In [1]:
```
import os
import shutil # move files and delete folders with files
import tarfile
import urllib.request # download files folder
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np
import pandas as pd
import IPython.display as ipd
import librosa, librosa.display
import re
import IPython # listen to sounds on Python
import pretty_midi

from scipy.io import wavfile
from scipy.spatial import distance_matrix
from matplotlib import colors
from itertools import product
from hmmlearn import hmm
from sklearn.metrics import f1_score
from utils.annotations import read_simplify_chord_file, simplify_predicted_chords, get_chord_notes, create_si
from utils.hmm_helper import calc_transition_prob_matrix, \
                             get_mu_sigma_from_chroma, \
                             create_ISPM,\
                             get_hmm_predictions, \
                             calc_initial_state_prob_matrix, \
                             adapt_initial_prob_matrix
from utils.files_processing import download_uncompress_data_to_local, \
                                   filter_lab_files, \
                                   delete_download_file, \
                                   delete_download_folder
from utils.signal import plot_signal, \
                         plot_spectra, \
                         stft_audio, \
                         calc_chromagram, \
                         chromagram_2_dataframe, \
                         get_frame_stats, \
                         get_annotated_chord_sequence, \
                         smooth_chords_by_beat
from utils.evaluation import calc_classification_stats, plot_performance


%load_ext autoreload
%autoreload 2
```

Notebook for Pydata 2019 conference - https://de.pycon.org/program/pydata-yxndb9-hidden-markov-models-for-chord-recognition-intuition-and-applications-caio-miyashiro/ (https://de.pycon.org/program/pydata-yxndb9-hidden-markov-models-for-chord-recognition-intuition-and-applications-caio-miyashiro/)

Intro

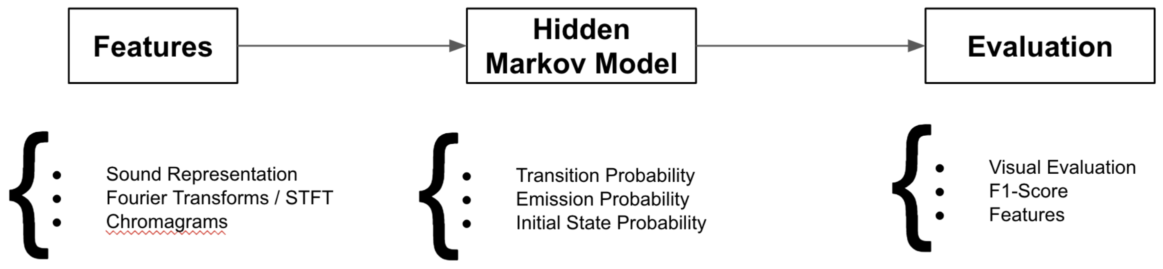
Welcome to the workshop! Today, you will learn the concept of **Hidden Markov Models (HMM)**, which is a probabilistic framework to work with sequence data, such as speech and language processing and GPS positions. The main idea is to show everyone What is a HMM and provide them with sufficient basis so you can use it in your next projects. The concept will be presented with a specific story and application in music:

- "Given an input music signal, can we identify which chords were played and in which sequence?"

Context

> Imagine you're listening to youtube and suddenly hear a music that you enjoy. A lot. You feel you want to take your guitar and play around with your family and friends. However, as you search on the internet, you find no suitable chords on that song or any how-to-play that could make your get-together happier. What now?

Music is a complex signal. Multiple voices, instruments and pre and post signal processing makes it difficult that we can get the features we want. Therefore, in order to work in our chord recognition problem, we need to understand a few processing techniques that'll make our lives "easier" for the task. Therefore, this is a small diagram covering the topics that we'll see today:



Overview

How is sound stored and represented by a computer?

Lets take a clear sound of a piano playing a A3 and display it:

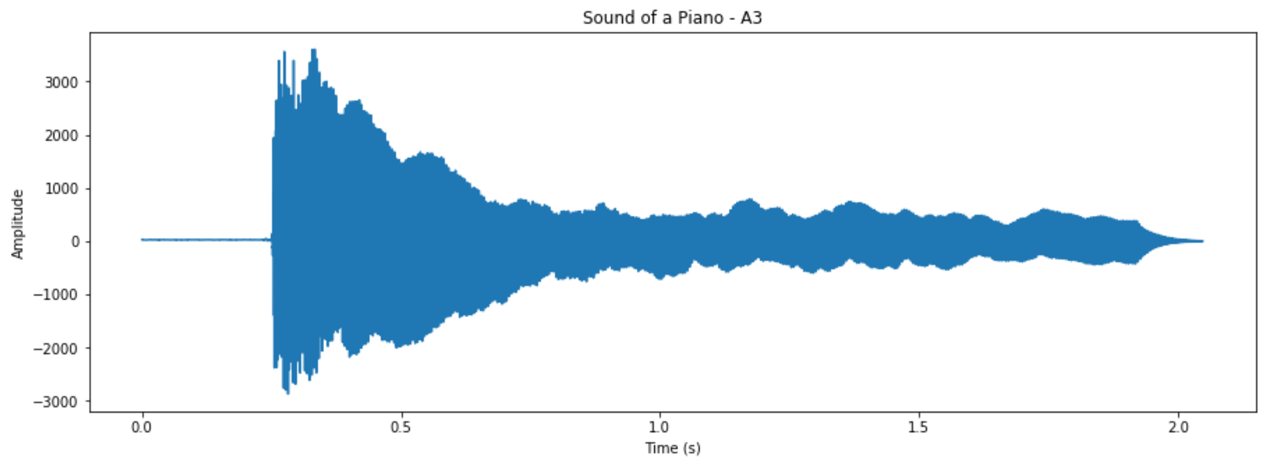
In [2]:
```
# https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.io.wavfile.read.html
Fs, x = wavfile.read("sounds/pianoA3.wav")
IPython.display.Audio(x, rate=Fs)
```

Out[2]:

```
0:00 / 0:02
```

The signal is represented with time on the x-axis and amplitude on the y-axis, where amplitude could be seen as the air pressure compression.

