Practical 3: Classifying Sounds

William Burke, Buse Aktas, Matthew Holman
wburke@g.harvard.edu, buseaktas@g.harvard.edu, mholman@cfa.harvard.edu
wburke, BuseAktas

April 6, 2018

1 Technical Approach

In this practical, we are trying to accurately classify urban sounds into these classes: air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren, and street music. The sound files in our training and test data consist of 88,200 columns, each column including an amplitude for an interval of time.

This is a very large data set, and it would be inefficient to try to include all 88,200 elements as features for thousands of different sounds both in our training data as well as in our testing data. Thus, we need some way to condense the information into smaller and more meaningful batches. Luckily, the LibROSA Python package provides a lot of tools for music and audio analysis.

We began this practical with exploratory data analysis. For a few examples of each category of sound, we used the LibROSA package to extract and display the spectrogram (showing the frequency spectrum as a function of time). Visually the similarity between the spectrograms of different samples of the same category of sound is readily apparent. We used this to guide our selection and development of features. We wrote a function to extract features and adopted a batching strategy so that we would not have to load all 6325 rows of 88,200 columns of training data into memory at the same time. To condense the features to a manageable size, we summed each of our features along the time axis.

We used the tools of the LibROSA package to extract 203 features from the spectrograms. We used a chunk size of 100 rows/batch, which took 1.5 hrs to calculate all of the features. The final training data consisted of 6325 rows of 203 columns, which was small enough to load into memory for training. The final test data was 1000 rows of 203 columns. In table 1, we list all the features we extracted, along with their dimensions.

- **Chroma:** After computing a chromagram from a power spectrogram, we take the normalized energy of each chroma for each frame, and take their mean throughout the time.
- **Mel:** We take the mean of the spectogram in the mel-scale across each time window.
- **MFCCS:** The mean of a mel-frequency cepstral coefficients sequence.
- **Spectral Centroid:** From the sound's magnitude spectogram determine the centroid and distribution of power as a function of frequency in each time window.
- **Spectral Bandwidth:** The mean of the spectral width at each time window. Specifically, the interval from 0.5*(max freq) to (max freq).
- **Spectral Contrast:** Similar to the mel-scale, but represents the averaged distribution across frequencies instead of the raw averaged mel-frequency data.

Summary of features used

Feature	Dimension
chroma	12
mel	128
mfccs	40
centroid	1
bandwidth	1
contrast	7
spectral flatness	1
rolloff	1
poly features	2
tonnetz	6
harmonic range	2
percussive range	2
TOTAL	203

Table 1: Features extracted using LibROSA library

- **Spectral Flatness:** Measurement scale from pure tone to random noise averaged across the time windows.
- **Spectral Rolloff:** The frequency below which which 85% of the energy is represented.
- **Poly Features:** Average coefficients of a polynomial fit to each time window.
- **Tonnetz:**Computes the tonal centroid features (tonnetz), first invented in 1739.
- **Harmonic Range:** The pair of minimum and maximum values from the harmonic spectrogram of a harmonic-percussive separation.
- **Percussive Range:** The pair of minimum and maximum values from the percussive spectrogram of a harmonic-percussive separation.

Given these features, we experimented with an initial set of scikit-learn classifiers including simple Logistic Regression, MultiLayer Perceptron Neural Network, a Support Vector Machine utilizing Singular Value Decomposition, as well as a Decision Tree. For the first set of experiments, we made sure to separate our data into a training set and a validation test in order to be able to gauge each model's bias and variance, and we started by using the default options and see which performed best. We specifically looked at a good performance in the training and validation accuracy alongside a small variation between the two accuracies to make sure that our model was not over-fitting. The categorization accuracy used was basically the percentage of correctly classified examples given the total number of examples:

$$\text{Categorization Accuracy} = \frac{\text{Number Correctly Classified Examples}}{\text{Total Number of Examples}}.$$

A more detailed explanation of our model selection process will be provided in the next sections.

Model	Training Acc	Validation Acc	Test Acc
Support Vector Classification	1.0	0.296	–
Decision Tree Classifier	1.0	0.958	–
Logistic Regression (LR)	0.995	0.985	0.3333
MLP Neural Net	0.950	0.918	0.3012
MLP - adding alpha, nodes, and layers grid search	0.994	0.990	0.3474
LR - adding cross validation and balanced weights	0.993	0.973	–
LR - adding C param grid search	0.995	0.985	0.3414
LR - adding solver type grid search	0.995	0.994	–
LR - adding solver = 'newton-cg', C=1)	1.0	0.990	0.4619
LR - adding solver = 'newton-cg', C=10	1.0	0.990	0.4719

Table 2: Accuracy of the Different Classification Models with Training, Validation and Test Data

2 Results

The classification methods and the performance we achieved with them on the training and validation sets, along with the performance achieved on the test sets are listed in table 2.

