POLITECNICO DI MILANO

Homework #2 - Template-Based Chord Recognition

Students: *Matteo Pettenò - Marco Viviani*

Course: *Computer Music - Representations and Models* – Professor: *Clara Borrelli*
Due date: *December 23th, 2021*

In generale: inglese da rivedere e check formattazione equazioni equazioni.

Introduzione

In this report we analyze step by step the code implemented in the attached *Jupyter Notebook* trying to answer the questions asked for this homework. Along with the reading of this report, please make sure to check out code comments in the notebook. The first cells of the notebook are used to initialize the context and the parameters of the algorithm.

Context initialization. In order to optimize the code for automation and readability, in this section we initialize a dictionary that contains all the details of the songs used in the experiment (*Corpus*). The chord labels are generated here too.

Parameters configuration. In this cell we declare and set to a default value all the template-based chord recognition algorithm's parameters. This approach allows us to centralize the management of the algorithm's behavior, making it easier to evaluate once implemented.

Question 1

Implement the template based chord recognition algorithm.

Answer. A chord recognition algorithm consists of two steps. In the first step, the given audio recording is cut into frames that are transformed into a feature vector. This is tipically done with chroma-based audio features, which contain the tonal information of the audio signal. In the second step, pattern matching techniques are used to map each feature vector to a set of predefined chord models. The best fit determines the chord label assigned to the given frame.

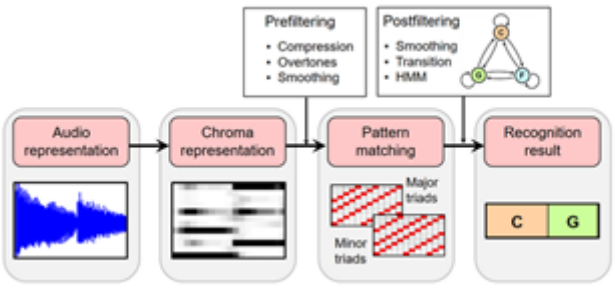


Figure 1: Template-Based Chord Recognition Pipeline

Features processing functions. Check punti 2, 3 e 4 che non siano troppo copiati.

In this section of the code all the functions that will be useful to implement features processing are defined. To improve the chord recognition results, in fact, additional enhancement techniques are applied either before the pattern matching step (*prefiltering*) or after the pattern matching step (*postfiltering*). In our implementation we only make use of prefiltering operations such as features compression, normalization, smoothing and downsampling.

1. `compress_feature_sequence` - This function takes in input the sequence of features and applies compression in a logarithmic fashion.

2. `normalize_feature_sequence` - This function normalizes the columns of a feature sequence. One normalization strategy is to choose a suitable norm and then to replace each n-feature vector by $x/p(x)$. The normalization procedure as described above replaces each chroma vector by its normalized version. Intuitively speaking, normalization introduces a kind of invariance to differences in dynamics or sound intensity. The normalization procedure is only possible if $p(x) \neq 0$. Also for very small values $p(x)$ which may occur in passages of silence before the actual start of the recording or during long pauses, normalization would lead to more or less random and therefore meaningless chroma value distributions. Therefore, if $p(x)$ calls below a certain threshold, the vector x may be replaced by some standard vector such as a uniform vector of norm one instead of dividing by $p(x)$. In the function exactly this steps are implemented. Normalization will be used also in the post-processing phase in order to normalize the chord similarity matrix.

3. `smooth_feature_sequence` - For certain music retrieval applications, chromagrams may be too detailed. In particular, it may be desirable to further increase the similarity between them. This can be achieved by applying smoothing procedures. The idea is to compute for each chroma dimension a kind of local average over time. More precisely, let $X = (x_1, x_2, ..., x_N)$ be a feature sequence with $x_n \in \mathbb{R}^k$ for $\in [1 : N]$, and let be w a rectangular window of length L. Then we compute for each $k \in [1 : N]$ a convolution between w and the sequence $[x_1(k), x_2(k), ..., x_N(k)]$. The result is a smoothed feature sequence of the same dimensions. For the window (also called kernel) results in a bandwise 1D convolution. Using the parameter *mode='same'* enforces the centered view. Applying temporal smoothing can be regarded as bandwise lowpass filtering, which attenuates fast temporal fluctuations in the feature representation.

4. `downsample_feature_sequence` - Often, to increase the efficiency of subsequent processing and analysis steps, one decimates the smoothed representation by keeping only every D-th feature, where $D \in \mathbb{N}$ is a suitable constant (typically much smaller than the window length L). This decimation, which is also referred to as downsampling, reduces the feature rate by a factor D.

Template-based chord recognition steps. Punto pattern matching: sistemare descrizione spiegando brevemente la similarity measure (np.matmul esegue semplicemente la moltiplicazione di due matrici). Sistemare equazione non in linea. Punto recognition result: VEDI MEGLIO. Here we define a function for each step performed by the template-based chord recognition algorithm.