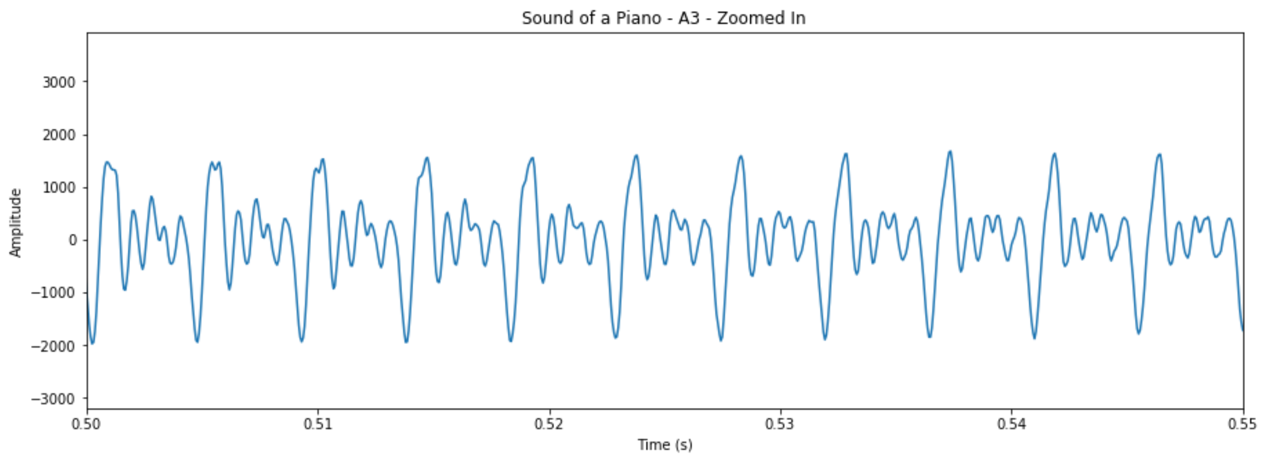
In [3]:
```
plt.rcParams["figure.figsize"] = (15,5)
plt.rcParams["font.size"] = 10

plot_signal(x, Fs, title='Sound of a Piano - A3', xlabel='Time (s)', ylabel='Amplitude')
```



Zoomed Signal:

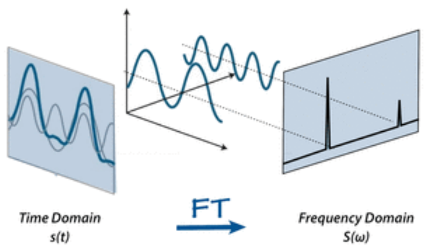
If we look close enough, **from 0.5 to 0.55 seconds more exactly**, we can see that the signal is quite periodic. This periodicity represent one of the most basic sound qualities: the periodic variation of air pressure.

In [4]:
```
plot_signal(x, Fs, title='Sound of a Piano - A3 - Zoomed In', xlabel='Time (s)', ylabel='Amplitude')
plt.xlim([.5,.55]);
```



From Time to Frequency Domain - The Fourier Transform (FT)

In western music, when we play a chord in our instrument, we are playing a set of specific notes. For instance we can differentiate a C minor from a C major by it's third (E or Eb). But, what if we could decompose any kind of sound in its respective notes (frequencies)? That's basically the Fourier transform. Despite its fancy name, this transformation basically just answer you the following question: " `What are the frequencies that, together, compose the sound I'm seeing?` "



http://mriquestions.com/fourier-transform-ft.html (http://mriquestions.com/fourier-transform-ft.html)

Whistle

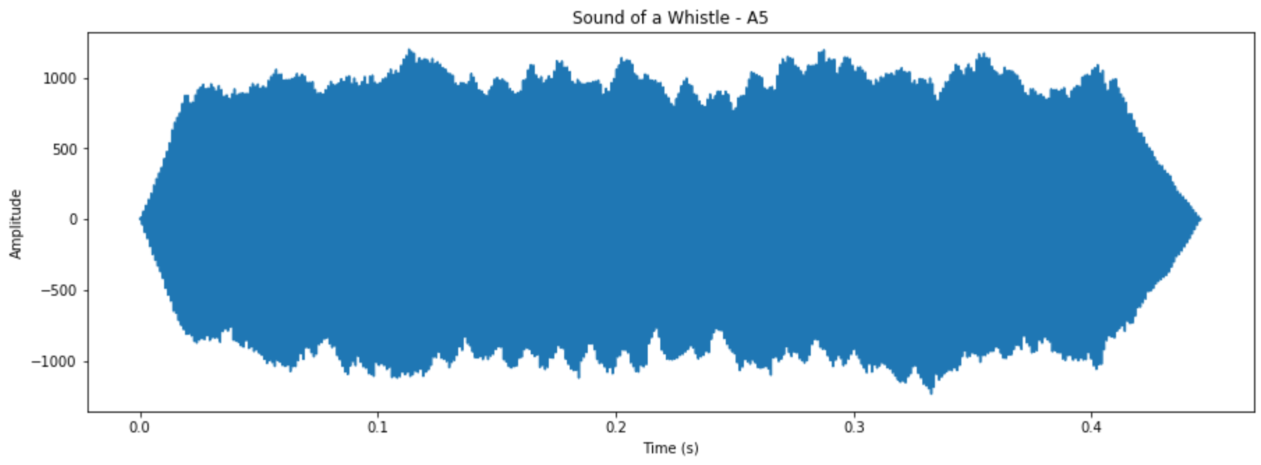
The whistling is at an A5, which is around ~880Hz (http://pages.mtu.edu/~suits/notefreqs.html). When we perform the fourier transform on it, we can that the highest peak is at this frequency. There is some energy amplitude also around the peak frequency and that's probably because we don't produce a perfect A5 whistle, but rather some variational and approximated version of it.

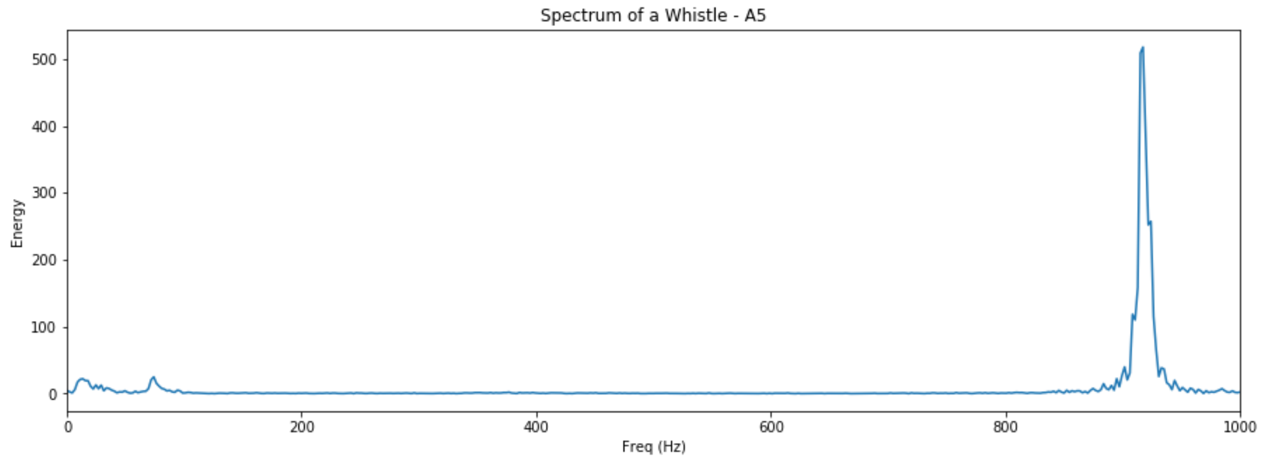
In [5]:
```
Fs_2, s_2 = wavfile.read("sounds/whistleA5.wav")
IPython.display.Audio(s_2, rate=Fs_2)
```

Out[5]:

0:00 / 0:00

In [6]:
```
plot_signal(s_2, Fs_2, title='Sound of a Whistle - A5', xlabel='Time (s)', ylabel='Amplitude');
plt.show()
plot_spectra(s_2, Fs_2, max_freq=1000, title='Spectrum of a Whistle - A5', xlabel='Freq (Hz)', ylabel='Energy
```





The FT of signal takes a "picture" of the whole signal and extract the frequency components out of it. This is good when we want to analyse the signal as a whole. However, sometimes we want to evaluate parts of it in sequence, such as in chord recognition, where the chords in a time window **will** be different sometime in the past or in the future. In order to have the fourier features for different parts of the music signal, we segment it in windows and apply the FT to each one of them: The **STFT**.

