Lab 4 - Cluster Lab

Thursday, July 24, 2008
7:53 AM

PURPOSE

Sometimes, an unsupervised learning technique is preferred. Perhaps you do not have access to adequate training data. Or perhaps the classifications for the training data's labels events are not completely clear. Or perhaps you just want to quickly sort real-world, unseen, data into groups based on it's feature similarity. Regardless of your situation, clustering is a great option!

First, we'll start with some easy framing and MFCCs….

SECTION 1 SEGMENTING INTO EVERY N ms FRAMES

Segmenting: Chopping up into frames every N seconds

Previously, we've either chopping up by the location of it's onsets (and taking the following 100 ms) or just analyzing the entire file.

Here's another technique for your arsenal that is good for analyzing entire songs, phrases, or non-onset-based audio examples.

To easily chop up the audio into frames every, say, 100ms, we can say:

```
>> f=mirframe(x,'Length',0.100,'s','Hop',1);    % this is a 100ms frame size with NO overlap
```

Very often, you will want to have some overlap between the audio frames - taking an 100ms long frame but sliding it 50 ms each time. To do a 100ms frame and have it with 50% overlap, try:

```
>> f=mirframe(x,'Length',0.100,'s','Hop',0.5);
```

More possible options include:

mirframe(x,..., 'Length', w, wu,'Hop', h, hu):

w is the length of the window in seconds (default: 0.05) and wu is the unit, either
's' (seconds, default unit),
or 'sp' (number of samples).

h is the hop factor, or distance between successive frames (default: half overlapping: each frame begins at the middle of the previous frame)
u is the unit, either
'/l' (ratio with respect to the frame length, default unit)
'%' (ratio as percentage)
's' (seconds)
or 'sp' (number of samples)

What do you DO with these frames? How do you access their raw numerical data, or either playing them out load or further feature extracting them as you've done before? Easy!

Option 1: Put the frame data into an array. As you can see, there are 87 frames now stored in "myFrames"

```
>> myFrames=mirgetdata(f);
>> size(myFrames)

ans =

     4410       87

>>
```

Option 2: Feed it directly into the MIRToolbox feature extractors. As many of you've noticed, you can feed a lot of stuff into the mir features, without needing a FOR loop. For example...

```
>> centroids = mirgetdata(mircentroid(f));

Computing mirspectrum related to x...
```

Computing mircentroid related to x...

```
>> size(centroids)

ans =

   87    1

>>
```

SECTION 2 MFCC

Load an audio file of your choosing from the folder \scratch\mir2008
Use this as an opportunity to explore this collection.

Test out MFCC to make sure that you know how to call it.

```
a=miraudio(x);  % x is your audio signal, that you read in from wavread

f=mirframe(x,'Length',0.100,'s','Hop',1);    % this is a 100ms frame size with NO overlap

mfccs_all = mirgetdata ( mirmfcc(a,'Rank', 1:12) )    = returns MFCC coefficients for 1 -> 12
                                We throw away the 0th coefficient since it's not too useful.


mfccs_delta         = mirmfcc(a,'Rank', 1:12,'Delta', 1)   = the delta-MFCC
mfccs_deltadelta  = mirmfcc(a,'Rank', 1:12,'Delta', 2)   = the delta-delta-MFCC
```

SECTION 3 CLUSTERING

Now we're going to try clustering with a familiar bunch of audio files and code. Sorry, the simple drum loop is going to make an appearance again. However, once we prove that it works - you can experiment with other audio collections that are posted.

1. Create a new .m file for your code.

2. Load simpleLoop.wav. (Sorry - we'll use other audio files soon! It's best to start simple - 'cuz if it don't work for this file, we have a problem.)

3. Segment them into 100ms frames based on their onsets.

4. Now, feature extract the frames using ***only*** zero crossing and centroid. Store the feature values in one matrix for both the kick and the snares… remember, we don't care about the labels with clustering - we just want to create some clustered groups of data.

