Toward Automatic Music Audio Summary Generation from Signal Analysis

Nicolas Ayroles, Michel Blancard

March 29, 2015

Introduction

Music summarization refers to the process of discovering the structure of a song. In our case, it implies mainly separating dissimilar segments of music and linking similar segments. Automatic summarization is a problem of great importance for online music distribution platforms and music search engines.

In [1], Geoffroy Peeters, Amaury La Burthe and Xavier Rodet propose a two-fold approach to automatically discovering the structure of a popular music song. In a first step, an initial set of states are derived from a first pass of the music. The segmentation is improved in a second step by modeling the structure with a hidden Markov model. Both passes rely on features computed from the original audio signal and selected during a supervised learning phase.

This report describes the pipeline we implemented, based on the work of Geoffroy Peeters, Amaury La Burthe and Xavier Rodet.

Mel filter bank

The first step to obtain features is to apply the Mel-scale filter bank to the original signal. This consists in computing the short term (windowed) Fourier transform (with a window size of 0.025s and a step of 0.01s), binning the Fourier coefficients in 26 channels and computing the logarithm of the magnitude of these 26 sums. The 26 bins are scattered on the frequency space following the Mel scale, a logarithmic scale that aims to reflect the humain ear's response. That is why the Mel features are very popular in the Automated Speech Recognition field.

We used the python package 'python_speech_features' to compute the Mel features. This package performs a pre-emphasis (high-pass filter) before the Fourier tranform and the binning.

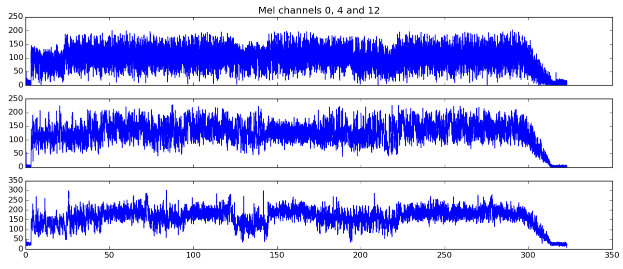

Figure 1: 3 Mel channels on the song 'Head over feet'

Dynamic features

In order to find states, the variations of the Mel features are more indicative than just their values at one date. This dynamic information can be exploited by computing the short term Fourier tranform of the Mel features, resulting in "dynamic" features.

The size of the window is crucial : a short window will produce more precise -but more varying- features than a long window would produce. Figure 3 shows the similarity matrices obtained with different window sizes. We found that a window size of 1024 samples, corresponding to 10 seconds of music) is a good compromize. With a step of 102 frames, the output features have a sampling rate of about 1Hz. This is much more convenient to process than the original 100Hz features, especially for computing the full similarity matrix.

Feature selection

With a window size of 1024 frames, 512 dynamic features are obtained from a single Mel channel. This gives a total of 13312 dynamic features. Unfortunately, only a tiny fraction of these features are informative in our context. The majority of non-informative features alter the quality of the similarity measure between frames, leading to poor results during the segmentation stage of the pipeline.

One way to eliminate this burden is to assess the quality of each feature using manually entered structures for a certain number of songs. First, we studied 21 popular music songs from several artists belonging to several genres. The structure of a song is noted by a sequence of pairs (begin, id) where 'begin' is the date of the begining of a segment and 'id' is a state id, a number identical for similar segments. For each state s, a ground truth vector $g_s(t)$ is equal to 1 if the song is in state s at time t and 0 otherwise. Then the mutual information can be estimated between g_s and each dynamic feature. Mutual information estimation is a non-trivial problem, especially for continuous variables when the samples are small. To overcome this difficulty, we binned the values of each feature in 10 bins uniformely distributed between the minimal value and the maximal value of that feature. This way, the distribution becomes discrete and a summation formula can be applied to compute the mutual information, at the price of a loss in precision. This estimation could be improved by using sophisticated bias-reduction techniques. The score of a features is the sum of the mutual information with every state, for every song in the training dataset.

Experiments on our labeled dataset showed that the results are best when 20 to 50 features are selected. We weight the features proportionnaly to their score, however this makes not much difference compared to the equal normalization of all the selected features.

Of course, we took care to always exclude the tested song from the training dataset used.