We exceeded the baseline performance, but it is clear from the results that we are over-fitting. The accuracy on the testing and validation sets is high, but the results on the test sets is much lower.

3 Discussion

We experimented with several simple supervised models before diving deeply into Multi-layer Perceptron (MLP) and the Logistic Regression (LR) models. As described in the technical approach section, we trained using a vector of 203 features. Given this set of features, we used a wide variety of classification methods, listed in table 2. We started with an initial experimentation with a Support Vector Classifier using Singular Value Decomposition,

Based on these initial results, we tested our simple LR and MLP models on Camelot. As in table 2, both performed roughly as well as the LR benchmark, so we decided to explore both further.

At a high level, we were concerned that our loss function might not be convex, leading to us getting stuck in a local minimum, and that we were most likely over-fitting to our data, which appeared to be very different from the data in the test set on Camelot. It was difficult to test if we were getting stuck in a local minimum because our training and validation accuracies were close to unity. As such, we hypothesized that there were most likely many models that would perfectly predict our training data. Moreover, we were massively over-fitting to patterns in the training data that were not present in the test set. Going forward, we resolved to use the highest possible regularization parameter when choosing between models with similar validation accuracy to reduce our over-fitting and pick the local model that would be least sensitive to noise in the training data.

One way to reduce over-fitting is to introduce a regularization parameter that penalizes the weights of the internal parameters, keeping them closer to 0. For MLP, the default regularization parameter, alpha, is 0.0001. We adopted an optimization strategy for alpha that was recommended in the scikit-learn documentation. [1]

[1] http://scikit-learn.org/stable/modules/neural_networks_supervised.html

That allowed us to optimize multiple parameters at once. We used GridSearchCV with 3-fold cross validation to search the range 10.0 ** -np.arange(1, 7) for the best alpha parameter, while also varying the number of nodes and layers. For MLP, the default for these parameters is 1 hidden layer with 100 nodes, i.e. $hidden\_layer = [100,]$ for a total of 3 layers. We used GridSearchCV to try with $hidden\_layer\_sizes = [50,], [100,], [200,], [400,], [600,], [1000,], [1000, 1000]$. See table 2 for the scores. The improved MLP model outperformed the LR benchmark, but only barely, so we moved on to the LR models.

Studying the LR scikit-learn documentation, we selected several parameters to optimize [2]. Since some of the classes, like 'air conditioner', in the in our training data were more prevalent than others, like 'gunshot', we felt is was important to use the default one-vs-rest training strategy with separate binary classifiers for each class and the l2 norm. To further ensure that we would correctly classify the 'gunshot' class, we opted to use balanced class weights for the penalty. Creating a pipline for 3-fold cross validation, we saw an immediate improvement in our performance. From there, we performed a grid search over the regularization parameter C, and tested our result on Camelot. Our score improved, but we could still do better.

We performed a grid search over the different solver options and regularization parameters and found that 'newton-cg' was both faster and had a higher validation score with C=0.01. Drawing on our suspicion that we were over-fitting, we set C=0.01 as the minimum on C and used grid search to find larger values. When C=1 worked much better on Camelot, we increased the regularization parameter again. Our best score came with a value of C=10. The final score was 0.4719

After completing the practical, we brainstormed further strategies for improving our performance on the urban sound classification task. Given the amount of time we spent optimizing our LR and MLP models, we would likely return to the feature engineering stage in future work. We had discussed alternative features to those provided by the LibROSA library, but ultimately did not go down that path. We are not experts in audio engineering, but other features we considered included the outputs of an unsupervised clustering task model as a Gaussian mixture model or the K-means algorithm. Another condensed representation we seriously considered was the sparse minibatch PCA algorithm. We could have decomposed our spectograms into their mxn eigenvectors and corresponding weights. Since the eigenvectors would span the space, we could have used the weights as features. Finally, we could have used time-sensitive features, which would have taken more space to store, but would contain more information. There are several tools in the LibROSA library to identify events in a time series that could have been useful for this task. Finally, we noticed that there are several rows of all 0s in the test dataset. Since there are not rows like this in the training data, we do not have a strong reason to classify them as members of one class or another. We discussed different approaches to classifying those rows, but did not include a special approach in our final solution. Since all zeros corresponds to no sound, we believe that these should go in an 11th class - 'nothing'. However, it is possible to figure out how those rows should be classified using Camelot - we could simply predict class '11' for all non-zero rows and then variously predict classes 1-10 for the remaining rows until we discovered the best distribution. This would improve our model performance, but not our understanding of machine learning.

[2] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html