15-323/15-626 Spring 2019

Project 6: Music Information Retrieval – Genre Classification

Due May 3

1. Overview

In this project, you will use what you learned from the lecture about music understanding and classification to build a genre classification system. The purpose of the project is to give you some hands-on experience with audio feature extraction and machine learning, as well as some widely used toolbox and software (we will introduce Weka[1] for machine learning, jAudio[2] for audio feature extraction). Machine learning is both powerful and applicable to a wide range of problems, including music systems and information processing!

This project is in fact very open-ended. There is no standard correct answer (there are expectations of what the results should look like though). This write-up is only intended to provide all the possible approaches to accomplish the task, and the remainder is left for you to explore. There will be some benchmark objectives, which are pretty simple given what the write-up introduces. You need to finish all of these to get the full score. For those of you who want to explore deeper, we also provide as much help as possible, and a significant amount of bonus points will be rewarded depending on what you end up with.

2. Specifications and Implementation Guide

Specifications:

We will provide audio files for 4 different genres (classical, jazz, metal, and pop), chosen from the 10-genre dataset GTZAN Genre Collection[3] (Yes, this is the dataset used in the milestone paper by Tzanetakis et al.[4] as Roger mentioned in the lecture) widely used by music information retrieval community for genre classification task. You will extract different features using jAudio, and save as the arff[5] format appropriate for Weka (jAudio will do this automatically). With the data, choose different classifiers in Weka. Evaluate the results using cross-validation.

Implementation guide:

1. Arff file format

Before talking about feature extraction, we will first briefly introduce the arff file format used by Weka as this will help explain some of the data processing in next few sections.
Arff format is actually pretty straightforward to read. It begins with a header including relation names and feature (called attribute in Weka) definitions:

[1] http://www.cs.waikato.ac.nz/ml/weka/

[2] https://sourceforge.net/projects/jaudio/

[3] http://marsyas.info/downloads/datasets.html

[4] http://webhome.cs.uvic.ca/~gtzan/work/pubs/tsap02gtzan.pdf

[5] http://www.cs.waikato.ac.nz/ml/weka/arff.html

```
@RELATION <relation-name>
@ATTRIBUTE <attribute-name>  <data-type>
```

The data type can one of the following:

- `NUMERIC`
- `{class1, class2, class3…}`
- `STRING`
- `DATE [<date-format>]`

The data begins with the indicator `@DATA`. Each line is an instance containing values for each defined attributes. A very simple arff file with 2 attributes and 4 instances will like this:

```
@RELATION test

@ATTRIBUTE temperature   NUMERIC
@ATTRIBUTE day   {sunny, cloudy, snow}

@DATA
90,sunny
28,snow
56,cloudy
43,sunny
```

2. jAudio tutorial.

Download jAudio from link provided. Open the jar file, the main UI will show up. The right column shows all the available features – you will have the chance to explore different features. However, in genre classification, the instrumentation/voices will definitely be an important factor. An important feature originated in speech recognition called mel-frequency cepstrum coefficients[1] (MFCC), not covered in the lecture, is designed to represent the sound texture, which is also used as one of the main features in Tzanetakis et al. jAudio provides MFCC, along with many variations, as their names suggest.

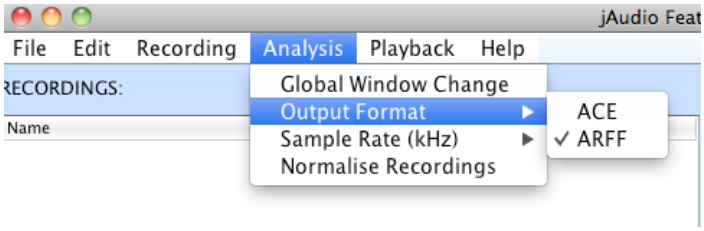
Besides MFCC, there are some other features that will help as well. Since the instrumentation/voices are better reflected in the frequency domain, features like spectral flux, spectral centroid, and zero-crossings will also help with the task. For more features, Tzanetakis et al. will be a good reference.

In order to extract features using jAudio, a few configurations are needed:

1) Change the output file format.

Since we will use Weka, the output format should be changed from xml to arff. This can be done in Analysis -> Output Format as shown below. And you should also change the save path at the bottom left accordingly.

[1] http://en.wikipedia.org/wiki/Mel-frequency_cepstrum

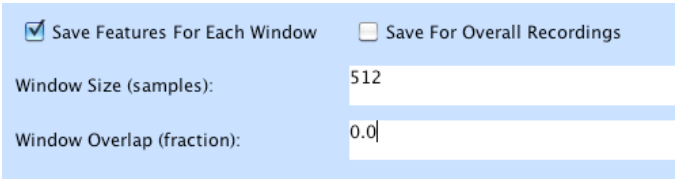


2) Change the sample rate to match the audio files.