Short Time Fourier Transform (STFT)

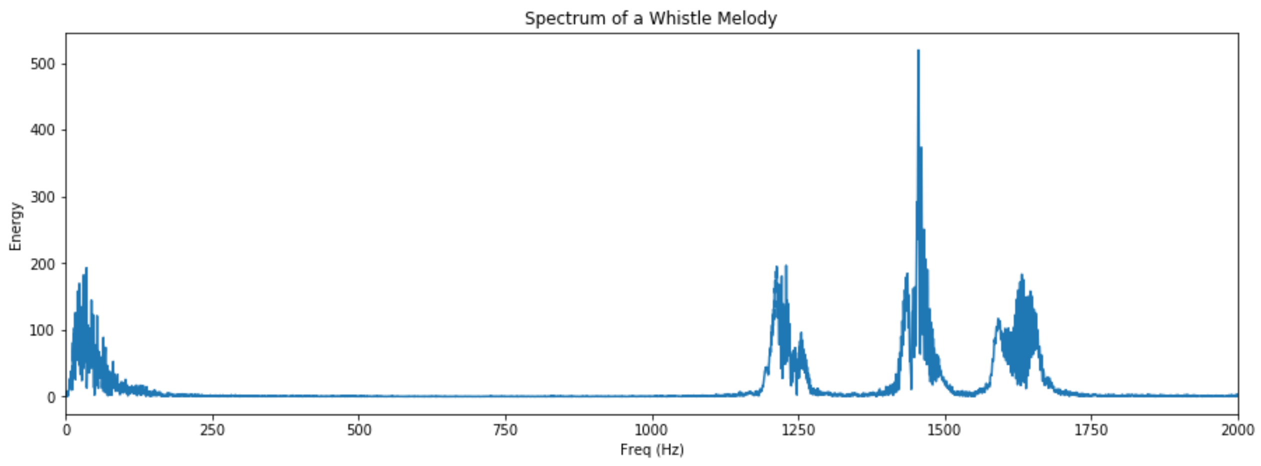
Lets take an example. If we hear the melody below, we clear hear different pitches being whistled in sequence. However, if we take the FT of it, all frequencies will appear on the same "picture" and we'll lose information of who occurred first.

In [7]:
```
Fs_3, s_3 = wavfile.read("sounds/whistleMelody.wav")
IPython.display.Audio(s_3, rate=Fs_3)
```

Out[7]:

```
0:00 / 0:02
```

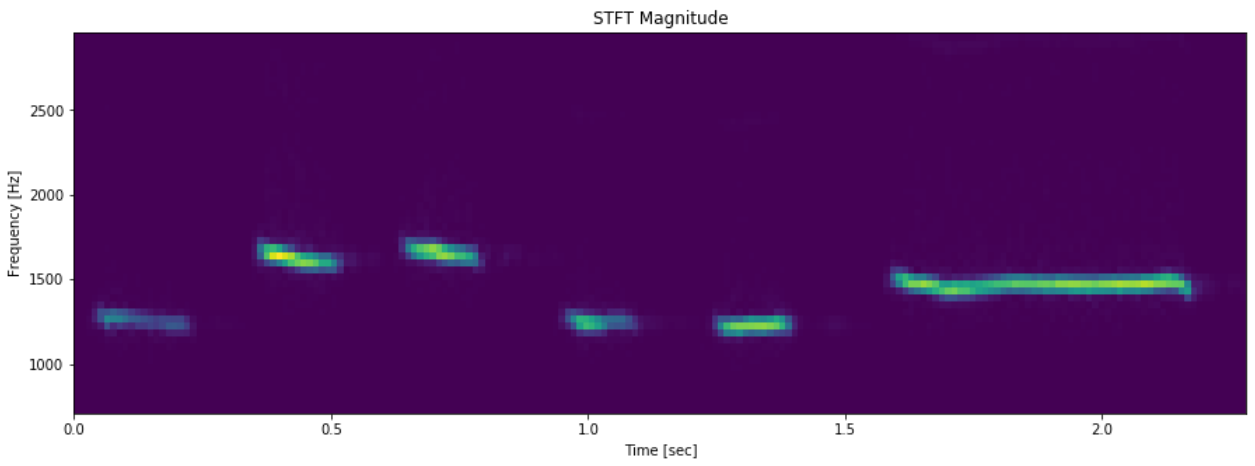
```
In [8]: plot_spectra(s_3, Fs_3, title='Spectrum of a Whistle Melody', xlabel='Freq (Hz)', ylabel='Energy', max_freq=2
```



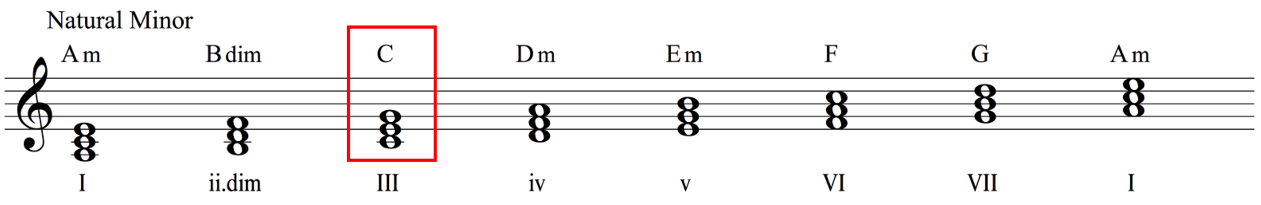
In order to visualise the STFT we need a different approach than FT. STFT has a new time dimension, while keeping the same frequencies dimensions from the FT. If we "bring" the right hand side of the FT plot (like above) to the top and concatenate them side by side, being each new item a FT of a time window, we can visualise this kind of information.

- **X axis** will be time
- **Y axis** is the frequency
- color (**Z-axis**) is the energy, *i.e.*, the peak heights from the previous FT plots:

```
In [9]: _, _, _ = stft_audio(s_3, Fs_3, True)
```



Chords, what are they?



https://www.musical-u.com/learn/discovering-minor-chord-progressions-minor-chords-part-one/ (https://www.musical-u.com/learn/discovering-minor-chord-progressions-minor-chords-part-one/)

We've talked a lot on the theory of the FT and STFT but not about our main task objective on identifying chords. After all, `what are chords?` .

Chords are a set of 3 or more notes played together [2] (https://www.coursera.org/learn/edinburgh-music-theory). These 3 notes are usually related to each other somehow (you can't play random 3 notes and call it a chord) and, depending on the notes choice made, they will have a specific role in your music, more specifically, in western music.