1. `load_audio` - Loads the *WAV* file.

2. `chroma_representation` - Computes (and eventually compresses) STFT-based chroma features.

3. `generate_triads_templates` - Generate chord templates for major and minor triads. This function takes no input and returns the matrix containing the chord

templates as columns. Every column will be a binary vector representing the template for every minor or major chord.

4. `pre_processing` - The aim of this function is to *pre-process* the features used in the algorithm. If needed, it normalizes, smooths and downsamples the chroma features and triads template with the previous defined functions.

5. `pattern_matching` - It computes the *similarity measure* (*np.matmul*) between the triads template models and the predicted chroma features, according to this formula $(x, y) = \langle x, y \rangle / (||x|| * ||y||)$.

6. `post_processing` - Apply features *post-processing*. As mentioned above, this phase must not be confused with the concept of *post-filtering*: here we do not carry out any post-filtering operations, but we simply normalize the similarity matrix.

7. `recognition_result` - This function takes as input the matrix of triads templates and compares the current chord with the various templates. The tamplate of the chord that maximizes the chord similarity is the correct one.

Template-based chord recognition implementation. Here we answer the question with the actual template-based chord recognition algorithm implementation: the function `compute_template_based_chord_recognition` receives as input the path to a *WAV* file, calls the step functions defined above and returns a list where each element is the predicted chord label for the time frame n and other useful information.

Let It Be - Perform template-based chord recognition and plot results. Then we perform the algorithm for the file `Beatles_LetItBe.wav` and plot the results.

