Music Informatics - Assignment 1

Brendan O'Connor

March 2021

1 Introduction

The task for this assignment was to design a program that matches a query recording to the most similar recordings within a given database. Müller (2015) describes a method used by Wang (2003) that is reportedly the basis behind the framework for the popular music-identification app, Shazam. The code content written to achieve such a task does not stray far from what is described in Müller (2015), so I will be sure to highlight any original contributions used to improve the processing speed and performance of the application. Section 2 describes each step that was used to achieve accurate document retrievel from the given database. Section 3 evaluates results produced by this implementation, and discusses the variations in parameters and analysis used to gain knowledge on how the implementation can be improved.

2 Implementation

The file `Assignment2_FinalDraft.ipybn` is provided to accompany the descriptions given in this written work. You will see at the top of this notebook there are a set of variables. The functions used to generate a working system will be described in the order they appear in the notebook. The file `Assignment2_oneLiner.ipynb` is also provided for a short and visually simplistic implementation of the code, where cell 4 demonstrates the specified functions in the assignment briefing as well as performing some evaluations (note that this particular notebook is for demonstration of the working implementation, and the results therefore do not echo the results reported in this write-up - use `Assignment2_FinalDraft.ipybn` to verify this).

As summerised in the function `audio_to_low_rez_cords`, recordings from the directory `database_recordings/` are converted into spectrograms. Peaks within a specified context of these spectrograms are identified using Skimage's convenient `peak_local_max` function, and a list of their coordinates describes a *constellation map* (CM) that represents the 'audio fingerprint' of the recording. These CMs are unique to the audio recording, and are fairly robust against noise. This is useful as this implies a query segment containing some noisier version of the recorded audio should still retain a lot of the peak information in

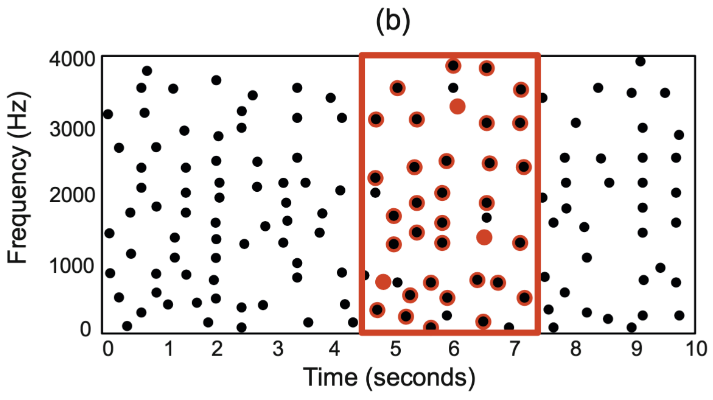


Figure 1: Taken from Chapter 7.1.2 of Müller (2015)

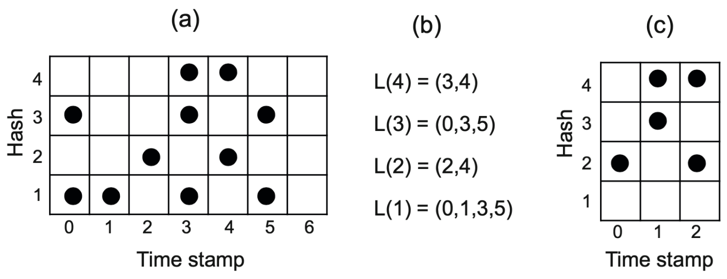
the same places. Equation 1 describes how the CM of a query recording $C(Q)$ is shifted m steps in time. By applying this shift, we can evaluate how similar $C(Q)$ is to a segment of a database audio document's CM $C(D)$ (illustrated in Figure 1 and formulated in Equation 2).

$$m + C(Q) := \{(m+n,k)|(n,k) \in C(Q)\} \tag{1}$$

$$\Delta_C(m) := |(m + C(Q)) \cap C(D)| \tag{2}$$

Due to the expectancy of this task to be applied to an enormous amount of documents in real-world contexts, Wang (2003) suggests using the indexing technique *fast combinatorial hashing* for database documents, which converts the peak coordinate list to fixed-length binary vectors $F(D)$ representing peak activity within different frequency bins for every timestamp. *Inverted lists* are then constructed for each hash, so that hash bins contain a list of timestamps where peaks are present. Instead of needing to scan over entire constellation maps for each document with the database, we can use these inverted lists $L(h)$ as templates to compare against a query's CM structure.

The CMs for all database documents are generated and stored in their hashed form. CMs are also generated for the query recordings but they do not go through the same hashing and inverted list process as those of the database documents mentioned above. Instead, each coordinate $(n,h) \in C(Q)$ is compared with the database document's $L(h)$, offset by the value of n (Equation 3) producing a 1 if there is a peak match, and 0 if not. These are then summerised for each timestep, and the maximum summation is recorded as the highest similarity offset by m (illustrated in Figure 2 and formulated in Equation 4). The highest similarities are recorded for every document in the database, after which the top 3 ranks are determined as the best matching document.



