Audio search - song recognition

Laboratory exercises

Introduction

We are going to experiment with audio search / song recognition methods on given audio samples. Various approaches will be compared to each other regarding speed, effectiveness, acoustic conditions and statistical indicators.

The basic task is the detection of a song in a reference database. We have a reference database of songs (containing only the fractions of the songs). Also, a test song is given that may have been altered (cut, noise or distortion added). Using various algorithms, the objective is to decide if the given short test recording can be recognized in the reference database, and if yes, with what timing (where does it start exactly).

The methods to be investigated:

- Trivial pattern matching in ***time domain*** (simplest way - without processing)
- Pattern matching in ***frequency domain*** based on spectrograms (sliding window + magnitude FFT)
- Pattern matching using MFCC (Mel-Frequency Cepstral Coefficients) vectors (a perceptually motivated feature extraction method)
- Pattern matching based on audio fingerprints (most commonly used in audio search systems)

The reference recordings in the database are always clean, original versions of the songs. We are going to modify and distort some of them and use as test samples.

Your task is to perform the measurements tasks and write a report. The documentation should contain the shell commands applied, modifications on the .m files, and the textual and graphical results of the programs. Use the pre-installed Libre Office to write the report. Press PrintScreen to create screenshots, they are saved in the home directory.

Directories and files in the main directory:

fingerprint: Matlab/Octave code for the audio fingerprint method

rastamat: Matlab/Octave code for MFCC feature extraction

octave_scripts: Matlab/Octave scripts used in the measurement task scripts

f1.m, f2*.m, f3.m f4.m: The scripts containing the measurement tasks. You must complete, run, modify (and re-run) them in order to accomplish the measurement tasks.

sox_scripts: these scripts can be used in the terminal, they are for the SOX sound processing program.

tracks: A collection of songs that can be used in the measurements. 10 song portions of different styles, approximately 16 s lengths, 16 kHz sampling frequency, 16 bit PCM coding, wav format

microphone: A test sample that is a part of one of the given reference recordings and is distorted in the following way: a part of the song was played by a phone's loudspeaker and recorded by a (built in) microphone of a laptop.

noises: Contains a few recorded noises from http://www.steeneken.nl/7-noise-data-base/ . They are originating from military machines, vehicles, etc.; the number 014 and 019 contain speech as well. Add noise to the clean test sample from this directory when it is needed.

csv: Contains a CSV table, the result of a bigger landmark cross test.

Exercise 1

The **f1.m** Octave script contains several pattern matching approaches: one operating in the time domain, another in the frequency domain, the third one using MFCC vectors (Mel-cepstral domain) and the last one is based on audio fingerprinting. The script is using one reference song and one test sample. Compare the 4 methods regarding their speed. Use **tracks/lvnp.wav** as a reference audio.

a) Create the test sample by cutting the section from 5th to 10th seconds of the reference audio file and use mp3 coding on it. (Use .sh scripts from sox_scripts directory)

b) Write the name of the test sample and the reference audio file in the beginning of f1.m script.

c) Study the f1.m script, and use time measuring commands to calculate the length of preprocessing the reference. (FFT, MFCC feature extracting, fingerprinting). Give back the results as RTF (Real Time Factor) values.

d) Measure the length of the whole process (preprocessing and pattern matching with the prepared reference) for the 4 methods separately like in c). Give back the results as RTF values.

Warning: During the pattern matching in time domain, the reference audio and test audio is realigned sample by sample using 0.0625ms steps (1/Fs) since there is no framing. This is really slow. Therefore, a shorter window sliding with shorter reference is prepared for this method. Use this information when calculating the RTF.

e) What do you think, in which practical application is it important to have a fast preprocessing and what applications are more sensitive to the speed of pattern matching?

Exercise 2

a) Create the following 8 versions of the 5 seconds long trimmed part of the test song used in Exercise 1.

1. clean, without any distortion
2. the version in the microphone directory (cut appropriately)
3. mix the original with **noises/SIGNAL019-20kHz-2min_16k.wav**
4. make the original faster by 10%, without changing the pitch
5. mix the original with another track
6. use a high-pass filter with cutoff frequenncy 4kHz
7. use GSM coding
8. use MP3 coding

Listen to the modified samples to verify them.

b) Write the names of the modified wav files to the proper place in f2_wav_names.m. Run the f2_wav.m, f2_AS.m, f2_MFCC.m, f2_fp.m scripts.

These scripts execute the preprocessing and the pattern matching with all the 4 methods on all the 8 sound samples. The reference is the same track as in Exercise 1. The results are shown as follows:

The waveform (**f2_wav.m**), the amplitude spectrum (**f2_AS.m**) and the MFCC (**f2_MFCC.m**) methods are based on Euclidean distance calculation with sliding window, and they create similar figures. All 3 of them creates 2-2 figures. On the first one the y axis is automatically scaled to let the values fill the figure vertically. On the second one, there are the same data with unified y axis. (Just like in Exercise 1 we use a smaller reference for waveform pattern matching, to make the running time shorter.). The 8 subplots are for the 8 test samples.

The fingerprint pattern matching script (**f2_fp.m**) creates a figure for each test file. There are 2 subplots for each figure. The first one is the spectrogram of the test sample, the second one is the matching part of the spectrogram of the reference. The matching landmarks are green; their number, the time (offset from the beginning of the reference) of the match and the name of matching file are also displayed on the figure.