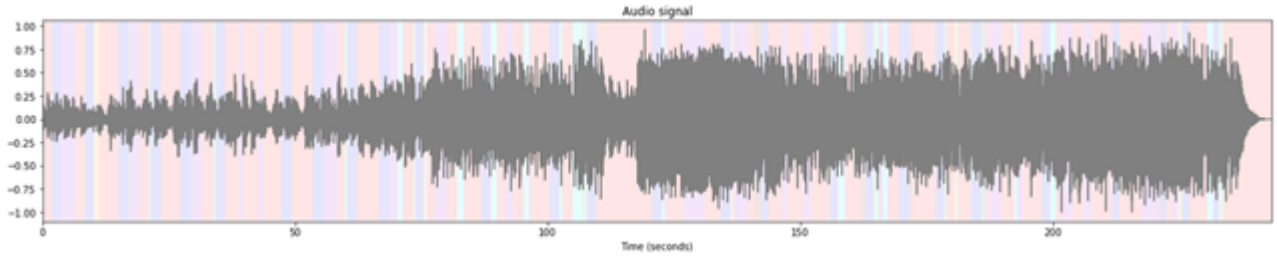


Figure 2: Audio Signal

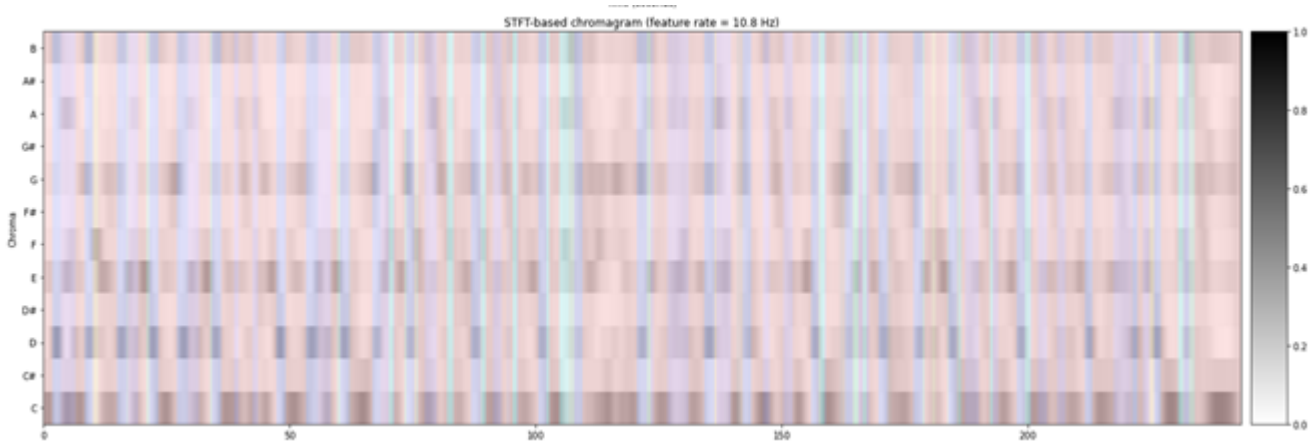


Figure 3: STFT-based Chromagram

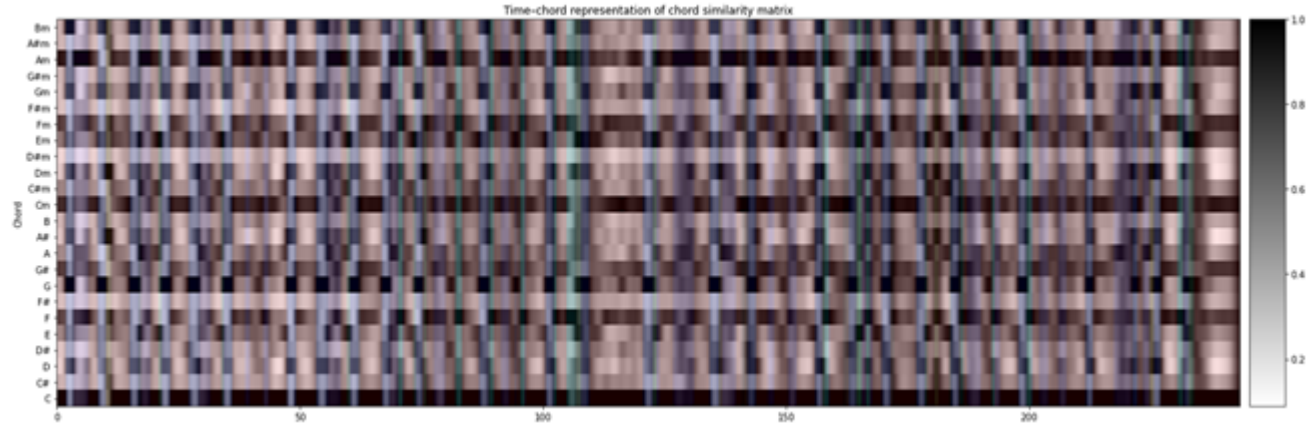


Figure 4: Chord Similarity Matrix

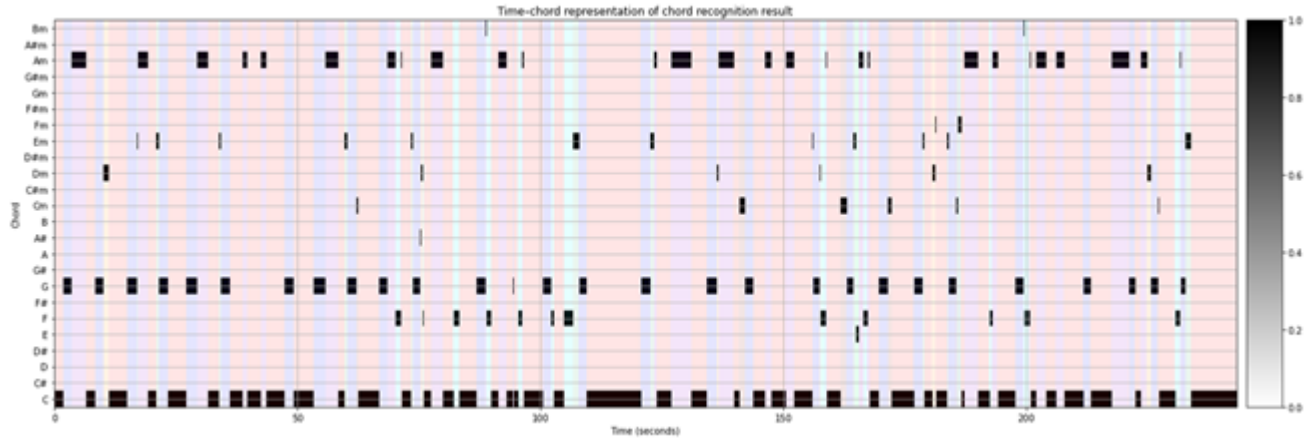


Figure 5: Chord Recognition Results

Question 2

Write a function to load and preprocess a reference annotation (or ground truth) file, saved in *CSV* format.

Answer. The function should take as input the path to a CSV file and produce as output a list of ground truth chord labels, after suitable pre processing.
The output must be a list where each element n is the ground truth chord label for the n-th time window.