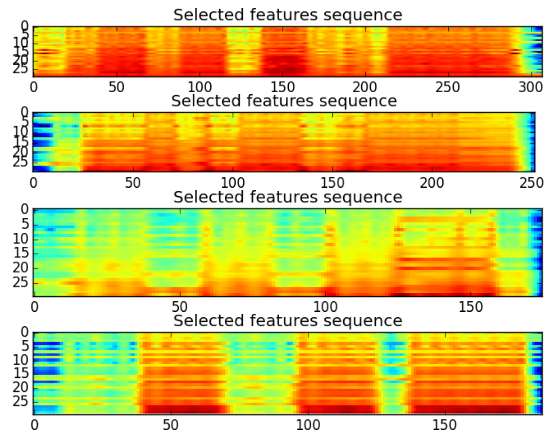

Figure 2: Selected features for the songs 'Head over feet', 'If I ever feel better', 'Shadow on the wall, 'SOS' (Abba)

Similarity matrix

The full similarity matrix is computed by measuring the similarity between two frames, according to their respective features. We use the well known cosinus similarity measure. Figure 3 illustrates the importance of the choice of the window size. Figure 4 shows the similarity matrix that we obtained for 4 songs.

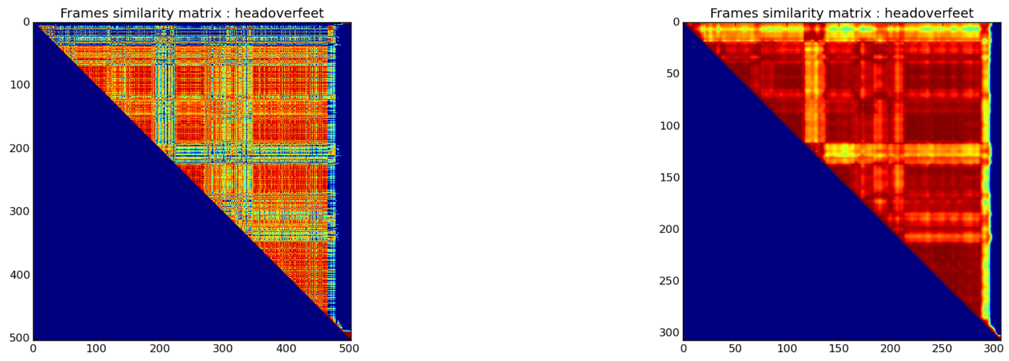

Figure 3: Small window (128 frames) vs large window (1024 frames) on he song 'Head over feet'

