Practical 3: Classifying Sounds

Emily Chen, Joanna Chung, Kevin Lee
Emails: emily-chen@, joannachung@, kevinlee01@college.harvard.edu
Camelot.ai usernames (Team machine_learners): emchen, jcat1, kevin

April 9, 2018

1 Technical Approach

Feature Extraction

Each observation in the given data set has 88,200 features. This makes training the usual models directly on the raw data difficult due to computational constraints as well as the challenges of high-dimensional data (number of features exceeds number of observations). These 88,200 amplitude samples, however, should not be treated as separate features. There is structure in the data as the measurements are sequential in time; for example, adjacent samples are highly correlated.

This suggests that we can try to reduce the dimensionality of the data. We do this by looking for features that capture important qualities of the sound. A paper on automatic sound recognition (http://ieeexplore.ieee.org/document/1716317/) suggested the following features:

- Mel Frequency Cepstral Coefficients: these represent the component frequencies of the sound as perceived by the human ear.
- Spectral centroid: a weighted average of the frequencies; measures the brightness of the sound
- Spectral flatness: measures how noise-like vs. tone-like the sound is
- Spectral rolloff: measures the right skewedness of the spectrum

These are computed with functions from the `librosa` library. We compute these features for overlapping 20 millisecond windows and then take an average. The recordings will change over time, so the smaller time windows captures features as the sound changes. Taking an average summarizes the characteristics of the whole sound. An average is a crude way of doing this but is a reasonable first approximation.

In addition, we also extracted the maximum, minimum, and average of the amplitude samples as a relevant feature for each audio file, with a hypothesis that the loudness of an audio clip is indicative of the type of sound.

Model Construction

To construct and evaluate our models, we first split the given "training" data set in two, yielding a training and a validation set. This new training set comprised roughly 90% of the original

data set, and the remaining 10 % became the validation set, on which we evaluated the different models.

To train our models, we made use of Python's Scikit-learn package and tested various types of classification models and model classes with the features already given to us:

- k-Nearest Neighbors (Model 1): We iterated through all possible combinations of the following hyperparameter values to maximize the accuracy score on our validation set:
  - k-values: 2, 5, 10
  - Weight function: neighbors weighted equally, neighbors weighted by the inverse of their distance from the point
  - Distance measure: Manhattan distance, Euclidean distance

  The combination that yielded the highest accuracy score was $k = 2$, neighbors weighted by distance, and the Manhattan distance measure.
- Random Forest (Model 2): We iterated through the following hyperparameter values:
  - Number of trees: 100, 500
  - Maximum number of features (to consider when looking for the best split, out of the total n features): $\sqrt{n}$, n
  - Maximum tree depth: 25, or nodes expanded until all leaves are pure
  - Class weight: class weights inversely proportional to class frequencies in the training data or weights constant across all classes

  The combination that yielded the highest accuracy score was 500 trees, maximum tree depth of 25, maximum number of features $\sqrt{n}$, and constant class weights.
- Logistic Regression (Model 3): We iterated through the following hyperparameter values:
  - Intercept: bias term included for the weights or no bias term
  - Number of cross-validation folds: 3, 5
  - Class weight: class weights inversely proportional to class frequencies in the training data or weights constant across all classes

  Here, we perform cross-validation while constructing our model, and select 3-fold cross-validation as the optimal process. We also include a bias term in our weights and employ L_2 regularization to encourage weights of smaller magnitude as a means of preventing overfitting and making the model more generalizable to new data. In addition, having constant class weights performs better than having balanced weights.
- Neural Network (Model 4): We iterated through the following hyperparameter values:
  - Loss activation function: identity, logistic, tanh, relu
  - L_2 penalty coefficient: .0001, .01, .1, 1

The combination that yielded the highest accuracy score was the logistic activation function with and penalty coefficient of $\alpha = 0.01$.

- Gradient Boosting (Model 5): We iterated through the following hyperparameter values:
  - Number of boosting stages: 50, 200, 500
  - Maximum depth of individual regression estimators: 2, 3, 5

  The combination that yielded the highest accuracy score was 500 boosting stages and a maximum depth of 5.

