Lab 4 - Gaussian Mixture Models

Monday, June 22, 2009
4:58 PM

PURPOSE

Goal: By the end of this lab, you will understand the how to use GMM models - a probabilistic clustering and "soft classification" technique.

Unlike the previous labs, this lab is more free form and allows you greater room to explore these new techniques. Special thanks and credit given to Dan Ellis at LabROSA / Columbia University for allowing modification and use of his labs and practicals. This lab includes commands and comments leveraged from his "Music Content Analysis" analysis.

DEMO

To view a demo of the EM algorithm, type: **demgmm1**

TODAY'S PROJECT

Create one of the following:

A simple set of artist classification GMMs.
A simple set of genre classification GMMs.
A simple set of < other audio-based >classification GMMs.
...using the audio files posted in your \scratch folder.

1. Here's some pseudo code for what you'll want to do:

```
% Extract features from entire audio file using frames
for loop for all files of a given artist / given genre / classifier
    Read in audio file
    for loop for frames of audio
        Extract features for currentFrame (use all the features we've learned thus far, including MFCCs!)
    end
end

% Create GMMs using the below procedure. You'll create 1 GMM per class that you want to classify. (e.g., 1 GMM per artist, or per genre)
```

If you're reading mp3 files, you'll benefit from the Reading MP3 Files script.

2. To create GMMs in Matlab's "netlab", you perform a few steps.

Below is a trivial example of creating 2 GMMs - one to represent kick drum samples and one to represent snare drum samples. Obviously, you'll need to modify the code to your own uses.

1. **Create default GMMs** specifying the number of dimensions (features) and number of Gaussian components in our mixture

   For example, for 2 features and 2 components:

```
% Initialize diagonal-covariance models for PDFs to be estimated
gm1 = gmm(2,2,'diag'); % GMM # 1
gm2 = gmm(2,2,'diag'); % GMM # 2
options = foptions;      % default optimization options
options(14) = 5;         % 5 iterations of k-means in initialization
gm1 = gmminit(gm1, features1, options); % Initialize from feature data for data type 1
gm2 = gmminit(gm2, features2, options); % Initialize from feature data for data type 2
```

2. **Run EM procedure** to estimate mixture parameters and build the GMMs.

```
options = zeros(1, 18);
options(1) = 1;
```

```
options(5) = 1;
options(14) = 25;          % Number of iterations
gm1 = gmmem(gm1, features1, options);          % run EM
        while isnan( gm1.centres(1) )          % Handles case where we get a NaN as result of EM
          gm1 = gmminit(gm1, features1, options); % Initialize from feature data for data type 1
          gm1 = gmmem(gm1, features1, options);
        end
gm2 = gmmem(gm2, features2, options);
        while isnan( gm2.centres(1) )          % Handles case where we get a NaN as result of EM
          gm2 = gmminit(gm2, features2, options); % Initialize from feature data for data type 1
          gm2 = gmmem(gm2, features2, options);
        end
```

Now you have successfully modeled some GMMs to be your classifiers.

Warning:

The netlab implementation of EM sometimes allows provides you centers that are -Inf, +Inf, or NaN. This seems to be the result of some occasional divide by 0 errors, which can arise when you are feature -extracting digital silence (which has a time domain value of 0.0). If you experiencing any errors or potential weirdness, please check out the value of your GMM centers by typing: gm1.centres. This can be a good sanity check to make sure that the center points of your GMM are not NaNs or Inf.

3. **Testing:**

Now, take your test material (say, a test song) and feature extract it.
We would pass the **testing_features** into the GMMs, querying the likelihood of it agreeing with each GMM.

For example, to obtain the likelihood that a given test feature vector for your GMMs, we'll take a feature vector(s) store in **testing_feature**.

Determine how these PDF estimates perform as classifiers by calculating the log of the ratio of the likelihoods:

```
% Likelihoods of new feature data fitting under model
likelihoodgm1 = gmmprob(gm1,testing_features);
likelihoodgm2 = gmmprob(gm2,testing_features);
% Log likelihood ratio
loglikelihood = log(likelihoodgm1 ./likelihoodgm2 )

% Emperically, if likelihoodgm1 > likelihoodgm2 , then their ratio will be > 1, hence the log will be > 0
%               If likelihoodgm1 < likelihoodgm2 , then their ratio will be < 1, hence the log will be < 0

% Try to classify based on a likelihood threshold  (items that most likely belong to gm1 are labeled "1")
mean ( (loglikelihood > 0) == 1 )
```

When do we run these likelihood calculations?

Here are a few examples:

- We could build many GMMs which are modeled after audio samples. (say, "kick" , "snare", and "hi hat") Then you could compare any arbitrary sample (i.e. it's feature vector) against each of these classifiers. The GMM with the highest log-loglikelihood is the most probable output.

- We could classify **each frame** in a piece of audio separately against the GMMs, obtaining loglikelihoods for each frame. A challenge with this approach is that the frame-by-frame classifications vary very rapidly. (For example, try it and plot the result of the loglikelihood over time.) One way around this is to take the mean-value value of the loglikelihoods over all frames, and call this the description of the audio file.

- We could compare **one distribution against other GMMs** (distributions) using a distance measure like EMD (Earth Mover's Distance), KL-divergence, centroid distance, etc. You would do something like this if you were comparing a GMM of a given song (all of the frames of a song) against the GMM

The simplest to understanding is the "centroid distance", which is the Euclidean distance between the means of the Gaussian models.

To do this, you would use the function **dist2** to get the distance between **gm1.centres and gm2.centres.**

Alternatively, we can use one the many statistical methods for computing distances between probability distributions. Since many of these methods are computationally intensive, one of the best (and fastest) is provided for you. (It's closely related to KL divergence.)

distance = ppk (gm1, gm2)

where gm1 and gm2 are netlab GMMs.

Note that the distance scores returned are relative, so it only makes sense to look at comparing the distances of multiple mixtures against each other and determining the best match. (such as 1 song worth of frames vs. 10 artist mixtures)