

## Series 3, Nov 14th, 2007 (pLSA)

Please hand in your solutions until Wednesday, Nov 28th, 2007.

The objective of this exercise is to implement the pLSA algorithm, and to apply it for estimation of a set of topics from a collection of documents.

**Brief repetition.** Statistical data analysis models (including pLSA) are often designed “backwards”: We start by defining a probabilistic model, the behavior of which is controlled by a set of parameters. The model describes how data is generated at random, given the values of the parameters. The actual data analysis is the *inverse* process, that is, estimate the parameters from given data.

For pLSA, we assume that the input data consists of  $n$  documents, containing a total number of  $J$  different words. The variables we need are:

- The input histograms. We denote the histogram vectors by  $\mathbf{h}_1, \dots, \mathbf{h}_n$ . Each histogram describes a single document. The vectors are of the form  $\mathbf{h}_i = (h_{i1}, \dots, h_{iJ})$ . Entry  $h_{ij}$  is the total number of times that word  $j$  occurs in document  $i$ .
- The *centroids*  $\theta_1, \dots, \theta_K$ . These are vectors in  $\mathbb{R}^J$ , with non-negative entries and normalized to sum 1. Each vector represents a topic, and entry  $\theta_{kj}$  is the estimated probability of word  $j$  occurring in a document of topic  $k$ .
- The *assignment probabilities*  $\mathbf{q}_1, \dots, \mathbf{q}_n$ . These are not actually part of the model, but will be used by the EM algorithm. Each of the vectors  $\mathbf{q}_i$  is of length  $K$ , i.e. contains one entry for each cluster  $k$ . The entry  $q_{ik}$  specifies the probability that the document of index  $i$  is assigned to topic  $k$ . The matrix which contains the vectors  $\mathbf{q}_i$  in its rows will be denoted  $Q$ .

Recall that pLSA, regarded as a statistical model, describes the random generation of a word from a document. The document is characterized by the topics it contains, each described by a vector  $\theta_k$ . We can use the mixture weights  $c_k$  to adjust how prominently a topic should be represented in the document. A word is generated at random by drawing a histogram  $\mathbf{h}$ , containing a single entry for a single word, from the mixture model

$$P_{\text{pLSA}}(\mathbf{h}|\mathbf{c}, \theta_1, \dots, \theta_K) = \sum_{k=1}^K c_k P(\mathbf{h}|\theta_k).$$

The components  $P(\mathbf{h}|\theta_k)$  are multinomial distributions:

$$P(\mathbf{h}_i|\theta_k) = \frac{(\sum_{j=1}^J h_{ij})!}{\prod_{j=1}^J h_{ij}!} \prod_{j=1}^J \theta_j^{h_{ij}} = \frac{(\sum_{j=1}^J h_{ij})!}{\prod_{j=1}^J h_{ij}!} \exp\left(\sum_{j=1}^J h_{ij} \theta_j\right) = f(\mathbf{h}_i) \exp\left(\sum_{j=1}^J h_{ij} \theta_j\right).$$

In the last equation, we have summarized the terms depending only on  $\mathbf{h}_i$  in the function  $f(\mathbf{h}_i)$ .

**The pLSA algorithm.** 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 groups (topics).
- A threshold parameter  $t$ .

The matlab function call should be of the form

$$[c, \text{Theta}] = \text{pLSA}(H, K, t)$$

where  $c$  is the vector of weights  $c_k$  and  $\text{Theta}$  a matrix collecting the topic vectors  $\theta_k$ .

Here are the individual steps of the algorithm:

1. Initialize: Choose  $K$  of the histograms at random and normalize each to unit mass by computing  $\frac{\mathbf{h}_i}{\sum_j h_{ij}}$ . These are our initial centroids  $\theta_1, \dots, \theta_k$ . Initialize the weight vector as  $\mathbf{c} = (\frac{1}{K}, \dots, \frac{1}{K})$ .

2. Iterate:

(a) Expectation step: Compute the components of each  $\mathbf{q}_i$ , the assignment probabilities  $q_{ik}$ , in two steps as follows:

$$(1) \quad a_{ik} := \exp\left(\sum_j h_{ij} \log(\theta_{kj})\right) \quad (2) \quad q_{ik} := \frac{c_k a_{ik}}{\sum_{\tau=1}^k c_{\tau} a_{i\tau}}$$

*Explanation:*  $a_{ik}$  is the multinomial probability of  $\mathbf{h}_i$ , up to the coefficient  $f(\mathbf{h}_i)$ . For  $i$  fixed, the coefficient is identical for all  $k$ , so we have dropped it, since it would cancel from the computation anyway.

(b) Maximization step: Compute new class centroids  $\theta_k$  as

$$(1) \quad \mathbf{b}_k := \sum_{i=1}^n q_{ik} \mathbf{h}_i \quad (2) \quad \theta_k := \frac{\mathbf{b}_k}{\sum_{j=1}^J b_{kj}}$$

The component weights  $c_1, \dots, c_K$  are computed as

$$c_k := \frac{\sum_{i=1}^n q_{ik}}{\sum_{\tau=1}^K \sum_{i=1}^n q_{i\tau}}.$$

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

$$\text{change} := \|Q - Q^{\text{old}}\|,$$

where  $Q^{\text{old}}$  denotes the matrix of assignments before the current iteration, and  $\|\cdot\|$  is the matrix 1-norm (the largest sum of column absolute values). The 1-norm is implemented in matlab as `norm(. , 1)`.