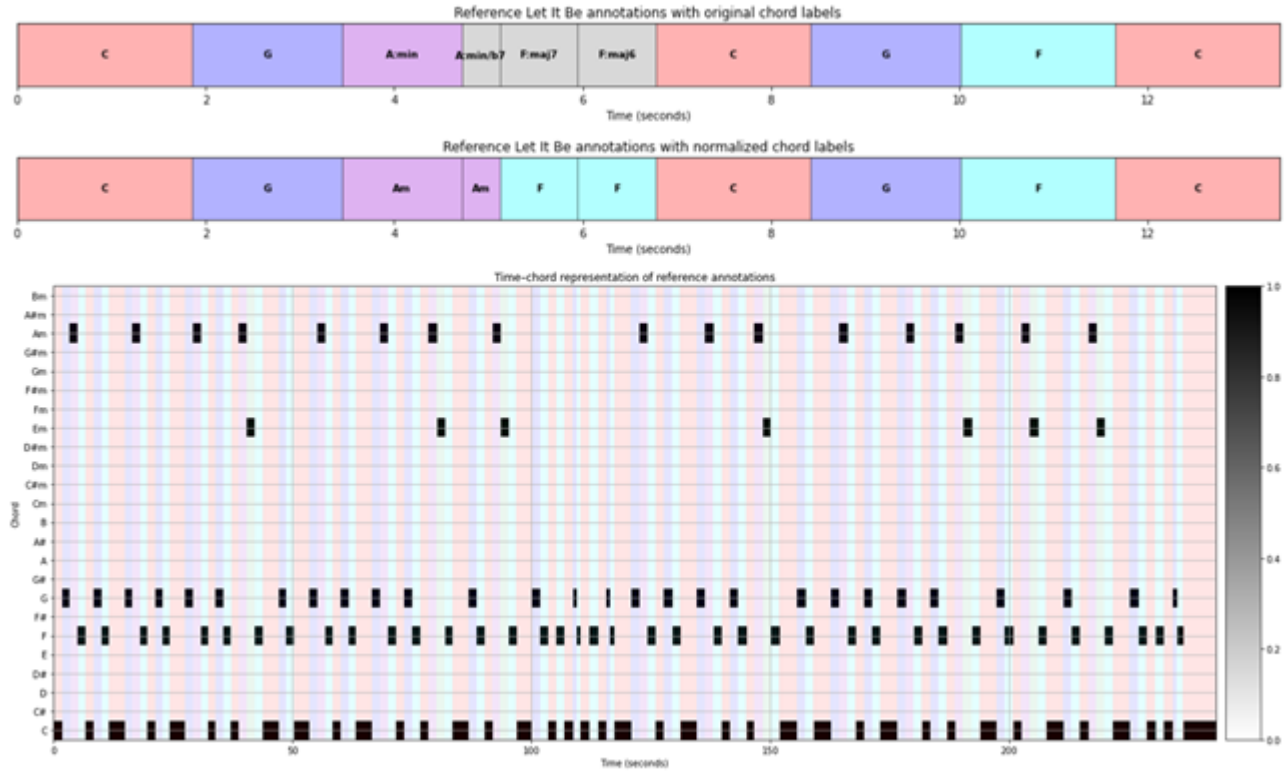
Ground truth processing functions. In this paragraph is explaied each step of the preprocessing phase, focusing in particular on the reduction strategy of the chord label set.

1. `read_csv` - Reads the path of the .csv file.
2. `convert_segment_annotations` - The function (required in the first point, question 2) has the important role to convert the segment-based annotation into a frame-based label sequence adapted to the feature rate used for the chroma sequence.
3. `get_binary_time_chord_matrix` - The function takes as input the sequence of lables and returns the matrix with the duration (in binary rapresentation) of the various chords. It converts the labels used in the annotation file to match the chord labels used for the chord recognition algorithm in terms of enharmonic equivalence. VEDI MEGLIO
4. `normalize_chord_labels` - Important to notice is the function that normalizes chord labels. It replaces for segment-based annotation in each chord label the string ':min' by 'm' and convert flat chords into sharp chords using enharmonic equivalence. We can also see that half diminished, diminished and minor chords are classified as minor chords with the letter 'm'. Maj, sus and slash chords are classified as major chords (' '). This is done by using Python's *regex* (regular expressions). In this way we implemented the reduction strategy of the chord label set.

Ground truth reading implementation. The role of this function is to load and preprocess a reference annotation (or ground truth) file, saved in CSV format. The function `read_ground_truth` should takes as input the path to a CSV file and produce as output a list of ground truth chord labels, after suitable pre processing. The output is a list where each n-th element is the ground truth chord label for the time window n.

The function converts segment-based chord annotation into various formats and returns a frame by frame reference chords label sequence, the binary time-chord matrix representation of the reference chords label sequence,the original reference annotations given in seconds, the normalized reference annotations given in seconds.

Perform ground truth reading. In this section the code performs ground truth reading, plotting the results of chord recognition.



Question 3

Propose a metric for evaluating the template based chord recognition algorithm.

Answer. In this section we define a function that takes as input the list of predicted chord labels, the list of ground truth chord labels and computes the proposed metric value: `compute_eval_measures`.

A metric is a scalar number that expresses how good is the algorithm in performing the task of chord recognition.

