Intuition

Kevin Montag's CS229 Project, Fall 2009

Intuition is a library for classifying and transforming audio based on a number of different learning algorithms, supervised and unsupervised, as well as a client of that library which utilizes it for real-time audio processing. The client was built as a final project for Music 256A (Music, Computing, and Design), so I will not discuss it at length in this paper; rather, I will focus on the library and the learning algorithms it employs. The library provides functionality to classify audio files using logistic regression; to find centroids of perceptual features in an audio file using k-means clustering; and to set parameters for audio production "plugins" (equalizers, compressors, etc.) in real time using locally weighted linear regression. It also allows users to "mix and match" these algorithms in ways that are relevant to audio production. I will discuss each of these aspects of the library's functionality separately.

First, however, it will be useful to discuss the framework which the library uses for audio processing. Intuition is written in C++, and leverages a number of open source libraries for dealing with audio data at low levels. Audio files are accessed using libsndfile (http://www.mega-nerd.com/libsndfile/), and features are extracted from raw audio using libxtract (http://libxtract.sourceforge.net/) for perceptual features, and FFTW (http://www.fftw.org/) for spectral content. Intuition also provides an abstraction for transforming audio using arbitrary LADSPA plugins (http://www.ladspa.org). A large number of LADSPA plugins are freely available, with a wide range of functionality – everything from simple spectral filters to vinyl emulators and elaborate reverbs. Each plugin defines a set of parameters which can be modified to change its effect on audio. Audio data is abstracted via the `samples` class, which allows audio features to be lazily computed and cached. The abstract `feature` class is extended to represent a particular type of audio feature, and the library's learning algorithms generally accept collections of `feature` instances on construction, which they then use to extract training data from `samples` objects which are provided to them. This design allows features to be defined generically; the library implements wrappers to libxtract to extract a number of perceptual (for example, smoothness, rolloff, flatness, spectral skewness) and non-perceptual (for example, variance and average deviation) from audio, and then the learning algorithms can be applied to arbitrary collections of these features.

The `feature_value_set` class provides an abstraction for k-means clustering of audio data with respect to some set of features. The class accepts a set of features at construction, the user adds training data in the form of audio samples, and then the class computes the centroids for those samples using k-means clustering. Clustering can be employed to extract feature values which "represent" a collection of audio files – I will discuss later how this is employed by the library's regression algorithm.

The `classifier` class allows users to perform multinomial logistic regression to classify audio. Again, a set of features are specified at construction, and the user provides training data in the form of `samples` instances and associated labels. We can then compute parameters for the regression by optimizing the log-likelihood for our training set, and then use the parameters to classify new audio. This can be used for genre classification – the user specifies a collection of features which they feel are relevant to distinguish genres, and then a number of samples from different genres. The success of the algorithm is strongly dependent on feature selection.

The `regression` class abstracts the most complex of the employed learning algorithms implemented by the library – performing locally weighted logistic regression to determine parameters for LADSPA plugins that transform sound in desired ways. The user specifies a collection of features to use for training data, a collection of LADSPA plugins to use for processing input, and a set of "target" features which they wish new audio to have. These target features are specified as a `feature_value_set` instance – the idea is that the user can extract feature centroids from a song or collection of songs that she likes with respect to some perceptual feature (if she likes, for example, the flatness of a Johnny cash song), and then train LADSPA plugins to take input data and output something which shares the specified features. Training data is specified as audio samples, and then the `regression` instance extracts the desired input features from those samples, and finds the optimal LADSPA parameters to make those samples sound like the target output with respect to each centroid in the `feature_value_set` (here the optimal parameters are found “manually”; the method for finding optimal parameters is discussed below). The `regression` instance stores the training point as a (input features, plugin parameters) point, along with a label representing the target-feature centroid which could be best matched by the plugins (internally these labels are managed by a `classifier` instance). Then, given new input data, the `regression` computes the input parameters for that data, uses multinomial logistic regression to decide which target-feature centroid to try to approximate (that is, it predicts the centroid which it will be able to best approximate), and performs locally weighted logistic regression using the optimal values it discovered for that centroid. The selection of plugin parameters given new data happens in real time, and the client written for Music 256A provides an interface for receiving audio from other applications (using JACK, http://jackaudio.org) and transforming it to match the desired target features.

To compute optimal values for LADSPA plugins for training data, we wish to minimize the function:

$$J(\theta)=\sum_{i=1}^{n}\|{y_\theta}^{(i)}-\bar{y}^{(i)}\|_2^{2}$$

Here there are n target features, ${y_\theta}^{(i)}$ represents value the *i*th feature given inputs θ to the LADSPA plugin, and $\bar{y}^{(i)}$ represents the desired value of the *i*th target feature. This function is poorly behaved in general with respect to optima, since ${y_\theta}^{(i)}$ is a complex function of the LADSPA parameters; simple gradient descent therefore performs poorly. However, we can't just search the entire space of plugin parameters for the optimal set of values, since this space can be very large. Instead, we take the approach of generating a large number of random points in the parameter-space for the plugins, choosing the one which gives the smallest value for *J*, and *then* performing gradient descent with that parameter choice as an initial set of values. We approximate the partial gradient with respect to each parameter by varying that parameter slightly and recomputing *J* for the new set of parameters; by doing this for each dimension of the parameter space, we can obtain an approximate total gradient for the function, and use this to perform the descent.

As mentioned, this project was created somewhat in conjunction with my Music 256A project for this quarter. Professor Ge Wang offered a lot of great insight with respect to the existing libraries available for low-level audio processing, and the scope of the project as it pertains to practical applications.