(d)

Query (*n*,*h*)	***L*(*h*) - *n***	**Indicator functions**									
		…	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**…**
(0,2)	(2,4)	0	0	0	0	1	0	1	0	0	0
(1,3)	(-1,2,4)	0	1	0	0	1	0	1	0	0	0
(1,4)	(2,3)	0	0	0	0	1	1	0	0	0	0
(2,2)	(0,2)	0	0	1	0	1	0	0	0	0	0
(2,4)	(1,2)	0	0	0	1	1	0	0	0	0	0
Matching function		**0**	**1**	**1**	**1**	**5**	**1**	**2**	**0**	**0**	**0**

Figure 2: Taken from Chapter 7.1.3 of Müller (2015)

$$L(h) - n := \{l - n | l \in L(h)\} \tag{3}$$

$$\Delta F(m) := \sum_{(n,h)\in F(Q)} 1_{L(h)-n}(m) \tag{4}$$

2.1 Improving Performance and Speed

One step taken to increase the processing speed involves reducing the CM dimensionality on the frequency axis, and removing the offsets for constellation maps so that their first peak coordinate occurs at timestep 0. This is achieved using the `reduce_cord_dims` function. Multiple parameters relating to the transform to spectrograms and peak-picking were altered to investigate their influence on performance. The results for these adaptations are present in Section 3. One step towards improving this process is comparing paired peaks to one another. This was not utilised in the submitted implementation. However its application would not only speed up the process of retrieving documents, but would also enforce specificity due to the fact that we are considering pairs of peaks as well as their matching distances.

3 Results and Evaluation

The highest similarity results between a given query and every document in the database are collected and sorted by their similarity ranking. This is truncated to produce the top 3 documents with highest similarity scores.

A relevance score is generated for each of the three predictions for every query. We then use the precision and recall equations defined in Equation 5 and 6.

$$P_Q(r) := \frac{1}{r} \sum_{k=1}^{r} X_Q(k) \tag{5}$$

$$R_Q(r) := \frac{1}{I_Q} \sum_{k=1}^{r} X_Q(k) \tag{6}$$

These calculations are executed in the notebook, but are not of much relevance in this instance, due to the number of known correct documents per query being considered as 1 for this assignment - although this is not actually the case (for example `pop.00056.wav` is a duplicate of `pop.00059.wav`). Instead, the accuracy is reported here as a percentage of document predictions that were correct. In Table 3 we can observe accuracy scores generated across the entire query set while under different conditions relating to transform and peak context parameters. It is clear that we can obtain better results with shorter timesteps between features, although the rate of change of computational speed

Hop/Window Size	Transform	Peak Context	Rank 1	Rank 2	Rank 3	Time (s)
256/1024	STFT	10	75	79	80	507
256/2048	STFT	10	76	81	81	884
256/512	STFT	10	52	56	56	139
512/1024	STFT	10	70	74	75	246
1024/2048	STFT	10	63	67	68	155
2048/4096	STFT	10	07	08	10	122
512/NA(n_bins=84)	CQT	10	14	16	17	190
512/1024	STFT	5	70	74	74	776
512/1024	STFT	20	64	70	71	140

Table 1: System performance reporting percentage accuracies for each rank and computational speed in seconds under multiple conditions

is less than that of performance. Interestingly, the performance and computational speed both decrease when the fft size of 1024 is halved, implying that there is a drastic loss of information when halving the frequency resolution from, but not a significant increase when doubling it. The rate of change of accuracy increases drastically as the hop size doubles, reflecting similar logic for the temporal resolution. Lowering the peak context causes the computational speed to increase while performance remains roughly the same. Increasing the peak context moderately decreases performance and increases computational speed. This makes sense as there would be less peak coordinates to work with, and so a straightforward trade-off is apparent regarding the peak context size. Predictably, using the CQT yields one of the poorest results, most likely due to its low dimensionality feature-space and human-like compromise to pin-pointing peaks in the spectrum.

To investigate how musical genre affects performance, accuracy scores were also generated for classical and pop queries separately. The results in Table 2 show the overall performance of the the system to be 75% when considering up to 3rd rank retrievals. These were generated with hop and fft window sizes of 512 and 1024 respectively, with a peak context of 10. It can also be seen that this is extremely high for popular music (98%), while classical music scores considerably lower (52%). It takes 246 seconds for the system to retrieve the most similar documents for all query recordings. This shows that the system performs differently for different styles of music. In this case, it is possible that stronger excitations in the higher frequency ranges of pop-produced music allows for more robust CMs than those of organically produced classical music. Therefore an additional improvement would be add mechanism to the system that determines how much importance is given to the higher frequencies if the genre type can be predicted beforehand.

Song Types	Rank 1	Rank 2	Rank 3
All	70	74	75
Pop	89	98	98
Classical	51	51	53

Table 2: Accuracy results for document retrieval across different song types

References

Müller, M. (2015). *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications.* Springer.

Wang, A. (2003). An industrial-strength audio search algorithm. In *Proceedings of the 4 th International Conference on Music Information Retrieval.*