We now introduce a simple evaluation measure. Recall that, given a chroma sequence $X = (x_1, x_2, \ldots, x_N)$ and a set $\Lambda := [C, C\#, \ldots, B, Cm, Cm\#, \ldots, Bm]$

In the context of music structure analysis, we introduce evaluation measures that are used in general information retrieval. We now adapt these notions to our chord recognition scenario. First, we define the set of items to be $L = [1 : N] \times \Lambda$. In particular, the non-chord label N is left unconsidered. Then: $L_+^{(}Ref) := \{(n, \lambda_n^{(}Ref)) \in L : n \in [1 : N]\}$ are the positive (or relevant items) and $L_+^{(}Est) := \{(n, \lambda_n^{(}Est)) \in L : n \in [1 : N]\}$

are the items estimated as positive (or retrieved items). With these notions, an item (n,λ_n) is called a true positive (TP) in the case that the label is correct (i.e., λ_n=$\lambda_n^{(}Ref)$). Otherwise, (n,λ_n) is called a false positive (FP) and (n,$\lambda_n^{(}Ref)$) a false negative (FN). All other items in I are called true negative. With these notions, one can define the standard precision (P), recall (R), and F-measure (F):

$$P = (\#TP)/(\#TP + \#FP)$$
$$R = (\#TP)/(\#TP + \#FN)$$
$$F = (2PR)/(P + R)$$

In the way we have formulated our chord recognition problem so far, we have exactly one label per frame in the reference as well as in the estimation. From this follows that #FP=#FN and that the definition of accuracy coincides with precision, recall, and F-measure. This is why in the function we use as measure the precision P. If #TP=0, the precision is setted to the default value 0.

Metric evaluation implementation. Report `compute_eval_measures` Python code

`compute_eval_measures` - The function takes as input the binary time-chord matrix representation of the reference chords label sequence and the binary chord similarity matrix containing maximizing chord. It implements the metric (precision) defined previously by calculating the cardinality (*np.sum*) of TPs (true positives), FPs (false positives), FNs (false negatives). If the numerator (#TP) is bigger than value 0, the precision value is calculated as defined before.

This function implements exactly the precision measure as said in the previous paragraph.

Perform metric evaluation. Improve description

We compute the evaluation for the song "Let it be" by Beatles. Using the STFT-based chromagram in combination with the template-based chord recognizer, we obtained a precision of F=60,27%. In fact, as the visualization shows, there are many sudden jumps between chord labels. We could optimize this result by choosing the optimal parameters setted at the beginning, but this won't be the best result.

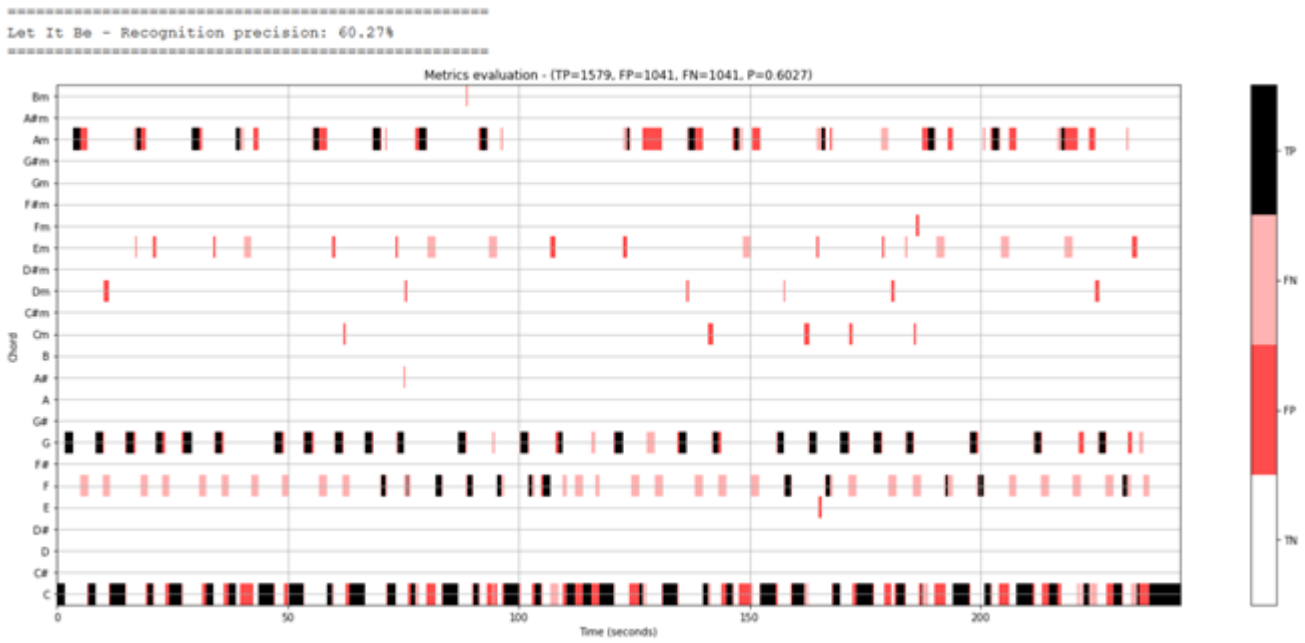


Figure 6: Let It Be - Metrics Evaluation

Can you imagine a musically informed strategy that weights diferently mismatch errors of the chord recognition algorithm?

