Lab 2 "My first audio classifier"

Tuesday, July 22, 2008
1:05 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.

On a Matlab-related note, we'll be using a new variable-type: a **structure**. A structure allows you to neatly store matrices, arrays, strings, cells, or other structures - within them.

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.

   In Linux,

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

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

   (Note: Due to the case-sensitive nature of Unix names, trying to list a directory for *.wav files yields a different result for *.WAV files. Keep this in mind if you are missing files at any point…)

   Note2: You must remove the . and .. from the file names

   or in Windows...

   ```
   snareDirectory = ['C:\Documents and Settings\Jay\My Documents\Presentations\Semantic Audio Analysis - AES 2008
   \audio\drum samples\snares\'];
   snareFileList = dir([snareDirectory '*.wav']);

   kickDirectory = ['C:\Documents and Settings\Jay\My Documents\Presentations\Semantic Audio Analysis - AES 2008
   \audio\drum samples\kicks\'];
   kickFileList = dir([kickDirectory '*.wav']);
   ```

   To access the filenames contained in the structure, add **.name** to the end of the structure element that you want to access.

   For example, to access the text name of the 1st file in the list, you would type **snareFileList(1).name**

When we feature extract a sample collection, we need to sequentially access audio files, segment them (or not), and feature extract 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 audio 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 first 4 spectral moments (centroid, bandwidth/spread, skewness, kurtosis). 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:

   featuresKick =

1.0e+003 *

0.2050	0.9821	0.0001	1.3512	1.3512
0.1500	0.6210	0.0001	0.2961	0.0001
0.1200	0.3616	0.0001	0.2638	0.2745
0.1350	0.8094	0.0001	0.8344	0.2745
0.2200	0.6347	0.0001	0.2745	0.0001
0.1750	0.5363	0.0001	0.1884	0.2745
0.1900	0.5670	0.0001	0.2530	1.3512
0.1350	0.7203	0.0001	0.3338	0.2745
0.1950	0.7785	0.0001	1.2328	0.0001
0.1850	0.5144	0.0001	0.1830	0.2745

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

```
[x,fs]=wavread([snareDirectory snareFileList(i).name]);
```

Here's an example of how to feature extract using the MIR Toolbox:

```
a=miraudio(x);                          % creates a MIR Toolbox object, containing the numerical vector x
features(1,1) = mirgetdata(mirzerocross(a));    % gets the feature data in numerical form, to store in a matrix
```

First, extract all of the feature data for the kick drums and store it in a feature array. (My example, above, is called "featuresKick")

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!

How do you extract the 4 spectral moments (centroid, bandwidth/spread, skewness, kurtosis)? Check it out in the MIR Toolbox Guide - you should find how to use them, along with some easy examples. While you are there, feel free to skim the Guide and observe it's capabilities.

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. Plot some of the features against each other to see how they visually group. This type of visualization is a great sanity check - it not only verifies that your feature calculations look OK, but they verify that your audio collection is homogenous. (To illustrate, compare your kick drum collection to your snare collection - look at it when you plot the clusters or look at the feature array. If one sample were accidentally loaded into the other, it would surely stick out of the plotted cluster.)

   In a single figure, plot the first feature against the second feature, for all of the audio files.
   If you find it necessary, you might want to "hold" the plot - which allows you to freeze it from being erased.
   You can use the "hold on" and "hold off" commands to do this.

3. 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.**

   >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.)

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.

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

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)

>help knnfwd
KNNFWD Forward propagation through a K-nearest-neighbour classifier.

Description
[Y, L] = KNNFWD(NET, X) takes a matrix X of input vectors (one vector per row) and uses the K-nearest-neighbour rule on the training data contained in NET to produce a matrix Y of outputs and a matrix L of classification labels. The nearest neighbours are determined using Euclidean distance. The IJth entry of Y counts the number of occurrences that an example from class J is among the K closest training examples to example I from X. The matrix L contains the predicted class labels as an index 1..N, not as 1-of-N coding.

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!)
   I just uploaded a new folder of various instrument samples into : "usr\ccrma\courses\mir2008\audio**Instrument Samples**"
   Choose two of these folders and create a new k-NN to be able to distinguish between them…

Help!

Here's how to plot feature 1 vs feature 2 in an array, using red 'x's as the marker.

```
plot(features(:,1),features(:,2),'rx')
```

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?