Introduction

In statistics, the expectation–maximization (EM) algorithm is an iterative method for finding maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. For this reason EM is frequently used for data clustering, verification and identification of the speaker (biometric tasks), author profiling based on his documents, automatic document categorization, and many more applications.

em4gmm[1] is a toolkit to work with Finite Gaussian Mixture Models. In fact, it is a very fast and parallel C implementation of the clustering Expectation Maximization (EM) algorithm for estimating Gaussian Mixture Models (GMMs), with some extra important improvements. This is a full list of features that this toolkit contains:

- Fast learning of Gaussian Mixture Models (GMMs) using multidimensional data.
- Fast merge similar components to simplify the learned Gaussian Mixture Model.
- Fast simple classification score for a group of data (the data can be compressed).
- Obtain a detailed classification for each sample of a group of data (auto-clustering).
- Use a World Model to normalize classification scores (for biometric tasks).
- Possibility of set the number of threads to use on the execution of any task.

In order to compile this toolkit, you need first compile and install the zlib library[2] from their website, or using your preferred software distribution channels (aptitude, yum, macports, etc) in order to install it (with the dev packages). On some systems this library can be installed by default.

Then, on Mac OS X and Linux distributions you can simple use the ***make*** command on the system shell to compile it, and then ***sudo make install*** to install it on yout system (by default on /usr/bin). We recommend the use of the lastest version of the GCC compiler (because the code generated by LLVM is, for now, much slower).

1 https://github.com/juandavm/em4gmm/
2 http://www.zlib.net/

Basic Usage

This toolkit has two different utilities: ***gmmtrain*** and ***gmmclass***. You can use the ***gmmtrain*** utility in order to train a Gaussian Mixture Model from a feature data file, and the ***gmmclass*** to classify (obtain the score/loglikelihood) one feature data file.

The data files used by this software are very simple. They are plain text files of decimal numbers, with a header, and one line per sample vector. This is an example of a very short multidimensional feature data file:

11	***4***									
1025	***7706***	***6830***	***5571***	***4169***	***2858***	***1809***	***1094***	***688***	***500***	***417***
1147	***5755***	***6636***	***6234***	***4118***	***4593***	***2750***	***3649***	***774***	***1568***	***1104***
932	***5381***	***5567***	***5175***	***3613***	***3499***	***2429***	***2536***	***652***	***913***	***337***
838	***6401***	***5961***	***5277***	***4418***	***3468***	***2516***	***1644***	***921***	***391***	***74***

On the header, the first number are the dimension and the second the number of samples. The sample's vectors can be integers or decimals (using "." as separator), and the dimensions must be space-separated. Also, you have an example of a data file on the ***dat*** directory of this project. If you want to save disk space you can compress data files using gzip format (.gz file). A simple way to do this compression is using the gzip Linux or Mac Os X command.

You can train a Gaussian Mixture Model using the ***gmmtrain*** utility on a feature train file. This will learn the Model from the provided data, and save it into a file. This are all the options that can be used with this utility:

Usage: gmmtrain <options>

Required:

-d file.txt|file.gz	***file that contains all the samples vectors***
-m file.gmm	***file used to save the trained mixture model***

Recommended:

-n 2-524228	***optional number of components of the mixture***

Optional:

-u 0.0-1.0	***optional merge threshold based on similarity***
-s 0.0-1.0	***optional stop criterion based on likelihood***
-i 1-10000	***optional maximum number of EM iterations***
-t 1-256	***optional maximum number of threads used***
-h	***optional argument that shows this message***

Using the special ***-u*** parameter you can activate the merge feature that will try to merge the similar components of the mixture, leaving the model as simple as possible, avoiding overtraining, and speeding up the classifier. Also, yo can obtain the score/loglikelihood of one feature test file using the ***gmmclass*** utility:

Usage: gmmclass <options>

Required:

-d file.txt|file.gz	***file that contains the samples vectors***
-m file.gmm	***file of the trained model used to classify***

Recommended:

-w file.gmm	***optional world model used to smooth***
-r file.json	***optional file to save the classify log***

Optional:

-t 1-128	***optional maximum number of threads used***
-h	***optional argument that shows this message***

The standard process is to train a model for each class, and then classify at the class with highest score. You also can obtain a detailed analysis of the classify process using the ***-r*** option on the ***gmmclass*** utility. This option lets you to know the component assigned to each sample, and the occupation of the components. This large report is useful to do automatic clustering of some data on a predetermined number of clusters.

Speaker Identification

In order to make a speaker identification system, we need to record and extract acoustic features with any external toolkit (for instance the HTK[3] or TLK[4] toolkit). We need to create one Gaussian Mixture Model for each speaker, so first of all we put all the acoustic features for each speaker together on one file (with the required format for this toolkit).

Then, we need to train all the Gaussian Mixture Models using the ***gmmtrain*** utility of this toolkit for each speaker like this:

gmmtrain -d speakerN.feas -m speakerN.gmm -s 0.001 -u 0.95 -n 256

The ***-d*** parameter specify path to the acoustic features extracted before, the ***-m*** is the place to save the model, the ***-s*** parameter is a stop criterion based on the improvement, the ***-u*** allows to the toolkit merge similar components (this may be slow and use high memory), and the ***-n*** is the initial number of components of the mixture. Of course, you can change the command to try to make the best model possible to your system, but this command probably can be a good startup point.

When you have all the models prepared, you can easy use them to identify the speaker. First of all the speaker must read or tell some short text, that must be recorded and transform into an acoustic feature like before (using the required format for this toolkit).

The last step is, of course, identify the speaker. In order to do it, you will need to use the ***gmmclass*** utility of the toolkit in order to get the scores for each trained model. You can use it with a command like that:

gmmclass -d unknown.feas -m speakerN.gmm

The score for this feature will be printed on the standard output. The real speaker of this new recorded acoustic features are the speaker with the highest score, when we use this command with the same acoustic features on all the speaker models.

This are the basics of the speaker identification task using the em4gmm toolkit.

3 http://htk.eng.cam.ac.uk/
4 http://www.translectures.eu/tlk/

Final Notes

Issues and Bugs

Do you have a bug or a feature request? Do not worry, open a new issue[5]. But please, before opening any new issue, search on existing the yours in order to avoid duplicates. And thanks you for your contribution!

Authors

Juan Daniel Valor Miró[6].

License

5 https://github.com/juandavm/em4gmm/issues/
6 http://www.juandaniel.es/