Answer. Manca risposta a questa domanda!Introduces a kind of context-aware post-filtering.

Question 4

Compute the proposed metric for the remaining 3 songs.

Answer. Improve description

We compute also the proposed metric for the remaining 3 songs. Averaging these numbers on the song level yields a mean precision P=60.48%. The value P=60.27% that we found in "Let it be" is very similar to the mean; we can say that this is still due to sudden jumps between chord labels. With HMM we can obtain P=78,5%, the value one also obtains for the entire Beatles collections (180 songs). More recent data-driven chord recognition approaches achieve recognition rates between 80 and 90 percent for the Beatles collection. In general, such numbers need to be taken with caution, since they only give limited insights into the complexity of the dataset.

Figure 7: Here Comes The Sun

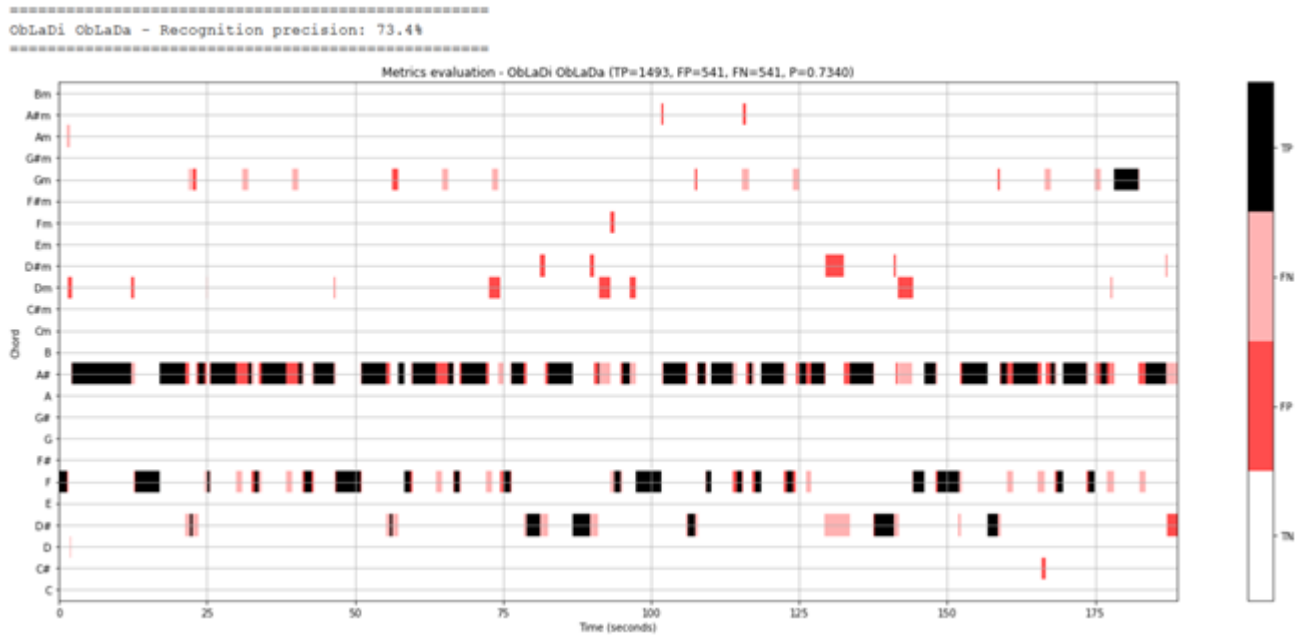


Figure 8: ObLaDì ObLada

Figure 9: Penny Lane

Question 5

Analyse how algorithm parameters affect the performance of the templated based chord recognition algorithm.

Answer. Aggiungere descrizione su come è stata fatta l'implementazione del testing (iterate over songs dictionary and overwrite globals)

Smooth filter length. Applying temporal smoothing using a rectangular or a Hann window can be regarded as bandwise lowpass filtering, which attenuates fast temporal fluctuations in the feature representation. This allows the chord estimation to be much more precise and not be affected by high frequencies.
We consider the following vector of values [0,30,60]: we can see an increase in accuracy up to 30 and then a substantial settlement with a slight decrease. The graphs have similar trends.
We can deduce that around the value 30 the precision of our algorithm will be higher. However, it is not particularly high. To do this it will also be necessary to observe which are the best values for the other parameters that the algorithm is using.