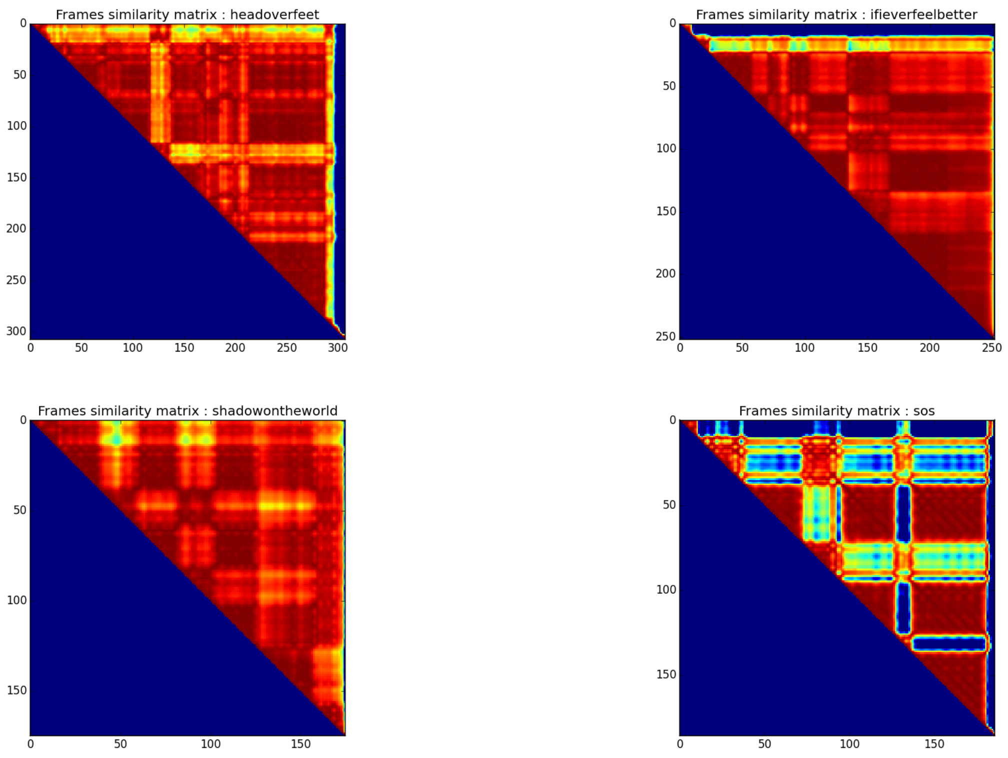

Figure 4: Similarity matrices for the songs 'Head over feet', 'If I ever feel better', 'Shadow on the wall, 'SOS' (Abba)

Segmentation

The upper (or lower) diagonal contains the frame to frame similarity measures. It is of much interest for our problem, as it is supposed to display values close to 1 except when the song goes from one state to another. A first segmentation can be obtained by selecting dates where the frame to frame similarity crosses a threshold (0.9995 to 0.9999 in our case). In 5, the guessed segmentation (in black) can be compared to the ground truth (in blue).

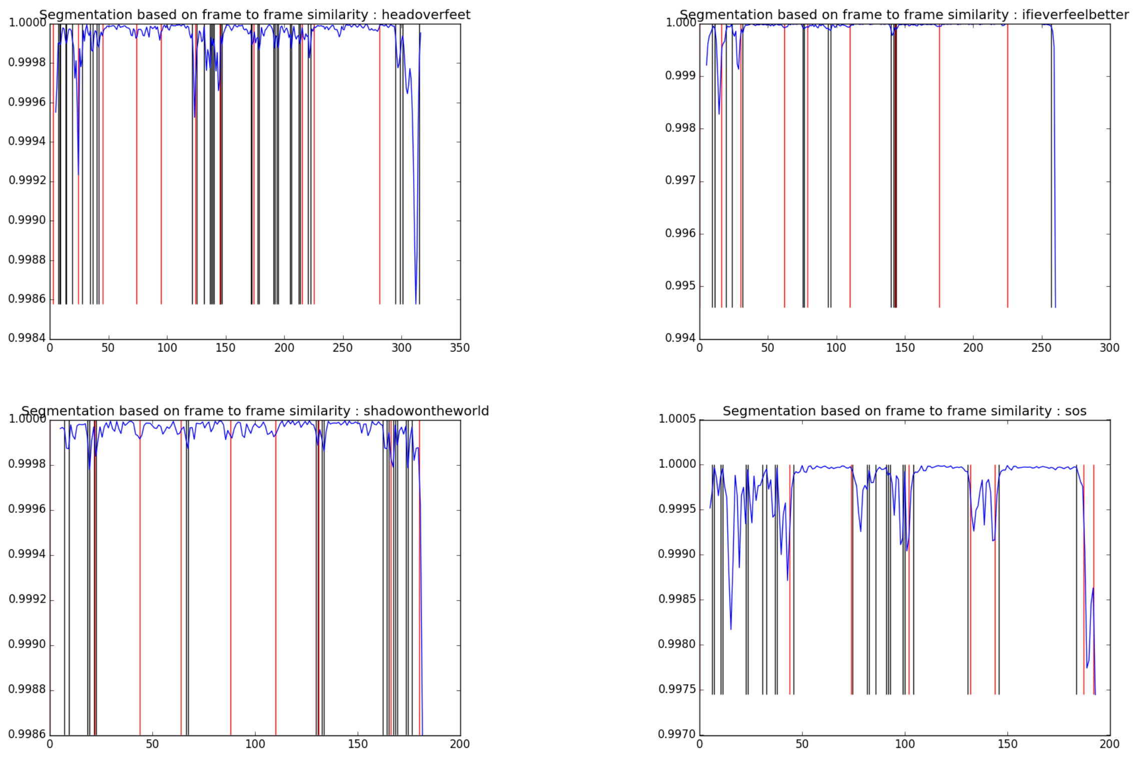

Figure 5: Upper diagonal for the songs 'Head over feet', 'If I ever feel better', 'Shadow on the wall, 'SOS' (Abba)

Potential states

For each of these segments, we define a *potential state* caracterized by the mean value of the feature vector over the segment. Figure 6 shows the *potential states* we obtained.

