Mixture of categorical distributions and the Expectation Maximization Algorithm

1 Theoretical Design

A survey institute asks n people to complete a questionnaire containing m questions. In order to simplify the problem, one assumes every question can be answered by one single choice among k possible answers. The answer given to question i by person j is denoted $a_{i,j}$ 2 f1 ... kg. The survey institute wants to analyse the answers to the questionnaire by clustering people in g groups (or clusters).

Question 1.1

Propose a mixture model to solve this problem and specify what are the visible and latent variables $V_1, V_2, \ldots$ and H of the model. Introduce a hypothesis to simplify the joint distribution $\mathrm{P}_{H,V_1,V_2,\ldots}$. Represent the resulting model as a Bayesian Network. Specify formally the parameters θ of the model and distributions q_i of latent variables $\mathbf{H}_i$.

Question 1.2

Apply the Expectation Maximization method to the mixture model. Write down the update equations for the E and M steps. One recalls that given a mixture model of observable variables $\mathbf{V}$ and hidden variable H and given a set of i.i.d samples $(\mathbf{v}_i)_{1\leq i\leq n}$,

The E step updates distribution q_i of the hidden variable H of a mixture model for sample i according to:

$$8i, 8h, q_i(h) \qquad \frac{1}{K}\,\mathrm{P}\,(\,\mathbf{V}=\mathbf{v}_i\;j\;H=h,\theta\,)\,\mathrm{P}\,(\,H=h\;j\;\theta\,) \tag{1}$$

where θ are the current values for the model parameters and K is a normalization factor so that $\sum_h q_i(h) = 1$.

The M step updates the mixture weights $p_h = \mathrm{P}\,(H=h)$ as

$$8h, p_h \qquad \frac{1}{K}\sum_i q_i(h) \tag{2}$$

where K is a normalization factor so that $\sum_h p_h = 1$.

The M step updates parameters of cluster distributions $\mathrm{P}\,(\,\mathbf{V}\;j\;H=h\,)$ using almost the same formulae as with a ML estimation, but where samples $\mathbf{v}_i$ are weighted by $q_i(h)$ (e.g. see formulae for Gaussian Mixture Models).

Hence the parameter $p_{h,\mathbf{v}} = \mathrm{P}\,(\,\mathbf{V}=\mathbf{v}\;j\;H=h\,)$ is updated according to:

$$p_{h,\mathbf{v}} \qquad \frac{\sum_i q_i(h)\,\mathbb{1}(\mathbf{v}_i=\mathbf{v})}{\sum_i q_i(h)} \tag{3}$$

NB: if $\mathbf{V}$ gather variables V_j that are independent conditionally to H, the formula remains true when replacing $\mathbf{V}$ by each of its component V_j.

2 Implementation

In order to help you to implement EM, a script `survey.py` written in Python3 is provided. It contains useful functions to generate a mixture model of categorical distributions, to generate data from a model, to display graphically parameters of a cluster distribution, to display distributions of hidden variables, etc. The only function that is left to you for implementation is the function `em`. But first test these useful commands in this order, from the directory containing `survey.py`:

```
$ ipython3                          # Launch an ipython interpreter
> import survey                     # Load the file survey.py
> import imp                        # Load the import module
> imp.reload(survey)
        # Reload the file survey.py every time you modify it
> help(survey.basicModel)           # Print help for a function
> M = survey.basicModel()           # Get a simple model
> M[0]                              # Print the mixture coefficients
> M[1]                              # Print the answer probabilities
> survey.drawModel(M)               # Plot the answer probabilities
> help(survey.generateData)
> D,C = survey.generateData(M,100)
        # Generate data D for 100 people
> D                                 # Print the data
> C                                 # Print the real clusters
```

Question 2.3

Implement EM in the function `survey.em`. In the first version one will run E and M steps for a fixed number of iterations (passed as an argument). One will think to use logarithm at the right places in order to avoid numerical accuracy problems. One will also think to initialize properly model parameters in order to avoid symmetry problems. In order to implement EM you will need to manipulate 1D and 2D arrays of numerical values. This is done using the `numpy` module (see documentation at `http://www.numpy.org/`). You can learn how to use `numpy` in the source code provided by `survey.py`. In a Python3 interpreter, type the following commands to understand the basics of `numpy`:

```
> import numpy as np
        # Load the numpy module symbols by prefixing them with np
> a = np.array([[1, 2, 3],[4, 5, 6]])
> a
> a.shape
> a.T
> a[0,2]
> (a+1)*a
> np.exp(np.log(a) * 2)
> np.max(a)
> np.sum(a,0)
> np.sum(a,1)
> a[:,1]
> a[:,1].shape
> a[1,0:2]
> a[1,0:2].shape
> (a - 1) / 2
> a[:,1] = 0
> a
> b = a[:,1].reshape((1,2))
```

```
> b.shape
> c = np.zeros((2,4))
> c
> np.random.random(size=(4,2))
```