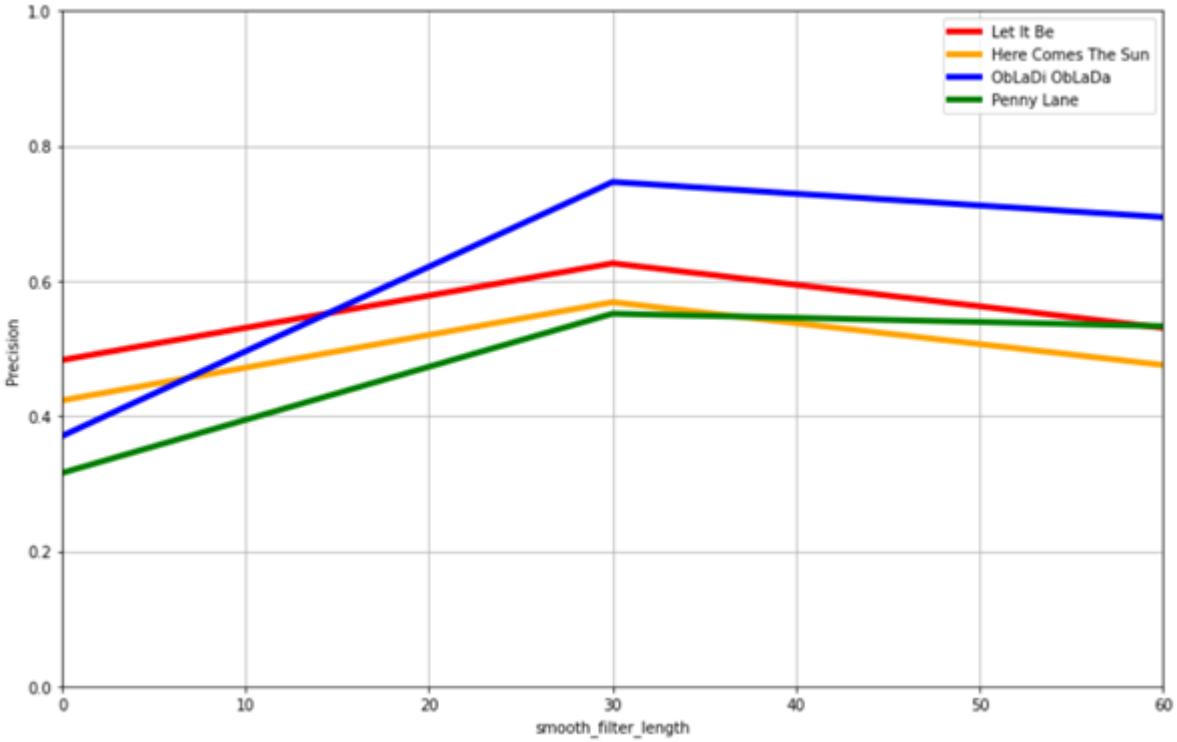


Figure 10: Smooth filter length

Down sampling factor. Often, after considering the smooth filter length, to increase the efficiency of subsequent processing and analysis steps, one decimates the smoothed representation by keeping only every H-th feature, where $H \in N$ is a suitable constant (typically much smaller than the window length L). This decimation, which is also referred to as downsampling, reduces the feature rate by a factor H.
We take into account these values: [0, 10, 100]: the more the downsampling factor is increased, the more you reduce the feature rate and the less you read the harmonic content (in larger steps). If chosen high, inconsistently with the window length, the precision decreases. It's good to use it in combination with the smoothing filter for efficiency. In this case 10 or less can be a good value.

