Lab 2 - "My first audio classifier"

Monday, June 22, 2009
4:11 PM

PURPOSE

My first audio classifier: introducing K-NN! We can now appreciate ***why*** we need additional intelligence in our systems - heuristics can't very far in the world of complex audio signals. We'll be using Netlab's implementation of the k-NN for our work here. It proves be a straight-forward and easy to use implementation. The steps and skills of working with one classifier will scale nicely to working with other, more complex classifiers.

We're also going to be using the new features in our arsenal: cherishing those "spectral moments" (centroid, bandwidth, skewness, kurtosis) and also examining other spectral statistics.

SETUP

To read **MP3 files** into Matlab, you have to do some additional setup on your local Linux machine. Please follow these steps to set this up:
https://cm-wiki.stanford.edu/wiki/Reading_MP3_Files

SECTION 1

First off, we want to analyze and feature extract a small collection of audio samples - storing their feature data as our "training data". The below commands read all of the .wav files in a directory into a structure, **snareFileList**.

1. Use these commands to read in a list of filenames (samples) in a directory, replacing the path with the actual directory that the audio \ drum samples are stored in.

```
snareDirectory = ['~/Matlab/audio/drum samples/snares/'];
snareFileList = getFileNames(snareDirectory ,'wav')

kickDirectory = ['~/Matlab/audio/drum samples/kicks/'];
kickFileList = getFileNames(kickDirectory ,'wav')
```

2. To access the filenames contained in the cell array, use the brackets **{}** to get to the element that you want to access.

   For example, to access the text file name of the 1st file in the list, you would type **snareFileList{1}**
   Try it out:

```
snareFileList{1}
```

   When we feature extract a sample collection, we need to sequentially access audio files, segment them (or not), and feature e xtract them. Loading a lot of audio files into memory is not always a feasible or desirable operation, so you will create a loop which loads an audi o file, feature extracts it, and closes the audio file. Note that the only information that we retain in memory are the features that are extracted.

2. Create a loop which reads in an audio file, extracts the zero crossing rate, and some spectral statistics. Remember, you did some of this work in Lab 1 - feel free to re-use your code. The feature information for each audio file (the "feature vector") should be stored as a feature array, with columns being the features and rows for each file.

   Or in Matlab, for example:

```
featuresSnare =

 1.0e+003 *

  0.5730   1.9183   2.9713   0.0004
  0.4750   1.4834   2.4463   0.0004
  0.5900   2.2857   3.1788   0.0003
  0.5090   1.6622   2.6369   0.0004
  0.4860   1.4758   2.2085   0.0004
  0.6060   2.2119   3.2798   0.0004
  0.4990   2.0607   2.7654   0.0004
  0.6360   2.3153   3.0256   0.0003
  0.5490   2.0137   3.0342   0.0004
  0.5900   2.2857   3.1788   0.0003
```

   In your loop, here's how to read in your wav files, using a structure of file names:

```
[x,fs]=wavread([snareFileDir snareFileList{i}]);    %note the use of brackets for snareFileList
```

   Here's an example of how to feature extract for the current audio file..

```
frameSize = 0.100 * fs;  % 100ms
currentFrame = x(1:frameSize)
featuresSnare(i,1)  = zcr(currentFrame);
[sc,ss, sfm]=getSpectralFeatures(currentFrame,fs);    % 1st value is spectral centroid, 2nd value is spectral spread, 3rd v alue is spectral
flatness measure
featuresSnare(i,2:4) = [sc,ss, sfm];
```

3. First, extract all of the feature data for the kick drums and store it in a feature array. (My example, above, is called "fe aturesKick")
4. Next, extract all of the feature data for the snares, storing them in a different array.
   Again, the kick and snare features should be separated in two different arrays!

OK, no more help. The rest is up to you!

SECTION 2

Building Models

1. Examine the feature array for the various snare samples. What do you notice?

2. Since the features are different scales, we will want to normalize each feature vector to a common range - storing the scaling coefficients for later use. Many techniques exist for scaling your features. We'll use linear scaling, which forces the features into the range -1 to 1.

   For this, we'll use a custom-created function called **scale.** Scale returns an array of scaled values, as well as the multiplication and subtraction values which were used to conform each column into -1 to 1. Use this function in your code.

   ```
   [trainingFeatures,mf,sf]=scale([featuresSnare; featuresKick]);
   ```

Building a k-NN

4. Build a k-NN model for the snare drums in Netlab, using the function **knn.**

   We'll the implementation of from the Matlab toolbox "netlab":

   ```
   >help knn
   NET = KNN(NIN, NOUT, K, TR_IN, TR_TARGETS) creates a KNN model NET
         with input dimension NIN, output dimension NOUT and K neighbours.
         The training data is also stored in the data structure and the
         targets are assumed to be using a 1-of-N coding.

         The fields in NET are
          type = 'knn'
          nin = number of inputs
          nout = number of outputs
          tr_in = training input data
          tr_targets = training target data
   ```

   Here's an example...

   ```
   model_snare = knn(4,1,1,trainingFeatures,[ones(10,1); zeros(10,1)]);
   ```

   This k-NN model uses 4 features, has 1 input (the label), 1 output (the label), and takes in the feature data via a feature array called trainingFeatures.

   The last input, in this example, is an array of ones and zeros, which looks like this:

   ```
   1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0  0  0
   ```

   These labels indicate which sample in our feature data is a snare, vs. a non-snare. The k-NN model uses this information to build a means of comparison and classification. It is ***really important*** that you get these labels correct - because they are the crux of all future classifications that are made later on. (Trust me, I've made many mistakes in this area - training models with incorrect label data.)

6. **Create a script which extracts features for a single file, re-scales its feature values, and evaluates them with your kNN classifier.**

   Evaluating samples with your k-NN
   Now that the hard part is done, it's time to through some feature data through the trained k-NN and see what it outputs.
   In evaluating a new audio file, we need to extract it's features, re-scale them to the same range as the trained feature values, and then send them through the knn.

   Some helpful commands:

   ```
   features = rescale(features,mf,sf) ;  % This uses the previous calculated linear scaling parameters to adjust the incoming features to the same range.

   model_output_snare = knnfwd(model_snare , features)
   ```

   Once you have completed function, first, test it with your training examples. Since a k-NN model has exact representations of the training data, **it will have 100% training accuracy** - meaning that every training example should be predicted correctly, when fed back into the trained model.

   Now, test out with the examples in the folder "test kicks" and "test snares", located in the drum samples folder. These are real-world testing samples...

   If the output labels "1" or "0" aren't insightful for you, you can add an if statement to display them as strings "snare" and "kick".

BONUS (ONLY IF YOU HAVE EXTRA TIME...)

1. While it's interesting to test one file at a time - try to evaluate an entire **file folder** of audio files.
   Create a script which extracts features for a **folder of audio files**, re-scales their feature values, and evaluates them with your kNN classifier.

   For a slick experience, check out the commands uigetdir and uigetfile -- these allow your matlab scripts to present a GUI browser to query for file locations.

2. Create a new classifier, using other audio samples. (Yay! No more drum samples!)
   Use the audio files downloaded in your **~\scratch\Instrument Samples** folder.
   Choose two of these folders and create a new k-NN to be able to distinguish between them…

NOTE!

We didn't separate audio into frames here, or use equally-sized time segments for feature extraction.
That's not a very good practice. Why?