In order to test and debug your algorithm, you can run the following commands in this order:

```
> M = survey.basicModel()        # Get a simple model
> D,C = survey.generateData(M,100)
        # Generate data D for 100 people
> help(survey.em) # Get the expected arguments and outputs of survey.em
> m,h = survey.em(D, 2,100)
        # Run your algorithm for 100 iterations of EM
        # on data D, for 2 clusters
> m      # Print the learnt model
> h      # Print the learnt cluster distributions (latent variables)
> survey.drawModel(m)    # Plot the learnt cluster distributions
> survey.drawModel(M)
        # To be compared with the real cluster distributions
> survey.drawCluster(h)
        # Plot the learnt latent variables distributions
> survey.drawCluster(C)
        # To be compared with real sample clusters
> help(survey.compareClusters)
> score, mapping = survey.compareClusters(C,h)
        # Compare real clusters C with learnt cluster distributions
> score # Print overlap score of C and h
> mapping
        # Print best mapping between real clusters and learnt ones.
> pm,ph = survey.permuteClusters(m,h,mapping)# Align the learnt
        # clusters with the real ones (if em worked correctly)
> survey.drawCluster(ph)
        # Plot the permuted learnt latent variables distributions
> survey.drawCluster(C) # To be compared with real sample clusters
```

Question 2.4

Evaluate the results of your algorithm for different models, thanks to the following commands:

```
> help(survey.generateModel)
> M = survey.generateModel(20,5,4)
        # Generate randomly a model of 20 questions,
        # 5 answers per question and 4 clusters
> D,C = survey.generateData(M,100)
        # Generate data D for 100 people
> m,h = survey.em(D,5,100)
> ...
```

Observe the quality of the results, depending on the contrast between distributions of the initial clusters.

Question 2.5

Change the EM initialization so that the parameters of the different cluster distributions $\mathrm{P}(\mathbf{V} = \mathbf{v}_i \, j \, H = h)$ are all equal. Test and draw the resulting latent variable distributions using `survey.drawCluster(h)`. Explain.

3 Improvements

Question 3.6

Add a function `survey.loglikelihood(M,D,H)` that computes and returns the loglikelihood of a model `M` given the data `D` and hidden variables distributions `H` (as returned by EM). Evaluate the loglikelihood of the learnt model. What happens when one increases the number of clusters for the learnt model?

Question 3.7

Make a copy of function `survey.em` and modify it so that EM does not terminate after a fixed number of iterations but when the learnt model has converged. To do so, one will compute the loglikelihood of the model at every iteration and stop when it does not change anymore (i.e. variation is less than some threshold ε). The final loglikelihood will be returned as an additional output of the function.

Question 3.8

Print the loglikelihood for every EM iteration. What do you observe?

So far the number of clusters used to run EM was chosen equal to the number of clusters used to generate the data. In practice however the ideal number of clusters to choose is unknown. Choosing a large number of clusters artificially increases the likelihood by creating overspecific models. In order to find the ideal number of clusters that produces accurate but not overspecific models, one can use a criterion that penalizes models with a large number of parameters. The Bayesian Information Criterion (BIC) is one of these criteria. While BIC has some theorical justifications for the exponential family of parameter distributions whose categorical distributions are member of, one uses it here as an empirical criterion. The BIC score for a model with g clusters is:

$$BIC(g) = \quad 2 \log \mathcal{L}_D(\hat{\theta}^g_{MLE}) + k(g) \log(n) \tag{4}$$

where $\log \mathcal{L}_D(\hat{\theta}^g_{MLE})$ is the loglikelihood of the MLE estimator for a model with g clusters, $k(g)$ is the number of free parameters in a model of g clusters and n is the number of samples used to learn the model.

Question 3.9

Using the BIC criterion, implement a function `findBestModel(D)` that returns the best model with the optimal number of clusters matching data `D`. Test on different examples. What happens if a cluster (in the real model) has a low mixture coefficient?