



Lab Project 1

In this assignment, you will create a speech recognition system which recognizes single digits between 0 through 9 and the word “oh” which is frequently used in place of zero. To complete this assignment, you will need to use a copy of the TIDIGITS corpus. While the audio data is available, features have been pre-extracted for you. Directions on how to access the corpus follow below. All files referred to throughout these directions are relative to the tidigits directory.

Your first task is to write a training routine which loads the data for each digit, and then trains a VQ model for that digit using the k-means algorithm described in class. For each digit, we will need to load examples of feature vectors that represent the word. One feature vector is generated every 10 ms, so the number of feature vectors depends upon the word duration. Since vector quantization does not model time dependencies, the feature vectors for each class can all be concatenated and the model trained.

Once all models are trained, it would be a wise idea to save them (doc save). The next stage is to write a test function which loads in words (or at least their corresponding feature vectors) and assigns a class based upon the lowest distortion codebook. As the real word is known, we can determine if the classification was correct or not. As tests are processed, generate a confusion matrix to track where the problems are. It would also be useful to note the actual test filename so that you can perhaps listen to some correct and incorrect classifications and see if you can find any patterns. At the end, the function should report the overall error rate and a plot of the confusion matrix (see the unit 1 slides), both of which should be turned in. This process should be repeated for at least three different codebook sizes. Start by trying codebooks between 40 to 80 codewords.

You can find a copy of TIDIGITS stored on the Windows lab server [\\cathouse\speech](#) or on rohan in `~mroch/corpora/`. See the course documents on Blackboard for instructions on how to mount remote directories under Windows. In either case, a directory called tidigits contains all of the files that are needed for this assignment. You are permitted to make a copy of this corpus, but be forewarned that you will only be using the isolated digits produced by adults. Note that if you are working on the lab machines or rohan, you *should not* copy these directories into your own account, as doing so will most likely exceed your disk quota.

The following subdirectories of tidigits are relevant for this assignment:

- doc – Contains files which indicate how the speech was collected. This will be useful to understand how the files are organized. Note that the Linguistic Data Consortium renamed the files when TIDIGITS was published. The speech (and feature) files consist of a sequence of symbols in the set $\{1,2,3,4,5,6,7,8,9,o,z\}$ which correspond to the speech produced by the speaker. Letter “o” represents the word

“Oh” and “z” “zero.” A letter follows the digits to distinguish multiple utterances of the same digit string.

- `train/test` – Audio files in the SPHERE audio file format. You can read these with `wavesurfer` [1]. See the course FAQ for information about using `wavesurfer` to listen to SPHERE files. The male and female directories of TIDIGITS are present. An example of a typical file is: `train/woman/il/4b.wav`.
- `mfc` – Feature files (Mel-filtered cepstral coefficients) corresponding to the SPHERE files. The filenames are similar to the SPHERE files, but end in `_01.mfc`. Example: `mfc/train/woman/il/4b_01.mfc`.
- `matlab` – Contains Matlab functions that you can use in the assignment. Consult the file comments (help function name) for more details. It is suggested that you read the code.

`spReadFeatureDataHTK.m` – Reads a feature file.

`tidigits_isolated.m` – Returns cell arrays which can identify the training and test data. The cell entries are relative to the `mfc` directory and do not contain extensions, e.g: `'train/woman/il/4b'`. Read about strings in the help browser for information on concatenating strings. You may also find the function `fullfile` and/or `sprintf` useful.

`tidigits_splitfiles.m` – Takes a cell array of filenames and returns a new cell array where each entry is all of the files associated with a class. The first entry is all files containing “one”, the second “two”, etc. The last entries are “zero” and “oh” respectively. This is useful to partition files by class. See the file header for details, but there is a gender flag which permits one to restrict files to a specific gender, and it is recommended that you do so although it might be interesting to compare performance of gender specific for both genders and gender neutral classifiers if you have the time.

As you develop this project, there are a number of common pitfalls that you may wish to avoid:

1. **As vector quantizers do not model temporal structure, the training feature data can be concatenated**, e.g. `Features = [Features ThisFeatureSet];` You can easily set up a loop for grouping features by initializing the variable you use for collection with the empty matrix `[]`. **Do not concatenate the test feature data as you need to test each production individually.**
2. **Only load data as you need it.** Do not try to load the data for all of the classes at once. Reuse the same variable for the next class so that you are not incrementally loading everything. Doing so is likely to result in your machine thrashing for a very long time.
3. **When you know the size of a matrix that will be constructed incrementally, preallocate it using the zeros command.** Adding one row at

a time to a matrix causes Matlab to copy the matrix each time and will result in your program running *orders of magnitude more slowly*. Note that for your training data, you will not be able to preallocate.

4. **Design your code modularly.** Have your training function return a set of VQ models and save it using Matlab's save command (see helpdesk). This will prevent you from having to retrain each time you have an error or need to shut down Matlab to do something important such as sleeping or eating while developing your test code.
5. **Think before you write.** A little intelligence can go a long way. As an example, when you're developing your code, prune your dataset so that you have a smaller number of samples. This will significantly increase the speed and let you find any errors more quickly. I also believe in **commenting as you write**, as it makes you think about what you are doing.
6. **Do not underestimate the power of Matlab's tools to help you work effectively.** The profiler and symbolic debugger are both worth taking the time to learn, and can save you considerable time. Both are explained in the helpdesk. In addition, if you use the Matlab editor as opposed to an external one (e.g. emacs, vi, or pico), the right column is highlighted when Matlab detects potential problems such as unused variables.
7. **Do not wait for the last minute to start the assignment. Do not be shy about using office hours (or making an appointment if you have a conflicting class).** While I expect you to have done some thinking on your own before coming, I strongly encourage you to come for help if you need it.

In addition to the code for this project, you should write a 3-5 page report describing your experiment. You should discuss the objective, methods, results, and your conclusions. Tables and figures should be captioned and well labeled. A significant portion of your grade will be based upon the report, so do not wait until the last minute to write it. Remember, it must be in your own words, heavy paraphrasing from the book or web is plagiarism.

References

- [1] K. Sjölander and J. Beskow, "Wavesurfer," 2006, <http://www.speech.kth.se/wavesurfer/>, accessed Feb 15, 2009.