Investigate and compare the Euclidean distance calculation figures and write down your observations by answering the following questions:

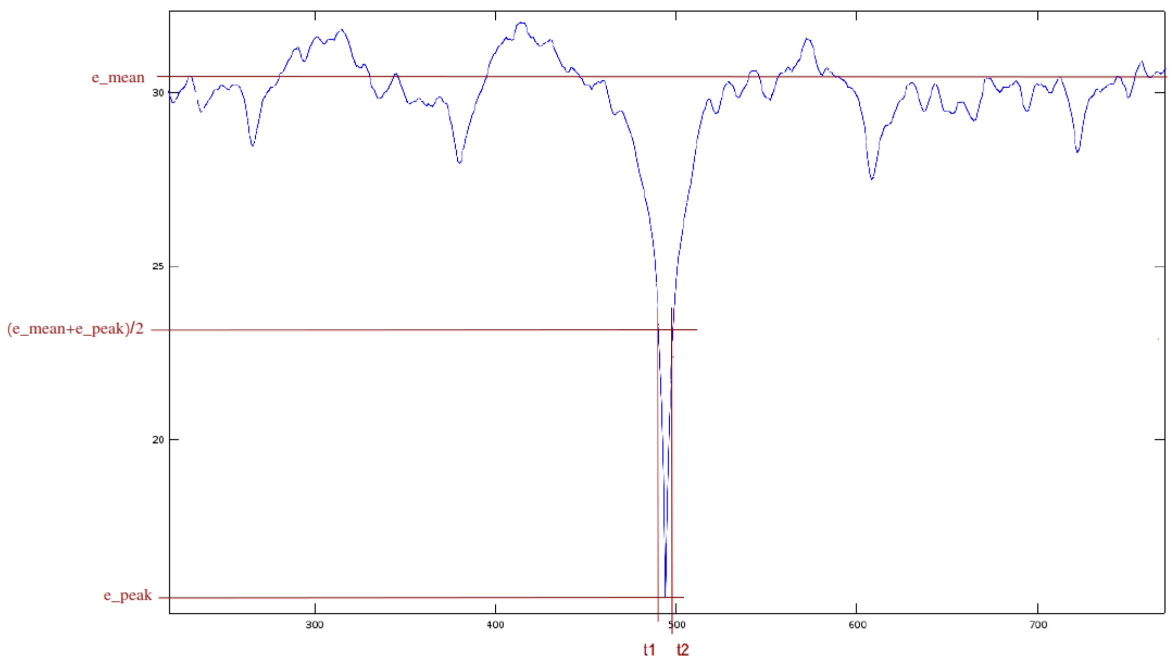
c) What justifies in the figures the difficulty of the identification of the noisy samples?

d) Which distortion is easier and which distortion is more difficult to deal with for each method (based on the figures)?

e) Estimate the minimum needed time resolution of the waveform, the spectrum and the MFCC based pattern matching methods considering the following:

In these tests, by default, the finest time alignment resolution (aligning by every sample or frame step) is used, what can be time consuming. To speed up the process, it would be better to use coarser time resolution, i.e., to apply bigger time shifts between possible alignments.

Suppose we want to set the threshold of the peak detection algorithm to be able to detect peaks half as deep as we see in our tests. To calculate the minimal temporal resolution, let's look at the (negative) peak on the figure below. Read the temporal distance between the two points next to the peak, where the curve crosses the half-deep threshold.



Study the figures of two test samples: the original and the GSM encoded samples. Read the t2-t1 value (in ms) by enlarging the figures of waveform, amplitude spectrum and MFCC methods.

Note that the interval of one step on the x axis is different for each method. For the waveform it is 1 sample (1/16000 s), for the amplitude spectrum it is 256 samples, for the MFCC it is 10 ms.

Study the figures of fingerprint pattern matching and answer the questions:

f) What differences can you spot between the distorted and the original spectrograms? Write down them based on 2 significantly modified spectrogram.

g) Which distortions are easier and which distortions are more difficult to deal with for the fingerprinting method?

Exercise 3

Look for the limits of usability of the fingerprinting and the MFCC song recognition methods. Create test samples with distortion and noise from the same song as you used in exercise 1 and 2, which can still be recognized by ear, but the audio search algorithms fail.

Check the solution by using the **f3.m** script. Write the name of the distorted wav file into f3.m, then run the script. It will look for a match and writes whether it succeeded to find one.

a) Create a distorted version of the 5 s long part of lvnp.wav, so that the f3.m script cannot identify it with the landmark method.

b) Create another (a different from the above) distorted version of the 5 s long part of lvnp.wav, so the f3.m script cannot identify it with the MFCC method.

Use the **sox_scripts**, octave commands, functions or maybe another audio processing program.

Exercise 4

Analyze the results of a previously done test series, which is in the **csv/cross_test_2000.csv** file. This file contains a 2000x2000 matrix, which is the result of cross testing 2000 parts of songs with the fingerprinting method. The j^{th} column of i^{th} row means the number of matching landmarks when we compared the i^{th} test song to the j^{th} reference song. (The running of these tests took several hours, so running it is not the part of the exercise.)

Every song compared with itself have obviously a large amount of matches, and the comparison with any other song of the database gives significantly fewer matches (provided that the database is composed properly). So, in the diagonal of the matrix there are big numbers, and everywhere else there are small numbers or zeros. Two songs, with identifiers *i* and *j* are considered as matched if the number of matching landmarks in the j^{th} column of the i^{th} row is greater than threshold *T*.

Evaluate the results given in the matrix in question. Use the **f4.m** script! It compares each values in the matrix with the threshold in variable *T*, and calculates the true positive, false positive, true negative, false negative, accuracy, recall, precision, F-measure metrics.

a) Run the script, then modify the values of T so that it recognizes every track. What value of which metric indicates this case?

b) Modify T so that the number of incorrectly recognized tracks be less than 20. How did the number of false negatives change? What could be the reason that caused the number of false negatives change this way?

c) List some of the track ID's that are affected by the database error discovered in task b).