The sample rate for audio files should be changed to 22.05kHz, which also can be found in Analysis -> Sample Rate (kHz).

3) Change the features calculating scheme.

As talked about in class, normally the audio features are extracted within a very small time window (usually called frame) with overlapping. For each audio file, you will have many instances of the features. In the jAudio UI, you should select 'Save Features for Each Window' as shown below.

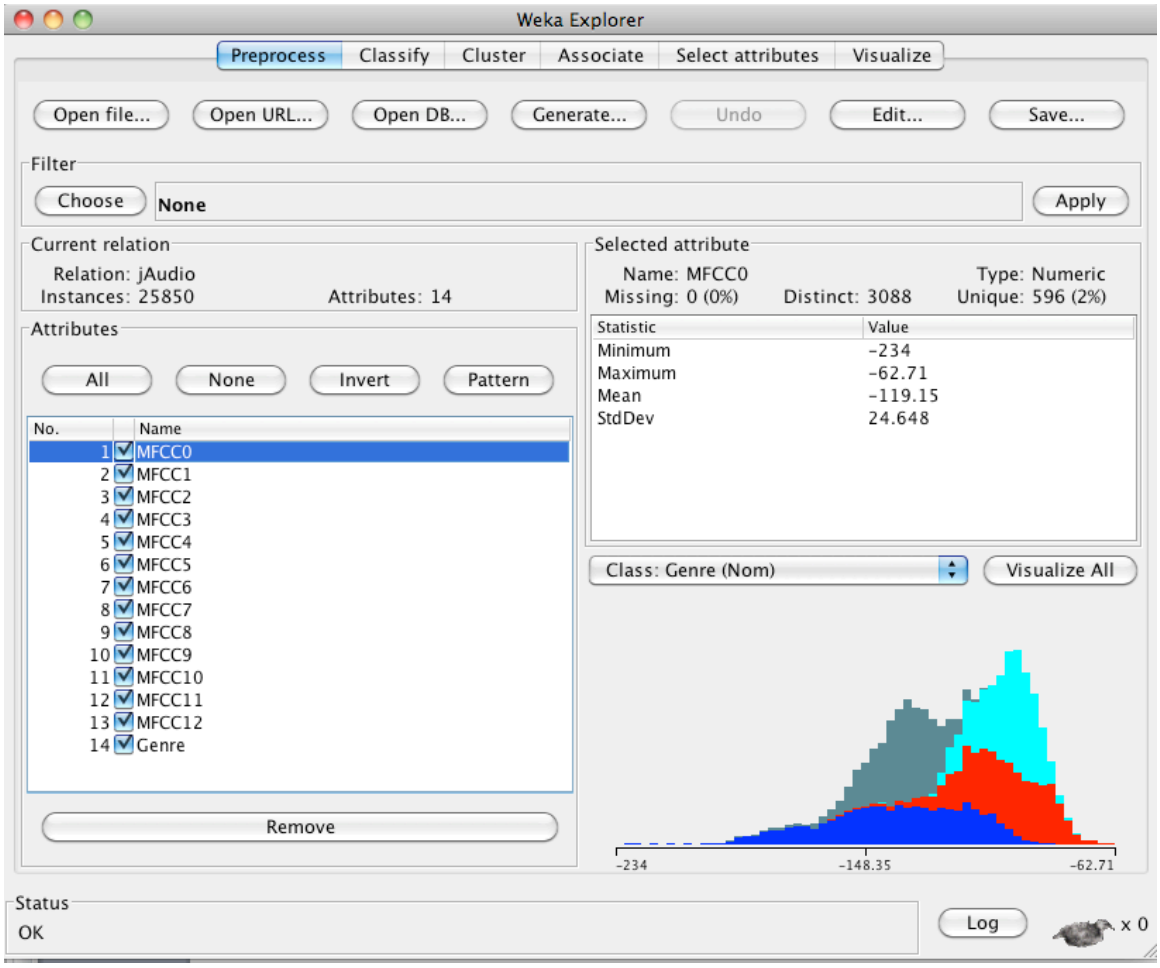


Notice that the window length and overlapping fraction are also configurable. You could try different windows sizes and overlapping fractions as well (the default 512 window size with 0 overlapping is OK).