5. Scale the features (using the scale function) from -1 to 1

6. It's cluster time! We're using NETLAB's implementation of the kmeans algorithm.
   Use the kmeans algorithm to create clusters of your feature. kMeans will output 2 things of interest to you:
   (1) The center-points of clusters. You can use the coordinates of the center of the cluster to measure the distance of any point from the center. This not only provides you with a distance metric of how "good" a point fits into a given cluster, bu t this allows you to sort by the points which are closest to the center of a given frame! Quite useful.

   (2) Each point will be assigned a label, or cluster #. You can then use this label to produce a transcription, do creative stuff, or further train another downstream classifier.

Here's the help function for kmeans:

> help kmeans

```
 KMEANS          Trains a k means cluster model.

       Description
        CENTRES = KMEANS(CENTRES, DATA, OPTIONS) uses the batch K-means
```

algorithm to set the centres of a cluster model. The matrix DATA
represents the data which is being clustered, with each row
corresponding to a vector. The sum of squares error function is used.
The point at which a local minimum is achieved is returned as
CENTRES. The error value at that point is returned in OPTIONS(8).

[CENTRES, OPTIONS, POST, ERRLOG] = KMEANS(CENTRES, DATA, OPTIONS)
also returns the cluster number (in a one-of-N encoding) for each
data point in POST and a log of the error values after each cycle in
ERRLOG. The optional parameters have the following
interpretations.

OPTIONS(1) is set to 1 to display error values; also logs error
values in the return argument ERRLOG. If OPTIONS(1) is set to 0, then
only warning messages are displayed. If OPTIONS(1) is -1, then
nothing is displayed.

OPTIONS(2) is a measure of the absolute precision required for the
value of CENTRES at the solution. If the absolute difference between
the values of CENTRES between two successive steps is less than
OPTIONS(2), then this condition is satisfied.

OPTIONS(3) is a measure of the precision required of the error
function at the solution. If the absolute difference between the
error functions between two successive steps is less than OPTIONS(3),
then this condition is satisfied. Both this and the previous
condition must be satisfied for termination.

OPTIONS(14) is the maximum number of iterations; default 100.

Now, simply put, here are some examples of how you use it:

```
% HOW MANY CLUSTERS DO YOU WANT TO FIND????
numCenters = 2;          % the size of the initial centers; this is passed to k-means to determine the value of k.
numFeatures = 5;       %  replace the "5" with however many features you have extracted
centers = zeros(numCenters , numFeatures );          % inits centers to 0

% setup vector of options for kmeans trainer
options(1) = 1;
options(5) = 1;
options(14) = 50;   % num of steps to wait for convergence

% train centers from data
   [centers,options,post] = kmeans(centers , your_feature_data_matrix , options);

% Centers contains the center coordinates of the clusters - we can use this to calculate the distance for each point in the
distance to the cluster center.
%  Post contains the assigned cluster number for each point in your feature matrix.  (from 1 to k)
```

7. Now, modify your script to play and plot the audio files which are closest to the center of your clusters.
   This hopefully provides you with which files are representative of your cluster.

Helpful Commands for sorting or measuring distance:

```
d = dist2( featureVector1 ,featureVector2 )   % measures the Euclidean distance betw/ point 1 and point 1
```

DIST2 Calculates squared distance between two sets of points.

Description
D = DIST2(X, C) takes two matrices of vectors and calculates the
squared Euclidean distance between them. Both matrices must be of
the same column dimension. If X has M rows and N columns, and C has
L rows and N columns, then the result has M rows and L columns. The
I, Jth entry is the squared distance from the Ith row of X to the

Jth row of C.

