# Video Classification Using a Histogram of Noisemes Representation of Audio Tracks

*Author:*
Tushar Pankaj

*Advisors:*
Gert Lanckriet
Emanuele Coviello

Completed at:
Statistical Visual Computing Lab, UCSD
9500 Gilman Drive
EBU1, Room 5101
La Jolla, CA 92093-0407

Summer 2014

# Video Classification Using a Histogram of Noisemes Representation of Audio Tracks

Tushar Pankaj

Summer 2014

## 1 Introduction

In this report, we discuss the results of a set of experiments in which we seek to classify videos using their audio tracks. We wish to create a mid-level feature representation for audio tracks by classifying small windows of the audio tracks into groups called noisemes [2]. This report focuses primarily on creating a feature vector for a classifier which classifies portions of audio files as belonging to a noiseme. We also discuss to a lesser extent the results of classifying videos using a histogram over the noisemes detected in the audio tracks.

## 2 Procedure

In this section, we first describe how we collect data from the Freesound [6] database. We then describe how we extract features from the audio files and create a histogram of noisemes representation for each audio file. We test multiple methods of representing each audio file as a feature vector until we find a method which has a reasonably high performance. We detail the results of each of these methods in Section 3.1. Finally, we describe how we classify videos using the histogram of noisemes feature vector.

### 2.1 Data Collection

We download sounds from Freesound using its Python API. The Python script uses the search engine on Freesound to search each noiseme in the Freesound database and download the first page (15 audio files) of results. This results in 615 audio files, each with the label of a single noiseme. We blindly trust the search engine and use these labels as the ground truth labels. We then convert the audio files to MP3 format to save disk space. This does not lower the quality of the dataset because the audio tracks of videos we classify are also compressed with lossy audio compression.

Later in the project, we collect more data by manually searching each noiseme on Freesound's web interface and downloading "packs," which are audio files bundled together by the uploader. Because the packs are created by the uploader, they are more likely to contain audio files which correspond well to the name of the pack. From our experience, only the first page of audio files we download through the search engine on the Freesound API are relevant to the search term. Additionally, the accuracy of the search engine for our search terms is not verified by a human being. Because we can trust the packs, we can download large quantities of packs with a large number of audio files in them. This method results in significantly more data, increasing our dataset size to 2116 audio files. Finally, we add 502 songs from the CAL500 music dataset [12] to the "music" noiseme category for a total of 2618 audio files.

All of the noisemes are either short sounds or a repetition of a single short sound. However, our dataset from Freesound contains some audio files which are very long ($>$ 5 minutes) and contain many sounds other than the noiseme. These extraneous sounds would lower the performance of our classifier. Because of this, we remove all audio files which are larger than 1 MB from the dataset. This helps ensured that the dataset does not contain any audio files which contain sounds other than the noiseme itself.

### 2.2 Feature Extraction

For each audio file, We use librosa [9] to extract the Mel frequency cepstrum coefficients (MFCCs), which have been shown to be applicable to speech and music discrimination [7]. The length of the FFT window is 2048 sample and the hop length is 512 samples. The sampling rate of the audio files is 44.1 kHz. This results in an MFCC frame duration of approximately 46 ms with the first sample of each frame spaced approximately 12 ms apart. The 180-dimensional MFCCs are decorrelated by multiplying by a rank 20 discrete cosine transform matrix from librosa.

## 2.3 Summary Feature Vectors

We use each dimension of the MFCCs as a feature. In order to have the same number of features for audio files with different amounts of frames, we create an 160-dimensional summary feature vector for each audio file. The first 80 dimensions are the means, variances, minimum, and maximum of the values of each MFCC dimension over time. The rest are the means, variances, minimum, and maximum of the values of the first derivatives of each MFCC dimension over time. We then use the principal component analysis algorithm [10] from Scikit-learn [11] to reduce the dimensionality of the summary feature vectors.

## 2.4 Codebook Generation

As an alternative to using the summary feature vectors decorrelated with a discrete cosine transform, we use vector quantization algorithms to generate a codebook.

### 2.4.1 K-means Clustering

The first algorithm we use is k-means [8] from Scikit-learn to transform each 20-dimensional MFCC frame into a single centroid. To train the k-means algorithm, we provide all of the MFCC frames from the entire training set as training examples, rather than distinguishing between the audio file each frame belonged to. The collection of centroids form a codebook and k-means transforms each frame into the closest centroid in the 20-dimensional space, which becomes the codeword for the frame. We then transform the list of codewords associated with each audio file into a histogram, using a technique similar to the bag-of-systems model [5]. Once we compute all of the histograms, we use a tf-idf transformation to appropriately weigh the value of each codeword. We use these histograms as the feature vectors for noiseme classification.