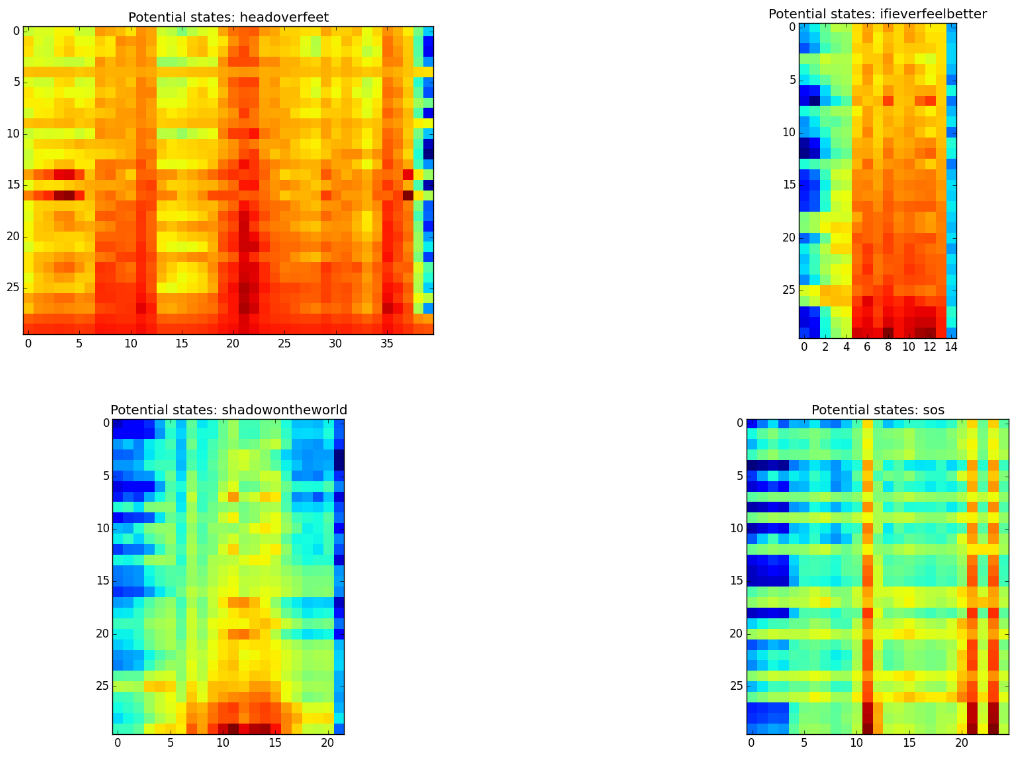

Figure 6: Potential states for the songs 'Head over feet', 'If I ever feel better', 'Shadow on the wall, 'SOS' (Abba)

Initial states

The next step is to build the similarity matrix of the *potential states* we have just computed. This allows us to compare all the *potential states* pairwise and group them if they have a similarity greater than the same high threshold used for the segments definition. The means of the *potential states* grouped together define what we call the *initial states* plotted on figure 7. We note K the number of *initial states*.

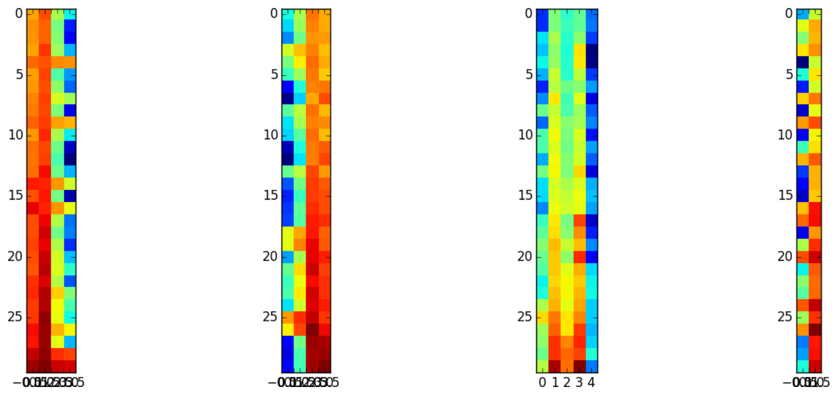

Figure 7: Initial states for the songs 'Head over feet', 'If I ever feel better', 'Shadow on the wall, 'SOS' (Abba)

K-means

These *initial states* are used for the initialisation of the K-means algorithm which is run on the frames. This means that after the K-means algorithm has been run, each frame is assigned a label $i \in \{1 \ldots K\}$ corresponding to the most similar *initial state*. This allows to define a first state sequence over the song as shown on figure 8.