The most common chords in western music are the major and minor chords. Possibly, any music could be composed by just using them and most instruments are capable of producing them, except melodic instruments such as violins or flutes.
Chords can be pictured on a FT as multiple peaks from different notes played or as parallel lines in a STFT

Features from STFT for Chord Recognition - Chromagram

In music, we have different pitches having the same note name. A low pitch A, a higher pitch A, even higher A, all have the same name A, differing just by a distance of an octave (https://en.wikipedia.org/wiki/Octave). When we play a chord, in order to identify it, we don't need to know which octave of the note has been played, but only that the note itself has been played. Because of this feature, we can actually sum up the frequencies that belong to the same pitch. If we do that, we can have an idea of how much of each "note" we had on that time window, and this will be the **main feature** that will be used for our chord recognition problem.

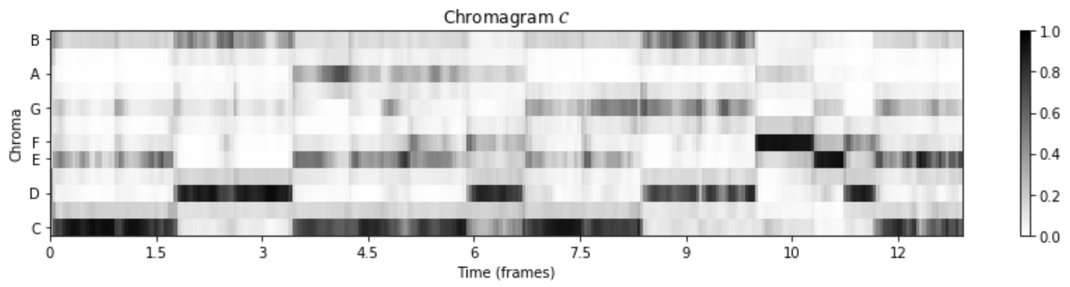
The technique takes the STFT of the music signal and, together with a table connecting frequencies to pitches (http://pages.mtu.edu/~suits/notefreqs.html), if finds the energy of the overall pitch representation, sum them up and normalise them so the energy values stays between the interval [0,1].

You can take a better look at the chromagram function (https://librosa.github.io/librosa/generated/librosa.feature.chroma_stft.html) from Librosa (https://librosa.github.io/librosa/index.html), which is the function we use to extract our chroma features.

```
In [10]: let_it_be_intro_path = 'sounds/Let it Be Intro.wav'
         x, Fs = librosa.load(let_it_be_intro_path)
         ipd.display(ipd.Audio(x, rate=Fs))
```

0:00 / 0:12

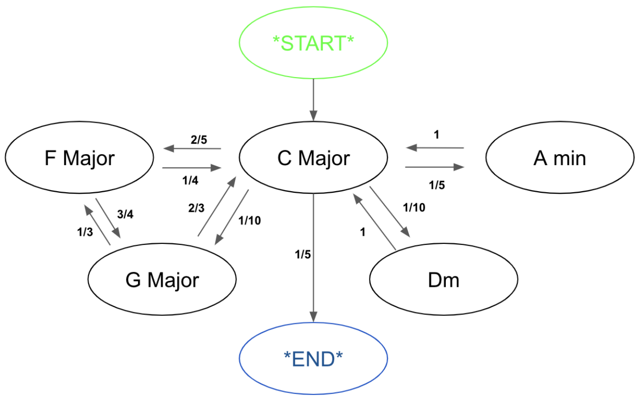
```
In [11]: _ = calc_chromagram(x, Fs)
```



We can see that until the first ~1.5 seconds, there is a predominance of the C, E and G notes (and a little of B). This set of notes compose the music's initial chord, a **C major** chord. There is energy spread around other notes in a smaller way, which are probably harmonics from the three actually pressed key notes of the piano. It seems it could be easy to identify the chords based on the chromagram. However, if we look at the following chord, there is a lot of energy around D, some more on B and almost nothing on G. This chord is actually a **G major** chord, but the D note was the lowest note in pitch on this chord, *i.e.*, an inverted chord. Therefore, we can't just look at the note with the biggest energy on it in order to classify the chord, we need more evidences. As it turns out, among chords, there's a high dependency of the current chord over the previous one, so we need a tool that helps us to deal with that, so next we'll introduce `Markov Chains` and `Hidden Markov Chains` .

Markov Chains - MC

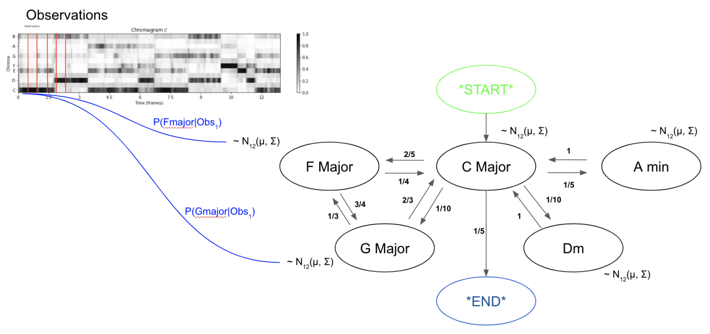
What is the chord on this time window given its chromagram and that the previous chord is a C major?



If we take as example the Let it Be intro, as we only analysed one music, we only have 1 C major chord after -START-. From there, we can go and go back to different chords until we end up at -STOP- and stop our chords prediction.

With a MC, if we have an initial chord, we can then predict what could be the next ones based on the previous chord. This would be already good, **if** we had the exact name of the played chord at hand. Remember that the input for our task today was a music signal. Behind those amplitudes and energies, the chord change chain is running behind it, "*hidden*". In order to be able to connect our chord chain with the windowed music signal, we need a extra feature, which is the feature of `Hidden Markov Models` .

Hidden Markov Models - HMM



$$best\_chord\_sequence = \underset{chord\_seq}{arg\_max}\mathbb{P}(chord\_seq|Obs) \approx \underset{chord\_seq}{arg\_max}\prod_{i=1}^{M}\mathbb{P}(Obs_i|chord_i)\mathbb{P}(chord_i|chord_{i-1})$$

Summary of main features of a Hidden Markov Model

Examples will be adapted from [4] (https://web.stanford.edu/~jurafsky/slp3/8.pdf) and this (https://www.youtube.com/watch?v=kqSzLo9fenk) youtube tutorial.

"A Markov chain is useful when we need to compute a probability for a sequence of observable events. In many cases, however, the events we are interested in are hidden hidden: we don't observe them directly. For example we don't normally observe part-of-speech tags in a text" [4] (https://web.stanford.edu/~jurafsky/slp3/8.pdf). Rather, we see audio signals, and must infer the chords out of it. We call the sequence hidden because they are not observed.

The main answer that a HMM can give us is:

- What is the most probable sequence of chords for a given sequence of observations?