[y,ind] = sort( )

>> help sort

```
SORT   Sort in ascending or descending order.
  For vectors, SORT(X) sorts the elements of X in ascending order.
  For matrices, SORT(X) sorts each column of X in ascending order.
  For N-D arrays, SORT(X) sorts the along the first non-singleton
  dimension of X. When X is a cell array of strings, SORT(X) sorts
  the strings in ASCII dictionary order.

  Y = SORT(X,DIM,MODE)
  has two optional parameters.
  DIM selects a dimension along which to sort.
  MODE selects the direction of the sort
    'ascend' results in ascending order
    'descend' results in descending order
  The result is in Y which has the same shape and type as X.

  [Y,I] = SORT(X,DIM,MODE) also returns an index matrix I.
  If X is a vector, then Y = X(I).
  If X is an m-by-n matrix and DIM=1, then
     for j = 1:n, Y(:,j) = X(I(:,j),j); end

  When X is complex, the elements are sorted by ABS(X).  Complex
  matches are further sorted by ANGLE(X).

  When more than one element has the same value, the order of the
  elements are preserved in the sorted result and the indexes of
  equal elements will be ascending in any index matrix.

  Example: If X = [3 7 5
                   0 4 2]

  then sort(X,1) is [0 4 2  and sort(X,2) is [3 5 7
                     3 7 5]                   0 2 4];
```

[y,ind] = sortrows ( )

```
SORTROWS Sort rows in ascending order.
  Y = SORTROWS(X) sorts the rows of the matrix X in ascending order as a
  group. X is a 2-D numeric or char matrix. For a char matrix containing
  strings in each row, this is the familiar dictionary sort.  When X is
  complex, the elements are sorted by ABS(X). Complex matches are further
  sorted by ANGLE(X).  X can be any numeric or char class. Y is the same
  size and class as X.

  SORTROWS(X,COL) sorts the matrix based on the columns specified in the
  vector COL.  If an element of COL is positive, the corresponding column
  in X will be sorted in ascending order; if an element of COL is negative,
  the corresponding column in X will be sorted in descending order. For
  example, SORTROWS(X,[2 -3]) sorts the rows of X first in ascending order
  for the second column, and then by descending order for the third
  column.

  [Y,I] = SORTROWS(X) and [Y,I] = SORTROWS(X,COL) also returns an index
  matrix I such that Y = X(I,:).
```

[y,ind] = sortrows (featureData_from_a_particular_cluster, clusterNum)

8. Write a script to audition all of the audio files that were categorized as Cluster # 1. Do the same or Cluster # 2.

9. Now, modify the script to play the audio files sorted via their distance away from a specified cluster center.

For example, you will be playing the audio file at the center of a cluster, then playing the next closest audio file to the center of the cluster, etc.

10. Let's add MFCCs to the mix. Extract the 12 MFCCs using the code that you wrote. Add those to the feature vectors, along with zero crossing and centroid. We should now have 14 features being extracted - this is started to get "real world"!

11. Repeat the above steps (5-8) with our new and improved MFCCs. You probably won't notice a difference, but at least it didn't break, right? Let's try it with the some other audio to truly appreciate the power of timbral clustering.

12. Now that we can take ANY LOOP, onset detect, frame, feature extract, and cluster it, **let's have some fun.**

    Take any audio file and use the above techniques break it up into clusters. Listen to those clusters.

    Some rules of thumb: since you need to pick the number of clusters ahead of time, listen to your audio files first.

    - You can break a drum kit or percussion loop into 3 - 6 clusters for it to segment well. More is OK too.
    - Musical loops 3-6 clusters should work nicely.
    - Songs - lots of clusters for them to segment well. Try 'em out!

During this lab, I'll be logging into your machine and posting A LOT more audio files for you in your scratch folders. Make sure to check there for updates and new audio files. (Yes, that Karaoke music will be posted.)

P.S

Note the talented efforts of Mike (and Jason?) to create a script which imports filenames regardless of the case sensitivity. Additionally, the script removes the annoying . and .. from the filelists.

The script(s) are located at:

http://ccrma.stanford.edu/~mw/

For example:
http://ccrma.stanford.edu/~mw/shared/getFileNames.m
>> [audioFiles] = getFileNames(directory, ext)