However, this assignment does not take into account the temporal ordering of the frames. Indeed, each frame has been assigned to the most similar *initial states* independently from its temporal position and from the state assigned to its neighbouring frames. This does not reflect to nature of music as a song is not merely a sequence of independent frames but rather a structured sequence of interdependent states. We thus improve our approach further by exploiting a Hidden Markov Model (HMM).

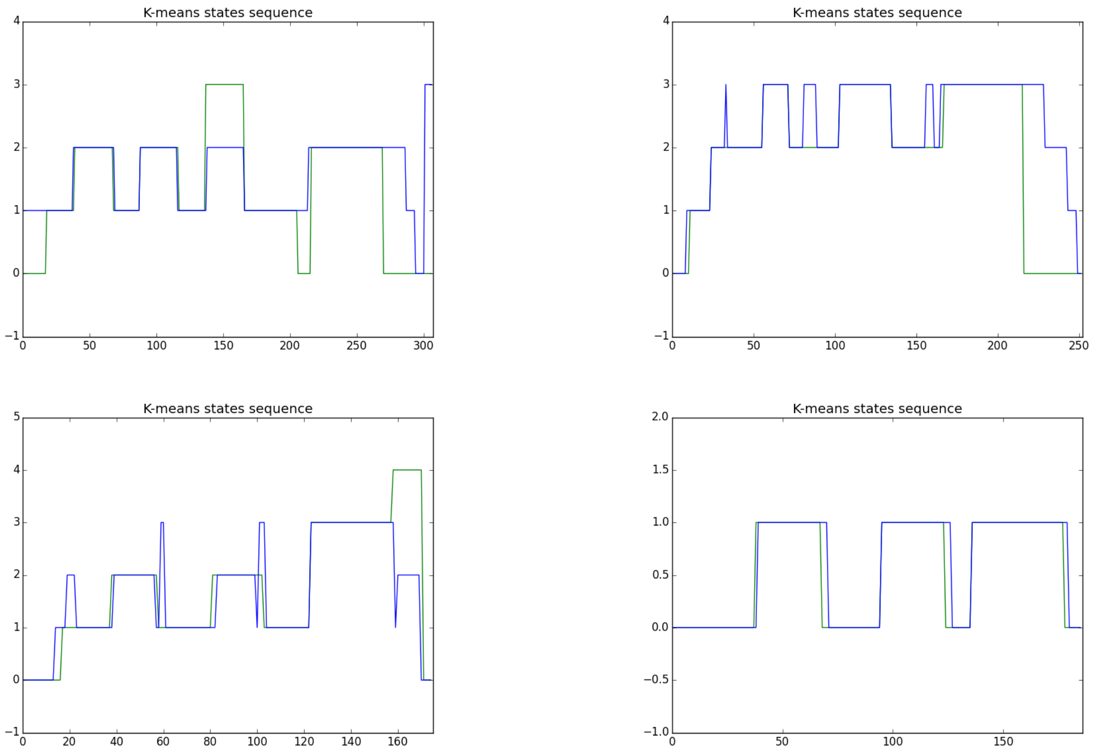

Figure 8: Sequence of states from K-means for the songs 'Head over feet', 'If I ever feel better', 'Shadow on the wall, 'SOS' (Abba)

Hidden Markov Model

In our case, the HMM defines a temporal sequence. Each instant t is caracterized by one of the K states defined previously. The model specifies the probability at each time t of being in the state i depending on the state j at instant $t-1$. It also gives the probabily of observing the feature vector $f(t)$ at instant t depending on the state i at instant t. Since we only observe $f(t)$ and not the actual state i, the variable describing the state is said to be hidden. In order to define the model, we thus need to explicit the transition probabilities from one state to another and the probability density function describing the relation between the hidden state and the observed feature vector. The model is trained on the observed feature vector of the song. We assume a gaussian probability density function. Once the model is trained, we can compute the most probable state sequence given the observed feature vectors. This was done with the Viterbi algorithm.

On the results plotted on figure 9, we can see that the HMM takes into account the temporal ordering of the frame and thus tend to smooth the state sequence. This allows to correct some errors that occured in the K-means.