In order to answer this question we will need a few things from the MC and some new features:

1. **Chord Transition Probability Matrix**: These are the notes probabilities explained in the MC section but having the chord transition probabilities instead.
2. **Emission probabilities**: Probability of an observation to belong to each one of the chords $\mathbb{P}(observation|chord)$.
3. **Initial State Probability Matrix**: Indicates what is the probability of a sequence to begin with a specific chord.

However, where we're going to extract these data from?

Annotation + Music Data - The Beatles

In order to be able to generate the probabilities above, we need:

- The music files in order to extract the chromagrams
- An annotaded dataset, so we can join the chord labels with the corresponding windowed chromagrams.

For the musics, we can obtain the music files by purchasing them online. For this workshop I'll be providing the files :)

For the annotaded dataset, luckily, some researches invest time on creating and publishing them, so they can be used as standards for a specific research domain or used as machine learning models tasks baselines. Here, we'll investigate the dataset generated by Harte (2010)[5] (https://code.soundsoftware.ac.uk/attachments/download/330/chris_harte_phd_thesis.pdf). Harte, together with other musicians and researches, manually extracted features from 120 musics from The Beatles, including beats, key changes and chords. You can download all annotations on their website (http://www.isophonics.net/content/reference-annotations-beatles)

For this workshop, I've already download, processed and joined them together with their respective musics and they're in the `lab_and_music` folder, together with their respective annotation files.

Read Annotations

In [12]:
```
DEST_FOLDER = 'lab_and_musics'

lab_file_path = f'{DEST_FOLDER}/Let_It_Be.lab'
chords_annotation = read_simplify_chord_file(lab_file_path,process_silence=True)
chords_annotation.head()
```

Out[12]:

	start	end	chord
0	0.000000	0.175157	C
1	0.175157	1.852358	C
2	1.852358	3.454535	G
3	3.454535	4.720022	A:min
4	4.720022	5.126371	A:min

Calculate framed Chromagram and join with chord labels

In [13]:
```python
x, Fs = librosa.load(let_it_be_intro_path)
let_it_be_chromagram = calc_chromagram(x, Fs, plot=False)
frames_per_sec, frame_duration_sec = get_frame_stats(chromagram=let_it_be_chromagram,
                                                     signal=x,
                                                     Fs=Fs)

pcp = chromagram_2_dataframe(let_it_be_chromagram, frame_duration_sec, test_version=True)

pcp['chord'] = get_annotated_chord_sequence(pcp, chords_annotation, test_version=True)
pcp.head()
```

Out[13]:

	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	start	end
0	0.332066	0.384424	0.431916	0.414366	0.131415	0.091793	0.108283	0.152824	0.190040	0.224761	0.313280	0.371681	0.000000	0.046419
1	0.664177	0.352761	0.109753	0.110625	0.366218	0.163517	0.087432	0.182613	0.090595	0.068443	0.157639	0.412985	0.046419	0.092837
2	0.754787	0.184479	0.045408	0.124241	0.429377	0.168346	0.106199	0.206016	0.082103	0.026816	0.040539	0.321305	0.092837	0.139256
3	0.734501	0.146538	0.047722	0.111363	0.507049	0.167623	0.137559	0.243638	0.065114	0.015817	0.024087	0.236238	0.139256	0.185674
4	0.833918	0.198937	0.056337	0.078762	0.421176	0.114144	0.089199	0.182769	0.042916	0.006769	0.013399	0.147586	0.185674	0.232093

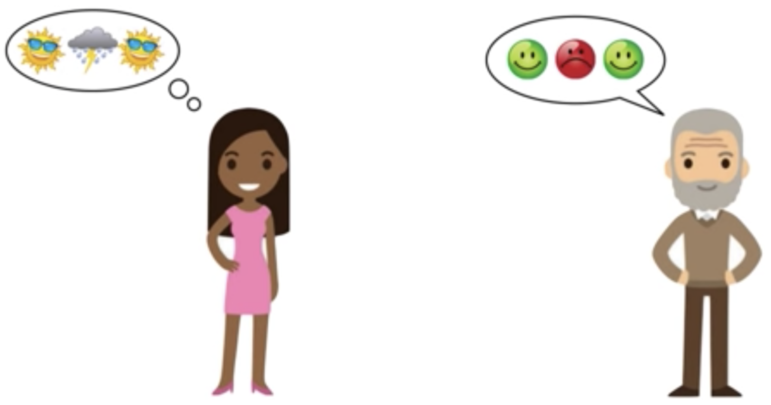
Calculate State Transition Matrix

In [14]:
```python
transition_matrix = calc_transition_prob_matrix(pcp, test_version=True)
transition_matrix
```

Out[14]:

sequence_chord	A:min	C	F	G
initial_chords				
A:min	0.972222	0.000000	0.027778	0.000000
C	0.000000	0.980198	0.000000	0.019802
F	0.000000	0.028169	0.971831	0.000000
G	0.014493	0.000000	0.014493	0.971014

Emission Probabilities



A friendly introduction to Bayes Theorem and Hidden Markov Models - https://www.youtube.com/watch?v=kqSzLo9fenk&t=1728s (https://www.youtube.com/watch?v=kqSzLo9fenk&t=1728s)

If you recall from the intro to HMM, **the emission probabilities connect our observations (visible) with our probabilistic network (hidden)**, showing the probability of an observation belonging to a state (chord) $\mathbb{P}(obs|state)$. In order to calculate them, we have to think differently based on the type of the observation we have:

- **Categorical**
- **Continuous**

Emission Probabilities for categorical observations

- Sad: {'Sunny': 100, 'Rainy':400}
- Neutral: {'Sunny': 300, 'Rainy':300}
- Happy: {'Sunny': 500, 'Rainy':100}

We just need to normalise our counts so we can have our **emission probabilities**. So first we calculate the sum of each variable as a normalisation factor:

- Sum of observations in "Sad" = 100 + 400 = 500
- Sum of observations in "Neutral" = 200 + 200 = 400

- Sum of observations in "Happy" = 500 + 100 = 600

And apply the normalisation factor to our conts in order to have probabilities:

- Sad: {'Sunny': 100/500, 'Rainy':400/ 500}
- Neutral: {'Sunny': 200/400, 'Rainy':200/400}
- Happy: {'Sunny': 500/600, 'Rainy':100/600}

=

- Sad: {'Sunny': 0.2, 'Rainy':0.8}
- Neutral: {'Sunny': 0.5, 'Rainy':0.5}
- Happy: {'Sunny': 0.83, 'Rainy':0.17}

Emission Probabilities for continuous observations

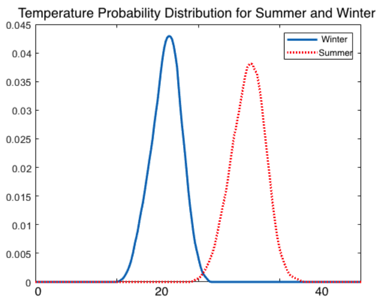
Impossible to count values as :

- $\mathbb{P}(temp = 20.7|sunny)$ **or**
- $\mathbb{P}(temp = 20.07|sunny)$ **or**
- $\mathbb{P}(temp = 20.007|sunny)$ **and so on...**

Need to do some **prior** assumptions on how our states looks like considering our variables of interest, such as *temperature* in this example.