The accuracy scores on the validation set for each of the aforementioned models are denoted in Table 1 below:

Model	Accuracy
Baseline 1: Linear Regression	0.30321
Baseline 2: Random Forest	0.10643
Model 1: k-NN ($k = 2$)	0.83728
Model 2: Random Forest (500 trees, max depth 25)	0.98104
Model 3: Logistic Regression	0.80095
Model 4: Neural Network	0.87046
Model 5: Gradient Boosting (500 stages, max depth 5)	0.97314

Table 1: Trained models and their validation set accuracy scores

After tuning the hyperparameters of our model classes during our first iteration, we found that Random Forest (with 500 trees of max depth 25) was the best model, yielding a validation accuracy score of 0.98104 (as indicated by Table 2).

2 Results

The final model we used was Random Forest (with 500 trees of max depth 25) trained on the features we extracted from the amplitude samples for each audio clip. This model yielded the highest validation accuracy score of 0.98104 running on roughly 90% of the given training set.

After training this model and using it to make predictions on the given test set, we uploaded our predictions to Camelot. We obtained a test set accuracy score of 0.70683, beating both the baseline Linear Regression Model (0.30321) and Random Forest Model (0.10643) by a significant margin.

We note, however, that our test accuracy is non-trivially lower than our validation accuracy. In order to remedy this, we notice that 147 out of the 1000 audio clips in the test dataset consist of rows entirely composed of 0's, telling us that the test dataset is systematically different from the training dataset in some way, since the training set does not have this same issue. In practical implementations of our model, we would refrain from making a prediction or give the caveat that we make this prediction with a low degree of confidence. However, for the purposes of the

Camelot submission, we instead predict the most frequent sound `air_conditioner` (Class 0) for these datapoints.

3 Discussion

The main source of improvement in our accuracy scores came from feature engineering, over model selection. As shown in Table 1, accuracy scores across models were all within 20 percent of each other. Improvement over the course of extracting different features however, increased our accuracy score from below the random forest baseline (approximately 10 percent) to our final Camelot score of over 70 percent.

Our approach to feature engineering came from researching academic papers on audio classification measures to inform us of the options available. We compiled a long list of possibilities, including MFCC, band energy ratio, spectral flux, statistical moments, linear prediction ceptral coefficients (LPCC), short-time energy, and many others.

After researching what each of these features measure about the audio spectrum of different sounds, we visualized and listened to the different classes of sounds available in our training and test sets to prioritize which features would best exploit the differences in what we observed.

The first feature we chose to implement was MFCC, which maps the fourier transform of an audio signal to a scale that approximates the human auditory response, takes the log and cosine transform at each of the frequencies, and then finds the amplitudes of the resulting spectrum. We believed this was the most intuitive measure of sound, since it indicated the mixture of frequencies available in the sound, or the ingredients in the dish, which would differentiate the type of sound. MFCC was chosen over other cepstrums, because we noticed that sounds that may look more similar in spectograms still sounded very different, and MFCC maps the transforms to a human auditory-modeled scale. Just this addition scored us 0.48193 on Camelot, but we believed we could do better.

Because MFCC was so successful in improving our score, we further followed our method of finding features that quantify intuitive auditory experiences for how humans differentiate sounds. Rolloff was chosen as another type of mapping - this time of energy, and centroid and flatness were chosen as indicators of sound brightness and tonality vs. noise. The addition of these boosted our score to 0.70683.

After running Random Forest, we confirmed our prioritization of features by calculating the importance each feature had in the accuracy score. We used a librosa function to obtain a weight breakdown: MFCC - 0.065, rolloff - 0.055, flatness - 0.033, centroid - 0.031. This was in line with our intuition, and indicated that we were thinking about auditory classification in the right way, and had experimented successfully to significantly improve our accuracy score, well above the baseline.

One thing we would like to look into next is the observation that our validation set accuracy scores were higher than the scores we obtained on the Camelot test set. We do not believe this is an issue of overfitting, as we constructed the validation set to circumvent this. Future experimentation may include the following ideas:

- Generate a confusion matrix to see what types of mistakes the classifier makes: http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py
- Continue to add more feature extractions from the list we generated, following our intuition of prioritization, and make marginal accuracy improvements this way.
- Continue to explore why there are all-0 entries in test set.