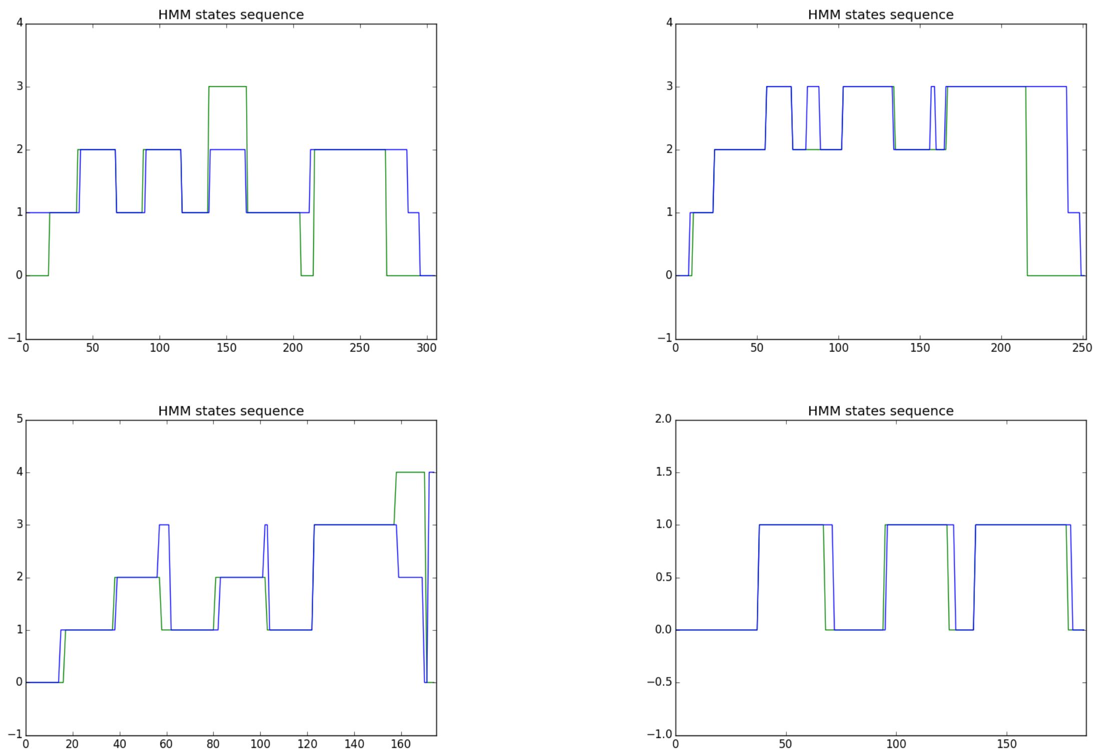

Figure 9: Sequence of states from HMM for the songs 'Head over feet', 'If I ever feel better', 'Shadow on the wall, 'SOS' (Abba)

Conclusion

We have seen the method proposed by Rodet and al. in [1]. This method consists in extracting features from the audio signal at very short time scale with the Mel features. In order to take into account the evolution of these features over time, the STFT of the Mel features is computed over 10 seconds. In order to study the similarity between the frames and find the most discriminative features, the mutual information between each feature and a set of labeled states is computed. We then seek to group the features by similarity in order to identify the number of states in the song. This allows us to run a K-means algorithm and assign a state to each feature vector. Finally, a HMM is run to better account for the temporal ordering of the feature vectors and to get a smoother sequence of states.

We achieved good results on some songs. We shall however precise that we were not able to get such good results on all songs. One of the reason might be the size on our training set, which contains 21 songs only. Setting the value of the similarity threshold is also challenging. In our experiment, its value was chosen manually and we used the same value for every song. Defining a rule to measure the correctness of the threshold and automatically adapt it to each song is a possible source of improvement. Finally, we noticed that the frame to frame similarity is not adapted to all musical genres. Indeed, for songs with a complex structure, many instruments and/or many variations, it is very difficult to build a good measure of similarity as the MFCC values vary a lot. Exploring distance measures between feature vectors other than the cosinus and consider other audio features should also be tried.

References

[1] Geoffroy Peeters, Amaury La Burthe, and Xavier Rodet. Toward automatic music audio summary generation from signal analysis. In *In Proc. International Conference on Music Information Retrieval*, pages 94–100, 2002.