> "Summer days in São Paulo - Brazil tend be between 30 and 40 degrees celsius"

> "Winter is a summer day that didn't work. Temperatures can be between 15 to 25 degrees celsius"



Example of Probability Distribution for Temperature

If we're calculating probabilities over continuous variables, we have to assume our data follows a specific distribution. For this chord recognition task and this temperature example, we'll assume that our observations follow a **normal distribution** [6] (https://pt.coursera.org/lecture/probability-intro/normal-distribution-M71Nv). Assuming a normal distribution for our temperature example states that:

- The temperature on different weather statuses has an average temperature
- The temperature can vary around this mean value, and the scale of the deviation from the mean value is defined by its variance.

and the formula to calculate the probability of observing a temperature (close to) t given the weather w defined by its mean $\mu_{weather}$ and variance $\sigma^2_{weather}$ is:

$$\mathbb{P}(t|\mu_{weather}, \sigma^2_{weather}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(t-\mu)^2}{2\sigma^2}}$$

oooor we call the equivalent python function in scipy.stats.norm.pdf (https://docs.scipy.org/doc/scipy-0.16.1/reference/generated/scipy.stats.norm.html) to check for that :D

Emission probabilities for chords and chromagrams

> Usually for a C major chord, we have this chromagram picure

The temperature example could work if we only had 1 variable for every state. Instead, in our chromagram, we have 12 variables, the energy of each one of the notes of the chromatic scale (A, A#, B, C and so on...). How do we do a probability of 12 variables belonging to a certain chord?

Lets think on the question, `"What is the probability of a chromagram showing a C major?"`. In this case, we'd expect a high energy on the notes `C`, `E` and `G` (major triad) and supposedly nothing else. The C major chord expect that the three notes have a high energy and expect that, if one is higher than another, which could happen, the chord would have some tolerance because it knows the energy of each note can vary **and** the energy on one note compared to other notes can also vary. In this case, the 12-variables probability distribution is just an generalisation of the original normal distribution we've seen, called **multivariate gaussian distribution** (https://en.wikipedia.org/wiki/Multivariate_normal_distribution). Instead of having one expected μ and σ^2 for each chord, we'll have a set of 12x μ and a set of σ^2 for each note and an extra statistic showing how the notes' energies vary among themselves, called covariance (https://pythonfordatascience.org/variance-covariance-correlation/).

Finally, the assumption we're going to make for each chord's note energy distribution will be taken from the data, *i.e.*, we will calculate the notes' mean and covariance from each labeled chord of our chromagram.

In the same way for one dimension, calculating the probability $\mathbb{P}(chromagram|\mu_{chord}, \sigma^2_{chord})$ has a specific (hard) equation (https://en.wikipedia.org/wiki/Multivariate_normal_distribution#Density_function). Lucky for us, we'll leave this responsibility to the HMM package we're going to use, so we just need to calculate the multivariate gaussian parameters. So lets go for it:

Calculate Emission Probabilities

Calculate distribution assumptions of each chord according to the mean energy in the labeled chromagram.

- For a HMM with M states, `mu_array` will be shape [M, 12], where 12 are the 12 notes from the chromagram. It'll show the average energy in each one of the 12 notes for each of the chords.
- For each chord of our HMM, we'll have a matrix of shape [12,12] for each note. It'll tell the HMM how, in each chord, the notes vary among themselves. For example, in a **C major** chord, we expect that, when the `C` note is high, the notes `E` and `G` are expected to be high as well. In this condition, `states_cov_matrices` is of shape [M,12,12]. The function `get_mu_sigma_from_chroma` runs a `for` loop on every chord and their respective chromagrams and then calculates the mean energy and their covariance.

In [15]:
```
mu_array, states_cov_matrices = get_mu_sigma_from_chroma(pcp)
# states_cov_matrices[0:2] = 0 # turn cov matrices of <START> and <END> to from nan to 0
```

Calculate Initial State Probability Matrix - ISPM

In [16]:
```
initial_state_probs = calc_initial_state_prob_matrix(process_silence=True)
# display(initial_state_probs[:5])

initial_state_probs = adapt_initial_prob_matrix(initial_state_probs, transition_matrix)
display(initial_state_probs)
```

```
A:min    0.000000
C        0.529412
F        0.058824
G        0.411765
dtype: float64
```

Initiate and Predict with Hidden Markov Model

We're **finally** ready to build our HMM! For that, we're going to use the hmmlearn (https://hmmlearn.readthedocs.io/en/latest/) python package. This package abstract a few complicated mathematics for us, leaving an interface similar to sklearn machine learning packages, where we build and then predict over new observations.

Exclusively for this function, I'll leave it on the notebook so we can better build and evaluate it. Lets take a look:

In [17]:
```
def build_gaussian_hmm(initial_state_prob, transition_matrix, mu_array, states_cov_matrices):
    # continuous emission model
    h_markov_model = hmm.GaussianHMM(n_components=transition_matrix.shape[0], covariance_type="full")

    # initial state probability
    h_markov_model.startprob_ = initial_state_prob
    # transition matrix probability
    h_markov_model.transmat_ = transition_matrix.values

    # part of continuous emission probability - multidimensional gaussian
    # 12 dimensional mean vector
    h_markov_model.means_ = mu_array
    # array of covariance of shape [n_states, n_features, n_features]
    h_markov_model.covars_ = states_cov_matrices
    return h_markov_model

h_markov_model = build_gaussian_hmm(initial_state_probs, transition_matrix, mu_array, states_cov_matrices)
```

We see below that the HMM doesn't save the state label (chord name). The states are saved on the same order they appeared in the transition probability matrix. Therefore, in order to account for that, we need to reconvert the numbers to the corresponding chords.

At the end, we have the predicted chords for our sequence of chromagrams :D

In [18]:
```python
COL_NAMES_NOTES = ["C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"]

chord_ix_predictions = h_markov_model.predict(pcp[COL_NAMES_NOTES])
print('HMM output predictions:')
print(chord_ix_predictions[:50])

# create map between numerical state index and chord
chord_numbers = range(len(mu_array.index.values))
chords = mu_array.index.values
ix_2_chord = {ix_: chord_str for ix_,chord_str in zip(chord_numbers,chords)}

chord_str_predictions = get_hmm_predictions(chord_ix_predictions, ix_2_chord)
print('Translated chords HMM output predictions:')
print(chord_str_predictions[:50])

pcp['predicted'] = chord_str_predictions
```

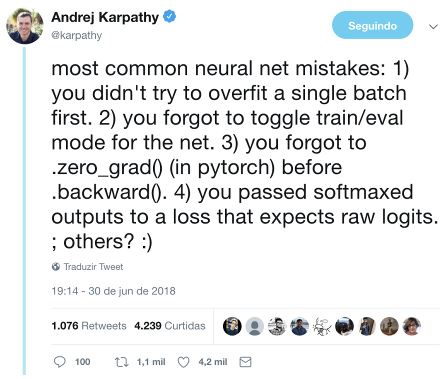
```
HMM output predictions:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 3 3 3 3 3 3 3 3 3 3 3]
Translated chords HMM output predictions:
['C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C'
 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C'
 'C' 'C' 'C' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G']
```

Visual Evaluation

Calculate True Positive, False Positive and False Negatives

Looking below, we can see that our HMM manage to correctly predicts the 12 seconds of the music "Let it Be" from the Beatles. Although we can argue this is just one single music and from a piece which just 1 instrument plays, **this result is quite important to understand that our model is actually "learning" something, or that the features bring some value for the prediction**.