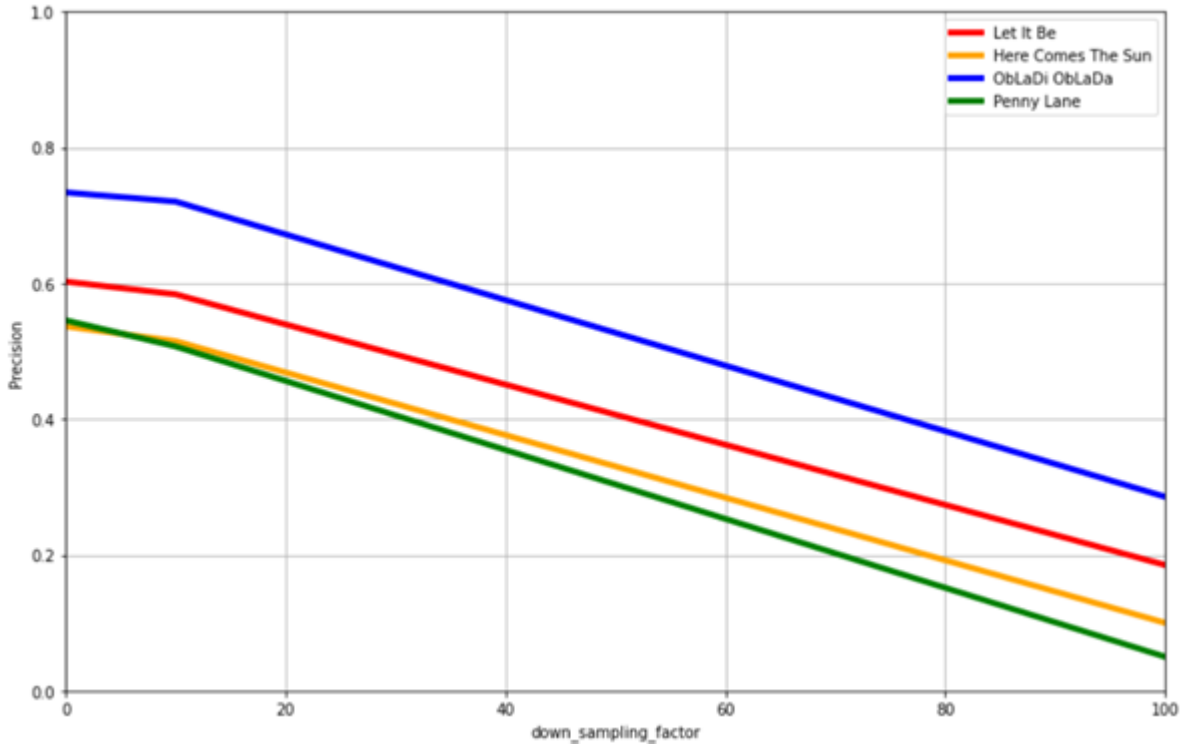


Figure 11: Down sampling factor

Window length. We consider the following vector of values of window lengths [2048, 8192, 32768]: using an analysis window with a short duration , each chroma frame contains the onsets of at most one note. Even though the sound of each note may last much longer than the notated duration, the harmonic content of each frame is dominated by only one or two notes. This explains the misclassifications and many chord label changes in the recognition result of the first setting.
An obvious strategy for improving the chord recognition result is to use larger window sizes. Anyway, the larger analysis windows smooth out the originally sharp transitions between different chords, which may introduce problems at chord changes.

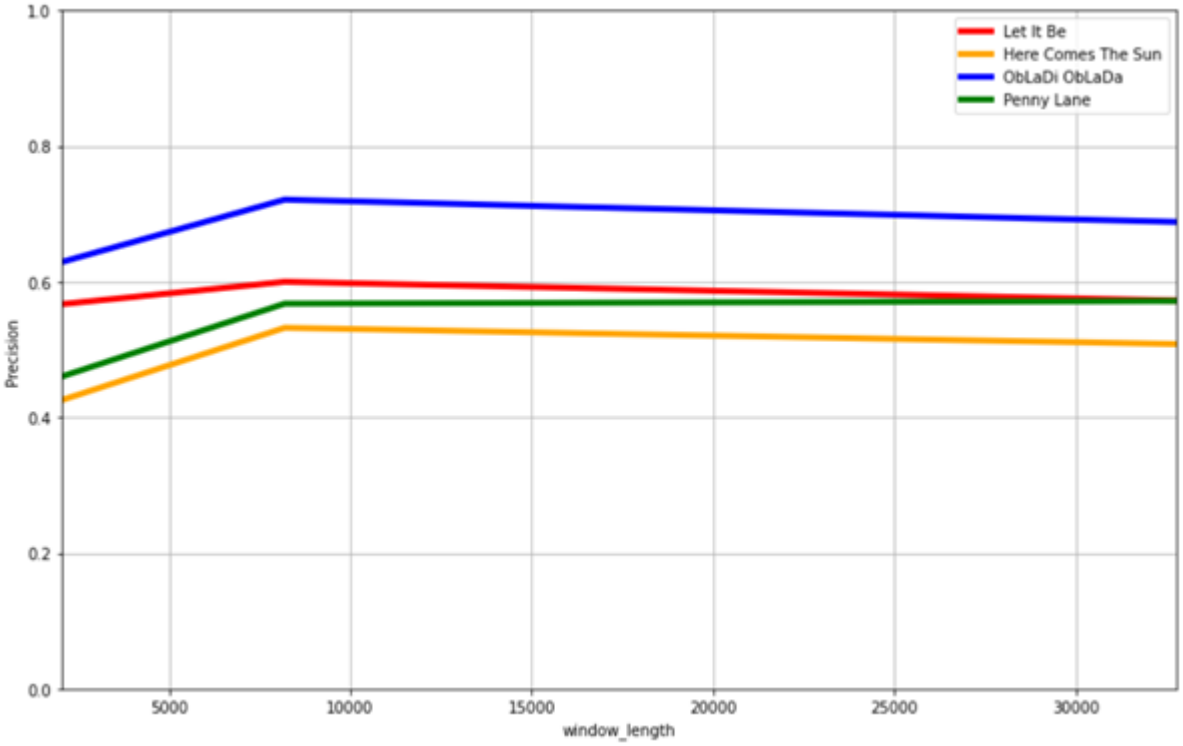


Figure 12: Window length