### 2.4.2 K-means Top 10 Centroid Assignment

We use a variation of the k-means clustering method by assigning the top 10 centroids as the codewords belonging to a single frame, rather than the top centroid.

### 2.4.3 Gaussian Mixture Models

The second vector quantization algorithm we use for codebook generation is Gaussian mixture models (GMM) from OpenCV [1]. In this method, we again train the algorithm without distinguishing between the audio file each MFCC frame belongs to. To transform each frame into a codeword, we select the GMM component with the highest posterior probability as the codeword. We use the same technique as in the k-means method to transform the list of codewords associated with each audio file into a histogram.

### 2.4.4 Individual Gaussian Mixture Models

We also use a variation of the GMM codebook generation method by using one GMM algorithm for each noiseme. In this method, we train each GMM with the MFCC frames from the audio files belonging to the corresponding noiseme. We generate the feature vector for a single frame by running each GMM on the frame and aggregating the resulting probability vectors. This results in a 410-dimensional feature vector. Finally, we normalize the feature vector with the $\ell 1$-norm and take the element-wise square root to account for non-linearity. We choose this as our codebook generation method as it results in the highest performance.

## 2.5 Noisemes Classifier

We use the individual GMMs codebook generation method to generate the feature vector for each audio frame. To classify the feature vector of an audio frame, we use the support vector machines algorithm [4] from LibSVM [3] through a Scikit-learn wrapper. We train the SVM with a linear kernel to classify an audio frame as belonging to one of 41 noisemes.

## 2.6 Histogram of Noisemes Representation

To generate the histogram of noisemes representation for an audio file, we classify each frame of the audio file using the noisemes classifier from Section 2.5 and create a histogram of the number of occurrences of

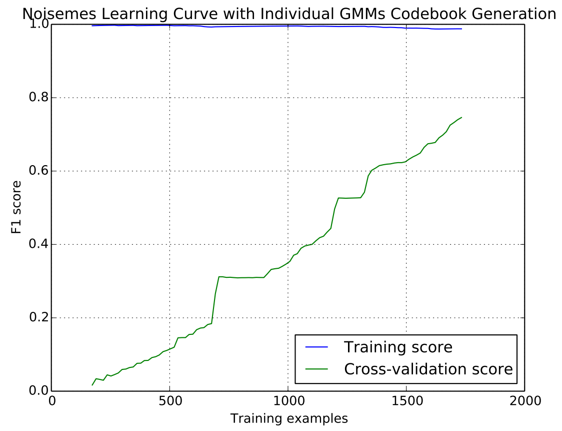

Figure 1: Learning curve for noisemes classification using a linear SVM with the individual GMMs feature vector

each noiseme in the audio file. We normalize the histograms with the $\ell 1$-norm and take the element-wise square root to account for non-linearity. We then use this histogram as the feature vector for video classification in Section 2.7.

### 2.7 Video Classification

To classify videos, we use the histogram of noisemes feature vector to train a multilabel support vector machine to tag videos.

## 3 Results

In this section, we discuss quantitative results of the noisemes classifier and the video classifier. The noisemes classification performance is very high but the video classification on the histogram of noisemes representation is poor.

### 3.1 Noisemes Classification

On average, the F1 score of the noisemes classifier is 0.76. The performance for each noiseme is listed in Table 1. There are no commonly confused classes. In the learning curve in Figure 1, we see that the cross-validation score does not reach a plateau. Finally, we detail the classification performance for each method of representing an audio file in Table 2.

