

Lab 2

Speaker Recognition Using GMMs

Prelab (20 points)

Implementing GMMs is difficult for some students. Consequently, this prelab exercise asks you to complete a partially written GMM module. The code has been placed on rohan and the GMCS Windows lab (~mroch/lib/cs682/gmm-partial and [\\cathouse\speech\gmm-partial](http://cathouse/speech/gmm-partial) respectively). The main entry point for training a GMM is *gmmCreate*. The function *gmmPr* returns the probability of an observation with respect to a model. Note that *gmmCreate* relies on *gmmPr*.

The usage of both functions and their children are described by the comments. Read through the functions making sure that you understand what they are doing and complete the missing functionality. The files *f_gauss.m*, and *gmmCreate.m* contain missing code as indicated by the comments that you should complete. Note that *gmmCreate* relies on *f_gauss*.

To ensure that you can perform the lab even should you encounter difficulties with the Gaussian mixture model, the main part of the lab provides an interface to a C library that implements a GMM. The C library is significantly faster than the GMM module that you will use in this lab, and it is highly recommended that you use the provided code even once you complete the prelab. Parts of the C library are used for the initialization of the GMM, so be sure to read the appendix on how to setup and configure the library.

To verify that the code functions correctly, you may want to compare the results of the GMM in the next part with the one that you complete on a small set of data of low dimensionality. One way to do this is to read the training data for one speaker, truncate the feature vectors to a low dimension, and then train your model. Compare it to the model that is created by the given code. Note that the provided code limits the GMMs to using diagonal covariance matrices, so you will need to set the *DiagonalCovar* flag to true. Do not be surprised if you do not see an exact match as minor differences in the code may produce different rounding errors, but it should be similar.

Lab (80 points)

In this experiment, you will create a closed-set text independent speaker recognition system using Gaussian mixture models and the TIMIT speech corpus. The TIMIT corpus is available both on rohan (~mroch/corpus/timit) and in the GMCS Windows lab ([\\cathouse\speech\corpora\timit](http://cathouse/speech/corpora/timit)). The README.DOC (text file, not Microsoft Word) contains a general description of TIMIT that you should read. Each of the 630 speakers has recordings of ten sentences, 2 of which were intended to highlight dialectical

differences (labeled SA), 5 of which provided good phoneme coverage (SX), and the last 3 which were intended to describe phonetic diversity (SI). There are 192 female and 438 male speakers. In order to reduce the time to train and test, you can limit yourself to testing 100 speakers of either gender.

TIMIT was originally designed for speech recognition experiments and is divided into train and test partitions with no overlap of speakers between the two. As we are using it for speaker recognition, we will ignore the train/test division and make the train/test partition on a per speaker basis. You will use the 5 SX sentences for training and test with the remaining five (SA/SI) sentences. Run each experiment with 16, 32, and 64 codewords.

Within the timit directory is another matlab directory which contains two functions designed to load the feature files: `timit_female.m` and `timit_male.m`:

```
speakers = timit_female;
```

`speakers{N}` is a cell array of sessions associated with female subject N.

Each cell contains a list of the names of the files (relative to the TIMIT base directory) for a given speaker. It will always be the case that you will see the sa sentences followed by the si sentences followed by the sx sentences. Thus one could select the five SX sentences of speaker 3 as follows:

```
speakers{3}{[6:10]}
```

Run experiments for the first 100 speakers. The included code initializes the GMM by creating a VQ codebook using the LBG algorithm (Linde et al., 1980) and setting the mean and variance of each Gaussian based upon the maximum likelihood estimators from the vectors in each partition of the codebook. Remember that the LBG algorithm performs binary splitting, so the most natural numbers of mixtures are powers of two although the code uses ad-hoc methods for other numbers of mixtures.

the resultant GMM testing with a minimum of 16, 32, and 64 mixture GMMs. You should be able to reuse much of your code from the previous experiment, and you are welcome to use code from the previous solution. Instructions for using the provided GMM code is provided and described in the appendix.

You should write a report describing your experiments. The report should contain an abstract and the following sections: introduction (describing what you will be doing), methods (indicating how the system works and the equations used for training and testing the GMM), results (describe the corpus¹ you are using, the results of your experiments including confidence intervals and confusion matrices), and a summary (your observations on the results). You should cite your sources. Remember that failure to put things in your own words is plagiarism *even if you cite* and can cause you to fail

¹ Remember to read the TIMIT readme files.

the course. Excluding figures, the report should be around 3-4 pages in length and in no case should exceed 5 and a half pages. a third of the points each will be assigned to your code, the contents of your report, and the style of your report. As the report counts for a significant portion of your grade, do not leave it for the last minute. As always, code should be submitted to blackboard, and you are free to reuse any of the solution code from previous assignments.

Appendix: Using the provided GMM code

As we will see when we study continuous-observation hidden Markov models (HMMs), a GMM can be thought of as a hidden Markov model with a single state. Three directories (folders) of Matlab code are provided that implement HMMs and their supporting routines. They are in `~mroch/matlab` on rohan and [\\cathouse\speech\matlab](http://cathouse/speech/matlab). Do not copy these directories to your home directory. Like the corpora, they can be used in place.

Some of the files in the directories are Matlab executable (mex) files. Matlab provides a means to load C/C++ library code into Matlab, and the bottlenecks of the library routines have been recoded as mex files for speed. They are version and architecture specific to the version of Matlab in the lab and on rohan. Copying these to a different machine architecture or a different version of Matlab² will not work. Unless you are an advanced Matlab user who has worked with mex files before, ***it is strongly discouraged that you spend time trying to port this to your home machine*** as the amount of effort it would require for an inexperienced user is probably as great as the assignment itself. I am willing to answer a question or two about porting this code, but I do not have the resources to provide extensive support.

To access the library code, you will need to modify Matlab's path so that it looks in the sigproc, hmm, and vq modules of research code that is used by my lab.

If you create a matlab directory, the UNIX Matlab will look for a startup.m file which is initialized when Matlab begins. Put the following lines in your startup.m:

```
% Add to path - UNIX (modify for Windows)
path(path, '~mroch/matlab');
path(path, '~mroch/matlab/audio/hmm');
path(path, '~mroch/matlab/audio/vq');
path(path, '~mroch/matlab/audio/sigproc');
```

For Windows, you can either source a similar file or modify a shortcut to start in the directory. The following functions can be used to create a model and determine the log likelihood of a test utterance.

```
Model = hmmCreateModel(States, Mixtures, Features, Info)
```

² Matlab 2007a is the current version in the Windows lab, Matlab 2008a for Solaris on rohan.

Features must be row vectors. As `spReadFeatureDataHTK` returns column vectors, you will need to use the transpose operator to put the feature vectors in a format that `hmmCreateModel` can use. The largely undocumented variable `Info` is an optional structure that can be used to control some of the parameters. The `MaximumIterations` field can be used to control the number of iterations. For your experiments, set `Info.MaximumIterations` to 5.

Example: Create a GMM with 16 mixtures

```
Info.MaximumIterations = 5;  
SpeakerModel = hmmCreateModel(1, 16, Features, Info);
```

```
LogLikelihood = hmmViterbi(Model, Features)
```

This computes the log likelihood of a feature set with respect to a specific model.

References

Linde, Y., Buzo, A. and Gray, R. M. (1980). An algorithm for vector quantizer design. In *IEEE Transactions on Communications*, vol. COM-28, pp. 84-95.