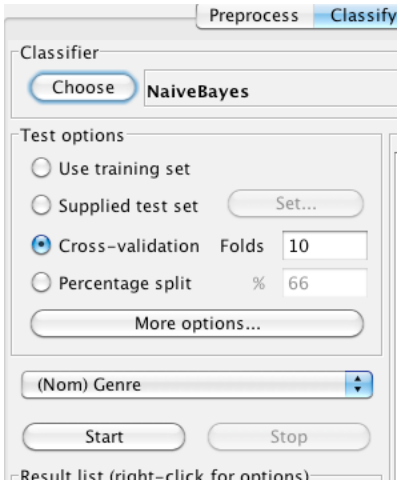
After these configurations, now you could start extracting features from jAudio by clicking on 'Add Recordings' button to select the audio files you would like to work on. We would recommend extracting features for each genre separately then manually combine the arff files for different genres into a final arff file, as the course staff failed to find an easy way to add 'class label' (in this case, genre) directly from UI as an attribute into the output arff file. Therefore, we provide a hacky helper code for you to add one more attribute to the arff file for a certain genre (it cannot add different genres in one arff file). The helper code is *Addgenre.jar*. In order to use it, you should first *cd* to the folder that contains the Addgenre.jar and the arff file. Then just type *java –jar Addgenre.jar <file.arff> <genre>*. You can of course write some hacky code by yourself once you fully understand the arff format.

3. Weka tutorial

After all the data preparation, now we can start learning! Start Weka and choose Explorer. In the preprocessing step, load data from arff file by clicking 'Open file…' button. Then in the 'Attributes' panel, be sure to include *All* the attributes. By clicking on the different attribute, the data distribution will be virtualized on the right side. One of the 14 dimensions of my example data MFCC and the genre labels will look like this in Weka:



Now we are ready to go to classify step. Below is a set up which uses 'Genre' as the class label, train a Naïve Bayes classifier (bayes.NaiveBayes), evaluating with 10-fold cross-validation. Click 'Start' and wait for the magic!



Try a different classifier as well (not limit to the following):

- Sequential Minimal Optimization (functions.SMO): a fast implementation of Support Vector Machine (SVM). SVM is a $O(n^2)$ algorithm, thus it's still much slower than Naïve Bayes. But, as the math underlying SVM indicates, SVM is the best linear classifier in general.
- Logistic Regression (functions.Logistic): A learning algorithm which has similar property with Naïve Bayes. As proved by Andrew Ng[1], given infinite data, Logistic Regression will always outperform Naïve Bayes. But Naïve Bayes converges faster than Logistic Regression. It's interesting to compare the accuracy between them in our case.
- Neural Networks (functions.MultilayerPerceptron): Old-fashioned machine learning algorithm. You could see this algorithm will take forever to run, as it's trained using a sophisticated iterative algorithm (part of the reason why it's not favored by machine learning people). But the non-linear natural of neural networks could give even better result then linear SVM (SVM can also be modified for non-linear classification). For this algorithm, Weka may run out of the Java heap size. This page[2] will help.

The results provided by Weka are too detailed for us to observe for this project. Just pay attention to accuracy (overall performance) and confusion matrix (the classification results within each genre). Does the result make sense to you?

4. A few comments

As you may see, this project is very open-ended, and the only expectation from us is to hope you feel the power of machine learning and like it. Feel free to do the project with software/language other than what provided here, and a few suggestions: MATLAB (be sure to check Dan Ellis' chroma code[3]), Marsyas[4], and Vamp Plugin[5]. If genre classification task cannot satisfy you (as it cannot for the researchers nowadays), free feel to contact the course staff for more advanced and state-of-art music information retrieval project. Have fun!

3. Grading Criteria:

1) Benchmark: Extract MFCC from audio files. Train a Naïve Bayes classifier or some other classifier for genre classification using the provided data, evaluate by 10-fold cross-validation. Report the results and your observation on the overall accuracy and confusion matrix.
2) 15-623 students should train on 3 classifiers and describe differences observed, if any. Which classifier works best? How would you explain your results? (Speculation that indicates an understanding of what you are doing is expected; rigorous analysis is not expected.)
3) Bonus points are assigned flexibly, a few suggestions:
    a. Logical and rigorous experiments design and report writing.
    b. Adding different features and different features combinations. If a feature doesn't perform well alone, you should not expect it to get better when used with other features.

---

[1] http://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf

[2] http://weka.wikispaces.com/Java+Virtual+Machine

[3] http://labrosa.ee.columbia.edu/matlab/chroma-ansyn/

[4] http://marsyas.info/

[5] http://www.vamp-plugins.org/

c. Write your own code instead of using jAudio and/or Weka if you have prior knowledge on signal processing and/or machine learning. Using other software /language also counts based on the difficulty.

4. Hand-in Instructions:

Please put any codes you wrote and a short report <***andrew_id***>.pdf into a zip file, named <***andrew_id***>_p6.zip. Submit the zip file via Autolab.