{ik} &:= \exp\left(\sum_j H_{ij}\log(\mathbf{t}_{kj})\right) \\ a_{ik} &:= \frac{c_k\phi_{ik}}{\sum_{l=1}^K c_l\phi_{il}} \end{aligned}$$

   Note that ϕ_{ik} is the multinomial probability of $\mathbf{H}_i$ with parameters $\mathbf{t}_k$, up to the multinomial coefficient $\frac{n!}{H_{i1}!\cdots H_{id}!}$, which cancels out in the computations of a_{ik}.

   (b) M-step: Compute new mixture weights c_k and class centroids $\mathbf{t}_k$ as

   $$\begin{aligned} c_k &= \frac{\sum_{i=1}^n a_{ik}}{n} \\ \mathbf{b}_k &= \sum_{i=1}^n a_{ik}\mathbf{H}_i \\ \mathbf{t}_k &= \frac{\mathbf{b}_k}{\sum_{j=1}^d b_{kj}} \end{aligned}$$

   (c) Compute a measure of the change of assignments during the current iteration:

   $$\delta := \|\mathbf{A} - \mathbf{A}^{\text{old}}\| ,$$

   where $\mathbf{A}$ is the assignment matrix (with entries a_{ik}), $\mathbf{A}^{\text{old}}$ denotes assignment matrix in the previous step, and $\| . \|$ is the matrix 1-norm (the largest sum of column absolute values). This is implemented in R as `norm( . ,"O")`.

3. Terminate the iteration when $\delta < \tau$.
4. Turn the soft assignments into a vector `m` of hard assignments by computing, for $i = 1, ..., n$,

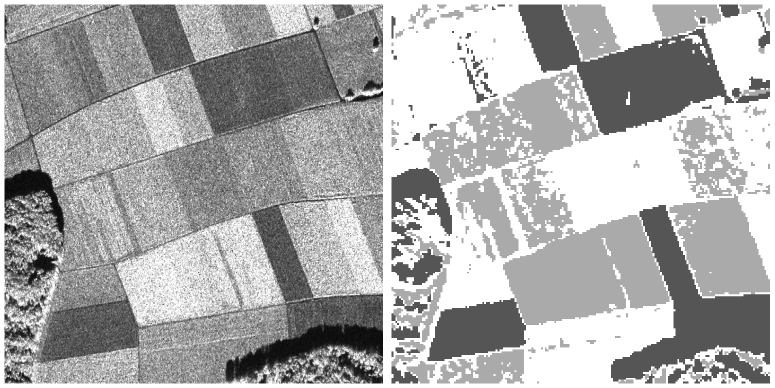
   $$m_i := \arg\max_{k=1,...,K} a_{ik} ,$$

   i. e. the index k of the cluster for which the assignment probability a_{ik} is maximal.

The histogram data: The histograms we use have been extracted from an image by the following procedure:

1. Select a subset of pixels, called the *sites*, at which we will draw a histogram. Usually, this is done by selecting the pixels at the nodes of an equidistant grid. (For example, a 2-by-2 grid means that we select every second pixel in every second row as a site.)
2. Place a rectangle of fixed radius around the site pixel.
3. Select all pixels within the rectangle and sort their intensity values into a histogram.

The data we provide you with was drawn from the 800x800 grayscale image shown below (left):



The image is a radar image; more specifically, it was taken using a technique called synthetic aperture radar (or SAR). This type of image is well-suited for segmentation by histogram clustering, because the local intensity distributions provide distinctive information about the segments. The image on the right is an example segmentation using $K = 3$ clusters, computed with the algorithm described above.

The histograms were drawn at the nodes of a 4-by-4 pixel grid. Since the image has 800×800 pixels, there are $200 \times 200 = 40000$ histograms. Each was drawn within a rectangle of edge length 11 pixels, so each histogram contains $11 \times 11 = 121$ values.

Homework problems. Download the data file `histograms.bin` from the course homepage. You can load it in R with the command

```
H<-matrix(readBin("histograms.bin", "double", 640000), 40000, 16)
```

which results in a 40000×16 matrix. Each row is a histogram with 16 bins. You can check that the matrix has been loaded correctly by saying `dim(H)` (which should be 40000 rows by 16 columns) and `rowSums(H)` (the sum of each row should be 121).

1. Implement the EM algorithm in R. The function call should be of the form `m<-MultinomialEM(H,K,tau)`, with `H` the matrix of input histograms, `K` the number of clusters, and `tau` the threshold parameter τ.
2. Run the algorithm on the input data for `K=3`, `K=4` and `K=5`. You may have to try different values of τ to obtain a reasonable result.
3. Visualize the results as an image. You can turn a hard assignment vector `m` returned by the EM algorithm into an image using the `image` function in R.

Hints:

- You may encounter numerical problems due to empty histogram bins. If so, add a small constant (such as 0.01) to the input histograms.
- Both the E-step and the M-step involve sums, which are straightforward to implement as loops, but loops are slow in interpreted programming languages (such as R or Matlab). If you want to obtain a fast algorithm, try to avoid loops by implementing the sums by matrix/vector operations.

Problem 3 (Convolutional Neural Networks for ZIP Code Data, 6 extra points)

In this problem, you are expected to design a convolutional neural network for a familiar data set—the ZIP code data. We will use the images for all ten digits as the training set, and evaluate the model on a separate test set. `zip.train` and `zip.test` can be downloaded on Canvas.

We hope you can use the same toolkit in order to simplify the grading process. We recommend Keras. Keras is a high-level neural networks API, written in Python. You can find the installation instructions and a brief introduction at https://keras.io. Before installing Keras, make sure that you have installed Python (https://www.python.org) on you computer (either Python 2 or 3). IPython Notebook (https://ipython.org/notebook.html) is also recommended (it's similar to R Markdown). With this interactive environment, you can easily compile and execute your code. There are several examples of Keras with Python code provided at
https://github.com/fchollet/keras/tree/master/examples.

Here, we have a classification problem with ten classes. The inputs are 16×16 images. Design a convolutional neural network for this problem. You can use the architecture in our textbook (Section 11.7) as a reference. Grading is based on the performance of your model: test error (measured by misclassification rate on test set) should be at least less than 5%.