I like Andrej Karpathy's twitter on Deep Neural Networks. Although far from being similar models, I found it a valid tip for every new ML technique you try :)



Andrej Karpathy
@karpathy
Seguindo

most common neural net mistakes: 1) you didn't try to overfit a single batch first. 2) you forgot to toggle train/eval mode for the net. 3) you forgot to .zero_grad() (in pytorch) before .backward(). 4) you passed softmaxed outputs to a loss that expects raw logits. ; others? :)

Traduzir Tweet

19:14 - 30 de jun de 2018

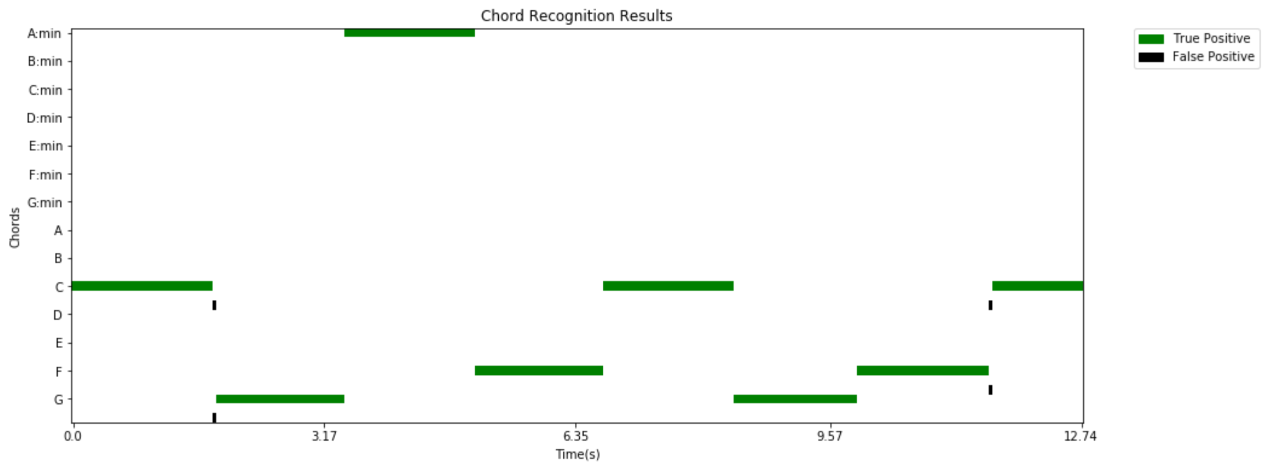
1.076 Retweets 4.239 Curtidas

100 1,1 mil 4,2 mil

https://twitter.com/karpathy/status/1013244313327681536 (https://twitter.com/karpathy/status/1013244313327681536)

In [19]:
```python
true_positives, false_positives, false_negatives, f1_score_= calc_classification_stats(pcp)

print(f'F-Score: {round(f1_score_,2)}')
plot_performance(true_positives, false_positives, false_negatives, frame_duration_sec)
```

```
F-Score: 0.99
```



Execute predictions

Based on the window prediction, we can actually regenerate the sound as a midi file.

If two adjacent windows have the same predicted chord, we join them both and adjust the `start` and `end` time to encompass for them.

Finally, we use pretty-midi (https://github.com/craffel/pretty-midi) python package in the `create_simple_midi` in order to generate the sounds.

In [20]:
```
chords_simplified = simplify_predicted_chords(pcp)
chords_simplified
```

Out[20]:

	predicted	start	end
0	C	0.000000	1.810323
1	G	1.810323	3.481390
2	A:min	3.481390	5.152458
3	F	5.152458	6.777107
4	C	6.777107	8.448174
5	G	8.448174	10.026404
6	F	10.026404	11.743890
7	C	11.743890	12.904354

In [21]:
```
create_simple_midi(chords_simplified, tempo=librosa.beat.tempo(x, Fs)[0])
```

Out[21]:

0:00 / 0:13

In [22]:
```
ipd.display(ipd.Audio(x, rate=Fs))
```

0:00 / 0:12

Try with whole song

Build Pipeline

In [23]:
```
def build_chroma(song_path, process_silence=True, test_version=False):

    # input data -> signal, sample frequency, chromagram and annotated dataset
    x2, Fs2 = librosa.load(song_path)
    C2 = calc_chromagram(x2, Fs2, False)
    frames_per_sec, frame_duration_sec = get_frame_stats(C2, x2, Fs2)
    annotation_file_path = f"{song_path.split('.')[0]}.lab"
    chords_annotation2 = read_simplify_chord_file(annotation_file_path,process_silence=process_silence)

    pcp2 = chromagram_2_dataframe(C2, frame_duration_sec, test_version=test_version)
    pcp2['chord'] = get_annotated_chord_sequence(pcp2, chords_annotation2, test_version=test_version)
    return x2, Fs2, pcp2

def calc_hmm_matrices(chromagram, process_silence=True, test_version=False):

    ##### HMM parameters
    # Transition Matrix
    transition_matrix2 = calc_transition_prob_matrix(chromagram, test_version=test_version)
    # Adapted Initial Probability Matrix
    init_states2 = calc_initial_state_prob_matrix(process_silence=process_silence, annotations_folder_path='l
    filtered_initial_states2 = adapt_initial_prob_matrix(init_states2, transition_matrix2)
    # mu and covariance matrices
    mu_array2, states_cov_matrices2 = get_mu_sigma_from_chroma(chromagram)
    return filtered_initial_states2, transition_matrix2, mu_array2, states_cov_matrices2

def get_predictions(h_markov_model, prediction_dict, song_path, test_version=False):
    # get predictions
    signal, sr, chromagram = build_chroma(song_path, process_silence=True, test_version=False)
    chord_ix_predictions2 = h_markov_model.predict(chromagram[COL_NAMES_NOTES])

    chromagram['predicted'] = get_hmm_predictions(chord_ix_predictions2, prediction_dict)
    chromagram['predicted_cluster'] = smooth_chords_by_beat(chromagram, signal, sr, predicted_col='predicted'
    return chromagram

def evaluate(chromagram, only_f1=False):
    return calc_classification_stats(chromagram, predicted_col='predicted_cluster', only_f1=only_f1)

##### EXECUTION
def train_pipeline(songs, test_version=False):
    chromagram = pd.DataFrame()
    for song in songs:
        signal, sr, chromagram_one = build_chroma(song, test_version=test_version)
        chromagram = chromagram.append(chromagram_one, ignore_index=True)
    initial_states, transition_matrix, mu_array, states_cov_matrices = calc_hmm_matrices(chromagram, test_ver
    ix_2_chord = {ix_: chord_str for ix_,chord_str in zip(range(len(mu_array.index.values)),mu_array.index.va
    h_markov_model = build_gaussian_hmm(initial_states, transition_matrix, mu_array, states_cov_matrices)
    return h_markov_model, ix_2_chord
```

