Assignment 3: Machine Learning and Buffer-based Time-Domain Audio

CS 4347: Sound and Music Computing

due Wednesday 18 February 2015, 11:59 pm

0. This assignment will use the same "music / speech" dataset that we used in asn 1.

1. Make a copy of your python program from assignment 2 and add 1 new feature. Your program should still calculate features based on the entire audio file at once, and output an ARFF file, but it does not need to output any PNG files.

   Mean Absolute Deviation (MEAN-AD):

$$X_{\text{MEAN\_AD}} = \frac{1}{N}\sum_{i=0}^{N-1}\left(\left|x_i - \frac{1}{N}\sum_{j=0}^{N-1} x_j\right|\right)$$

2. Make a copy of the previous program. Modify it such that it will:

   - Read the collection file
   - Load each wav file (divide by 32768.0) and split the data into buffers of length 1024 with 50% overlap (or a hopsize of 512) Only include complete buffers; if the final buffer has 1020 samples, omit that buffer.

     Hint: the starting and ending indices for the first few buffers are:

Buffer number	start index	end index (not included in array)
0	0	1024
1	512	1536
2	1024	2048
...		

     I recommend that you use the numpy "array slice" feature:

     ```
     for i in range(num_buffers):
         start = ...
         end = ...
         buffer_data = whole_file_data[start:end]
     ```

     This should give you 1290 buffers of length 1024. If you are comfortable with linear algebra and numpy, the program will run significantly faster if you store this as a single 1290x1024 matrix and avoid any loops during the feature calculations.
   - Calculate 5 features for each buffer: RMS, PAR, ZCR[1], MAD, MEAN_AD. This means that $N = 1024$ in the equations, instead of the previous $N = 661500$. You should end up with 1290 feature vectors of length 5, or a single 1290x5 matrix.

[1] You may be concerned about the boundaries between buffers when calculating the ZCR: "if buffer i ends with a negative value and buffer $i + 1$ begins with a positive value, what happens to that zero-crossing?". The answer is that for this assignment, we strictly use the formula from assignment 1. This means that zero-crossings that are exactly between buffers are never counted. This is not an ideal solution since it means that the "total" number of zero-crossings can change based on the buffer size. However, this vastly simplifies the programming, so we adopt this solution. Some well-known audio analysis tools such as Marsyas do the same thing.

- After calculating the features for each buffer, calculate the mean and uncorrected sample standard deviation for each feature over all buffers for each file. You should end up with a single feature vector of length 10, or a 1x10 matrix for each file.
- Output the data to a new `ARFF` file (with a different filename from the whole-song `ARFF` file) with the same format as before, but the header should be:

  ```
  @RELATION music_speech
  @ATTRIBUTE RMS_MEAN NUMERIC
  @ATTRIBUTE PAR_MEAN NUMERIC
  @ATTRIBUTE ZCR_MEAN NUMERIC
  @ATTRIBUTE MAD_MEAN NUMERIC
  @ATTRIBUTE MEAN_AD_MEAN NUMERIC
  @ATTRIBUTE RMS_STD NUMERIC
  @ATTRIBUTE PAR_STD NUMERIC
  @ATTRIBUTE ZCR_STD NUMERIC
  @ATTRIBUTE MAD_STD NUMERIC
  @ATTRIBUTE MEAN_AD_STD NUMERIC
  @ATTRIBUTE class {music,speech}
  ```

  and the data lines should be:

  ```
  @DATA
  RMS_MEAN1,PAR_MEAN1,ZCR_MEAN1,MAD_MEAN1,MEAN_AD_MEAN1,RMS_STD1,PAR_STD1,ZCR_STD1,MAD_STD1,MEAN_AD_STD1,music
  ...
  ```

  Concretely, the `@DATA` section should be:

  ```
  @DATA
  0.057447,3.234547,0.191595,0.037308,0.044607,0.027113,0.397827,0.036597,0.018317,0.021898,music
  0.026583,2.590328,0.039416,0.016488,0.018977,0.013284,0.384978,0.015575,0.011143,0.012273,music

  ...
  0.062831,2.822102,0.082504,0.042271,0.049270,0.032323,0.799688,0.070962,0.027540,0.029103,speech
  ```

3. Classify the results of the whole-song and buffer-based `ARFF` files in Weka with `trees.J48` and save the results. Choose at least 1 other algorithm, and classify both `ARFF` files with that algorithm, again saving the output.

   Write a file called `classifications.txt` which compares the results of these 2 algorithms on the two `ARFF` files. The file should begin with 1-3 sentences stating which was the best results and summarizing its performance in comparison to the other results. This should be followed by the Weka results for the four cases.

4. Upload your ARFF file to:

   `http://cs4347.smcnus.org`

   This will automatically grade the values you calculated. If any mistake is found, please check your program and resubmit – you are welcome to submit as many versions as you wish before the submission deadline.

   Submit a zip file containing your program's source code (as a `.py` file), the ARFF file, and an optional README.txt file to the same website.

   - You may use anything in the python standard library, numpy (including pylab / matplotlib), and scipy libraries. No other libraries are permitted.

If you are familiar with python and understood the lecture, this should take about 1 hour.
Grading scheme:

- **3/6 marks**: correct ARFF file (automatically graded by computer).
- **2/6 marks**: readable source code (good variable names, clean functions, comments when needed).
- **1/6 marks**: Discussion of weka output