3. Terminate the iteration when  $\text{change} < \tau$ .

4. Return the model parameters, that is, the coefficient vector  $\mathbf{c}$  and the topic parameters  $\theta_1, \dots, \theta_K$ .

**The data.** The data in the file `data_ex3.tgz` contains 2243 documents (news items by the associated press). The data has to be preprocessed for the algorithm to work. The files contained in the archive are:

- `histograms_preprocessed.mat` - Preprocessed histograms on which the algorithm should output reasonable results. Use these to test your implementation. Format: Each row is a histogram, columns correspond to words.
- `vocabulary_preprocessed.mat` - The list of words contained in the documents, for the preprocessed histograms. This is a list of strings, where the word in entry  $j$  corresponds to the entry  $j$  in the vectors  $\mathbf{h}$  and  $\theta$ .
- `histograms_original.mat` - The original histograms (no preprocessing), provided for the enthusiastic. Apply your own preprocessing, if you like.
- `vocabulary_original.mat` The word list for the original histograms.
- `ap_texts.txt` The original input text files, to give you an idea of what you are working with.

For preprocessing, we have (i) removed all words not occurring more often than 20 times in at least one document; (ii) removed all documents with zero words from the resulting data set; and (iii) normalized each histogram to mass 10. The last step highlights a problem with the numerical stability. If histograms are normalized to mass one (the way normalization usually works), the results are decidedly worse (try it). The “right” mass depends on several factors, such as the number of bins, the number of documents, and the average sparseness of the histogram matrix.

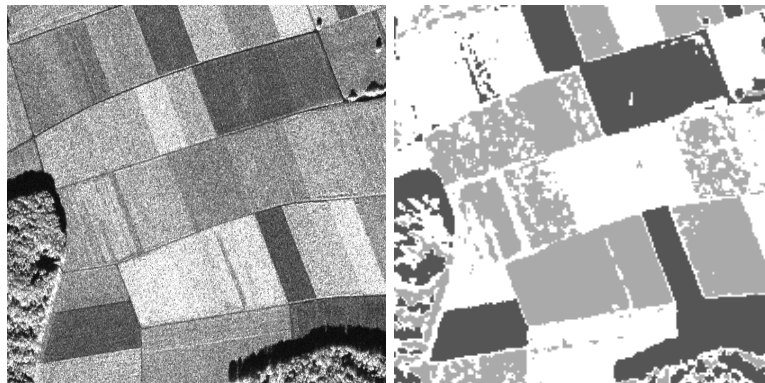
**What we would like you to do.**

1. Implement the algorithm as a matlab function `[c,Theta]=pLSA(H,K,τ)`.

2. Run it on the preprocessed data. Start e.g. with  $K = 10$  clusters, and a convergence threshold of  $t = 10^{-3}$ . (A few hundred iterations at most should do.)
3. For each of the topic descriptors  $\theta_k$ , select the ten largest entries, and map them back to the vocabulary. The list will contain the ten most frequent terms (or most “typical”) terms for the estimated topic.
4. Try to experiment with the number  $K$  of topics.

As solution, please let us have a copy of your code and the ten-term lists representing your estimated topics.

**Optional application: Image segmentation.** The multinomial mixture model implemented by your matlab function `pLSA` can be applied to estimate groups (also called *clusters*) from any kind of histogram data, not just text documents. A particularly illustrative application is image segmentation: The advantage here is that we can simply plot the result as an image, instead of looking at lists of topic terms. Though the problem has no direct connection to text retrieval, it may help you to get to grips with the algorithm. The data file `image_histograms.mat` contains a matrix of histograms (= rows of the matrix) extracted from the radar image below (left).



The data was extracted at follows: A small sliding window ( $11 \times 11$  pixels) was moved over the image, and centered at each node of a 4-by-4 grid. At each position, the grayscale intensities of all pixels within the window were collected in a histogram with 16 bins. The possible grayscale values are  $0, \dots, 255$ , such that each histogram bin encodes a range of 16 values. Intuitively, each histogram measures the local distribution around a point in the image. You can try the following:

1. At the end of your function `pLSA`, add a line `[dummy,I]=max(Q')`; . This turns the soft assignments into hard assignments. For each histogram, the vector `I` contains the index of the cluster the histogram is assigned to. Return the vector `I`.
2. Apply your `pLSA` implementation to the histogram data set, e.g. with  $K = 3$  clusters.  
**Hint:** If you encounter numerical problems due to empty histogram bins, try to add a small constant (like 0.01) to the whole histogram matrix before you pass it to the algorithm.
3. Visualize the result as an image by the command `result=reshape(I,200,200)/K`. This rearranges the vector according to the image structure, and division by  $K$  rescales the entries to the unit interval. You can display it using `imshow(result)`. It should look somewhat like the image on the right above.
4. Repeat the experiment with different numbers  $K$  of clusters.
5. You can watch your algorithm at work by visualizing the current solution during each step of the iteration. Put something like this at the end of the iteration loop:  

```
[dummy,I]=max(Q');
result=reshape(I,200,200)/K;
imshow(result);
drawnow;
```

**Implementation hint.** Matlab is an interpreted language, and excruciatingly slow at executing loops. You will need to loop over iterations, but try to avoid loops everywhere else. To execute computations of the form  $\sum_i a_{ij} b_i$ , use a matrix-vector multiplication. To do the same for multiple vectors, collect them in a matrix and use matrix multiplication. If you have to divide rows or columns of a matrix by different entries of a vector (e.g. when computing the  $q_{ik}$ ), use the `repmat` command and element-wise division (by the `./` operator).