In [24]:
```
song_path = 'lab_and_musics/Let_It_Be.mp3'
h_markov_model, ix_2_chord = train_pipeline([song_path], test_version=True)

# new prediction
chromagram = get_predictions(h_markov_model, ix_2_chord, song_path)
print('1 Song F1-Score:')
evaluate(chromagram, only_f1=True)
```

```
1 Song F1-Score:
```

Out[24]:
```
0.6855649376759148
```

In [25]:
```
chords_simplified = simplify_predicted_chords(chromagram)
create_simple_midi(chords_simplified, tempo=100)
```

Out[25]:

0:00 / 3:51

In [26]:
```
x, sr = librosa.load(song_path)
ipd.display(ipd.Audio(x, rate=sr))
```

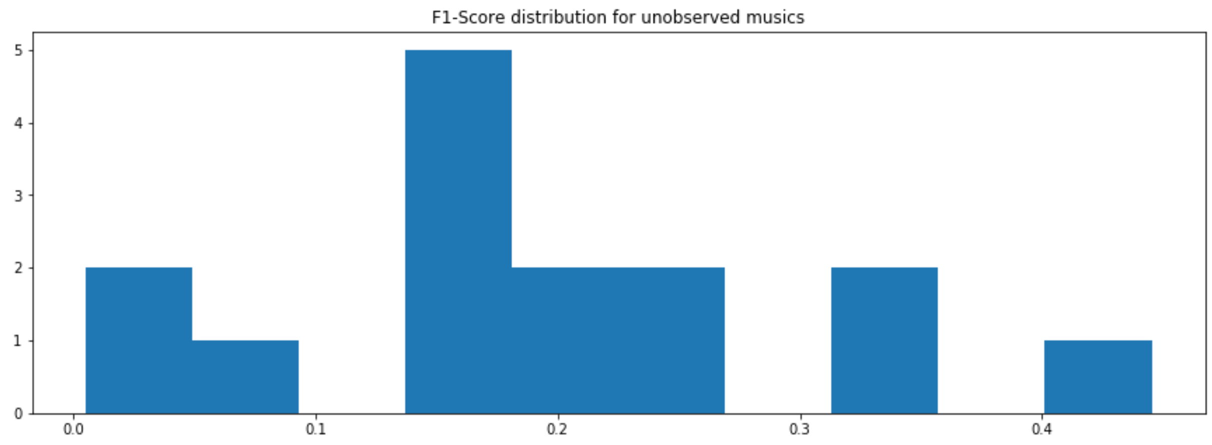
0:00 / 3:50

Predict

For the final combo, lets see how it goes with the 50 songs that you have available, together with their respective chord annotations.

In [27]:
```python
songs = []
for file_name in os.listdir('lab_and_musics'):
    if(file_name.endswith('.mp3')):
        songs.append(f'lab_and_musics/{file_name}')

h_markov_model, ix_2_chord = train_pipeline(songs[:35])
```

In [28]:
```python
ind_f1_score = []
for song in songs[-15:]:
    chromagram = get_predictions(h_markov_model, ix_2_chord, song)
    ind_f1_score.append(evaluate(chromagram, only_f1=True))

plt.hist(ind_f1_score)
plt.title('F1-Score distribution for unobserved musics');
```



Conclusion

For this first iteration, we've seen our HMM performs poorly in general. But we had lots of learnings for possible next steps!

- First of all, just by hearing the complete 'Let it Be' music, we can notice how the Voice and Drums hampers the chord recognition. Searching in the literature, in the source separation domain, it seems it's easier to extract drums out of the sound, like this Librosa function on Harmony Percussion Source Separation (https://librosa.github.io/librosa/generated/librosa.effects.hpss.html) - (HPSS).
- Secondly, at least in tradition rock/pop songs, chords tend to stay within the beats of a song, not changing every chromagram window. In this work, I secretly used a Librosa function on beat detection (https://librosa.github.io/librosa/generated/librosa.beat.beat_track.html) but more work could be devoted to this.
- Thirdly, a music theory based feature. Analysing the chord predictions, I've notices that a common error that the HMM makes is to confuse a note by its VI degree. This is more music especific but, in Harmonic fields, the first chord of the field and its corresponding VI degree have the same notes (http://www.simplifyingtheory.com/relative-minor-major/). For instance, a C major chord and a A minor are composed by the same 3 notes - `C`, `A` and `E`. This makes the transition matrix sometimes go from `C major` to `A:min` and, because the transition probability matrix doesn't have a path from `A:min` back to `C major`, it doesn't come back. Check the transition probability from the whole music evaluation with 'Let it Be'. In this case, we could adapt the transition probability matrix to allow a connection between every chord and their respective VI degree so that, in case of mistakes by the HMM, we leave a space so it can fix itself.
- Lastly, transition probability matrices could be different based on the key tonality of the music. If we manage to extract the music's tone, we could choose specific key-tonality transition probability matrices that could adapt to our chord sequence in a better way.
  - For instance a transition probability between a `C` and a `B` in a `C major` tone would be higher than the same transition but for musics in a `F major` tone (no note `B` in the scale)

In the end, it'd be all about the features :)

What could we do even more?

- Extract lyrics from websites and insert them into the beat + chord structure and produce a pdf with lyrics + chords
- Could we use music websites as annotaded chord songs? We would need to find a way to match the written chords with time beats and use them as proxy for chord start and end annotation.
  - With the bigger amount of songs, we could even think on LSTMs for chord prediction

Main References

[1] (https://www.amazon.com/Music-Data-Analysis-Foundations-Applications/dp/1498719562) - Music Data Analysis: Foundations and Applications (Chapman & Hall/CRC Computer Science & Data Analysis)
[2] (https://www.coursera.org/learn/edinburgh-music-theory) - Fundamentals of Music Theory - Coursera & The University of Edinburgh
[3] (https://hackernoon.com/from-what-is-a-markov-model-to-here-is-how-markov-models-work-1ac5f4629b71) - From “What is a Markov Model” to “Here is how Markov Models Work” - Accessed on 15.09.2019
[4] (https://web.stanford.edu/~jurafsky/slp3/8.pdf) - Speech and Language Processing. Jurafsky, D. & Martin, J. - Chapter 8 - Draft of August 27, 2019.
[5] (https://code.soundsoftware.ac.uk/attachments/download/330/chris_harte_phd_thesis.pdf) - Harte, 2010 - Towards Automatic Extraction of

Harmony Information from Music Signals
[6] (https://pt.coursera.org/lecture/probability-intro/normal-distribution-M71Nv) - Introduction to Probability and Data - Coursera & University Duke