| Noiseme | Precision | Recall | F1 score |
|---|---|---|---|
| animal | 0.60 | 0.68 | 0.63 |
| applause | 0.80 | 0.80 | 0.80 |
| bang | 0.47 | 0.64 | 0.54 |
| beep | 0.53 | 0.57 | 0.55 |
| cheer | 0.38 | 0.43 | 0.40 |
| child | 0.90 | 0.90 | 0.90 |
| clap | 0.80 | 0.84 | 0.82 |
| clatter | 0.57 | 0.67 | 0.62 |
| click | 0.61 | 0.69 | 0.65 |
| crowd | 0.53 | 0.69 | 0.60 |
| cry | 0.61 | 0.79 | 0.69 |
| engine_heavy | 0.58 | 0.92 | 0.71 |
| engine_light | 0.80 | 0.86 | 0.83 |
| engine_quiet | 0.58 | 0.58 | 0.58 |
| hammer | 0.77 | 0.67 | 0.71 |
| human_noise | 0.83 | 0.83 | 0.83 |
| knock | 1.00 | 1.00 | 1.00 |
| laugh | 0.59 | 0.67 | 0.62 |
| micro_blow | 0.50 | 0.43 | 0.47 |
| mumble | 0.73 | 0.69 | 0.71 |
| music | 0.97 | 0.99 | 0.98 |
| music_sing | 1.00 | 1.00 | 1.00 |
| other_creak | 0.81 | 0.68 | 0.74 |
| phone | 0.67 | 0.55 | 0.60 |
| power_tool | 1.00 | 1.00 | 1.00 |
| radio | 0.73 | 0.85 | 0.79 |
| rustle | 0.68 | 0.96 | 0.79 |
| scratch | 1.00 | 0.50 | 0.67 |
| scream | 0.91 | 0.67 | 0.77 |
| singing | 0.83 | 0.62 | 0.71 |
| sirene | 0.71 | 0.77 | 0.74 |
| speech | 0.96 | 0.95 | 0.96 |
| speech_ne | 0.79 | 0.77 | 0.78 |
| squeak | 0.80 | 0.80 | 0.80 |
| thud | 0.67 | 0.36 | 0.47 |
| tone | 0.80 | 0.57 | 0.67 |
| washboard | 1.00 | 1.00 | 1.00 |
| water | 0.52 | 0.35 | 0.42 |
| whistle | 1.00 | 0.93 | 0.96 |
| white_noise | 0.83 | 0.73 | 0.78 |
| wind | 1.00 | 0.91 | 0.95 |
| Average | 0.77 | 0.77 | 0.76 |

Table 1: Noisemes classification performance for the highest performing pipeline

| Method | F1 score |
|---|---|
| Summary feature vectors | |
| K-means clustering | 0.36 |
| K-means top 10 | 0.44 |
| Gaussian mixture models | 0.73 |
| Individual GMMs | 0.76 |

Table 2: Average F1 score for each method of audio file representation

| Video Tag | Precision | Recall | F1 score |
|---|---|---|---|
| music background | 0.35 | 0.88 | 0.50 |
| gardening | 0.12 | 0.75 | 0.20 |
| gaming news | 0.12 | 0.94 | 0.21 |
| food | 0.20 | 0.80 | 0.32 |
| racing | 0.32 | 0.77 | 0.45 |
| tennis | 0.27 | 0.78 | 0.40 |
| newscast | 0.34 | 0.91 | 0.49 |
| hot | 0.09 | 0.75 | 0.16 |
| beach | 0.42 | 0.76 | 0.54 |
| exercise | 0.28 | 0.59 | 0.38 |
| star interview | 0.12 | 0.87 | 0.21 |
| Average | 0.29 | 0.79 | 0.41 |

Table 3: Video classification performance

### 3.2 Video Classification

On average, the F1 score of the video classifier is 0.41. The performance for each tag is shown in Table 3.

## 4 Conclusion

We conclude that vector quantization algorithms produce a richer feature vector for audio frames than using the low-level MFCC features. We also find that a histogram of noisemes representation produces adequate, but not ideal, video classification performance.

### 4.1 Highest Performing Pipeline

In this section, we describe the overall highest performing pipeline for video classification. For a given audio file, we extract the MFCCs as described in Section 2.2. We compute the feature vector for the noisemes classifier using the individual GMMs method described in Section 2.4.4. We train an SVM to classify noisemes as described in Section 2.5. We then compute the feature vector using the histogram of noisemes representation as described in Section 2.6. Finally, we train an SVM to classify videos using the histogram of noisemes feature vector as described in Section 2.7.

## References

[1] Gary Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] Susanne Burger, Qin Jin, Peter F Schulam, and Florian Metze. Noisemes: Manual annotation of environmental noise in audio streams. 2012.

[3] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[5] Katherine Ellis, Emanuele Coviello, Antoni B. Chan, and Gert Lanckriet. A bag of systems representation for music auto-tagging. *IEEE Transactions on Audio, Speech and Language Processing*, 21(12):2554–2569, December 2013.

[6] Frederic Font, Gerard Roma, and Xavier Serra. Freesound technical demo. In *ACM International Conference on Multimedia (MM'13)*, pages 411–412, Barcelona, Spain, October 2013. ACM, ACM.

[7] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *International Symposium/Conference on Music Information Retrieval*, 2000.

[8] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. California, USA, 1967.

[9] Brian McFee. Librosa. `http://github.com/bmcfee/librosa/`, 2014. Accessed: 2014-07-07.

[10] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Douglas Turnbull, Luke Barrington, David Torres, and Gert Lanckriet. Towards musical query-by-semantic-description using the cal500 data set. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 439–446, New York, NY, USA, 2007. ACM.