AARES Project

Preliminaries

- In the first part of the project, we will use a dataset generated using the Spotify API. It contains features extracted and gathered for many songs. You can check the features and more information on the data in Appendix A.
- In the second part of the project, we will generate unique identifiers of songs. A folder with some songs is provided to generate a small dataset.
- Avoid using ChatGPT or any other large language model to finish the TP.
- The file `utils_projet.py` contains a series of useful functions for the project. Make sure to install the libraries `numba` and `librosa` for the second part of the project.
- The following exercises are a guide for the report. Feel free to add and try methods, techniques, or ideas as you would do in a context where you discover the data and want to know what you can do with it.

Part 1: Song classification and recommendation

In this part use : `spotify_dataset_train.csv` and `spotify_dataset_test.csv`. The `.csv` file for training contains a dataset of 25492 songs belonging to one of those 23 musical genres : r&b, rap, classical, salsa, edm, hip hop, trap, techno, jazz, metal, country, rock, reggae, latin, disco, soul, chanson, blues, dance, electro, punk, folk, pop.

Exercise 1 : Classification challenge

1. Analyze the data. What can you say about it?
2. Train some classifiers, analyze the results and draw conclusions.
3. Using your best classifier predict the musical genre of the 2833 songs of test dataset in order to compete for the challenge! The evaluation metric will be the F1 score with micro average.

Exercise 2 : Data analysis

In practice, the musical genre in Spotify is not that well filled. We are therefore going to work on another dataset in `spotify_dataset_subset.csv`.

1. Try to predict the "popularity" of a song.
2. How can you handle the class "genres"?
3. What can you show from these data? Patterns, visualization, interpretation...

Exercise 3 : Song recommendation

We have seen that we can group songs by their features. We might assume that two "close" songs, in a certain space, are therefore similar and equally appreciated by a user. We are going to work on another dataset in `spotify_dataset_recommendation.csv`.

1. Propose a method that returns a playlist (approx. 10 songs) given a song (or a list of songs).
2. How can you extend this to take into account all users experience?

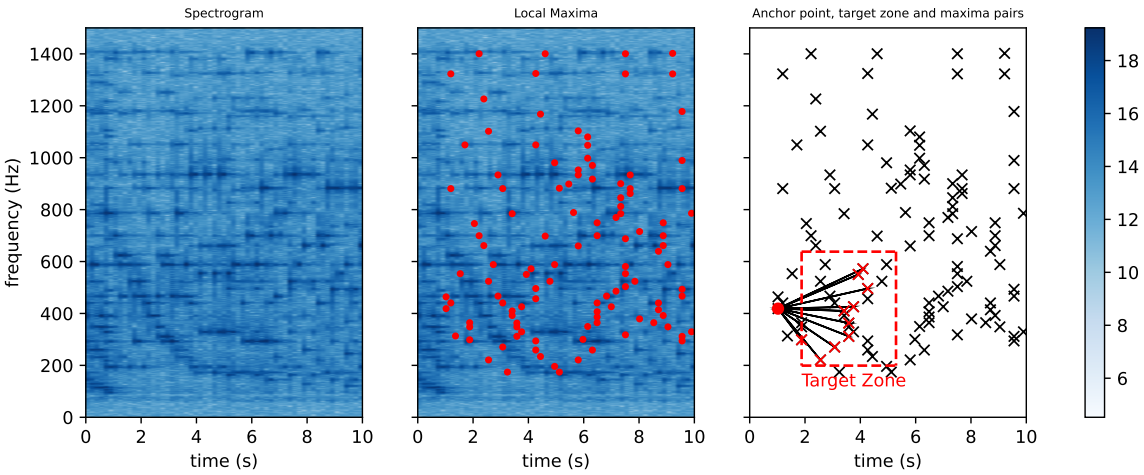

Figure 1: First Figure: Spectrogram corresponding to 10 seconds of a song. Second Figure: Spectrogram maxima as red dots. Third Figure: Spectrogram maxima are shown as crosses. The red dot is an anchor point (one of the maxima) and the red crosses in the Target Zone are the 10 highest maxima within it.

Remark. For the first part of the project you have to hand in 3 files (file names including your names separated with "_"):

1. One `.csv` file containing your prediction for the challenge, such as: the `spotify_test_dataset.csv` completed with a new column named "genre" or a new file containing only one column named "genre".
2. One `.pdf` file containing your report,
3. One `.zip` compressed archived with your code (`.py` or `.ipynb`).

Part 2: Audio fingerprints

Exercise 4: Produce unique identifiers for a song

1. Create a function called `compute_spectrogram(x)` that computes the spectogram of the signal `x` using `librosa.stft` with the following parameters: `win_length=1024`, `n_fft=2048`, `hop_length=512`. A reminder on the spectrogram is given in Appendix B. To load a song from a file you can use:

   ```
   >>> song, fs = librosa.load('path/to/song', sr=3000)
   ```

   where the parameter `sr` is the sampling rate.

2. Use the given function called `get_maxima(S)` to find local maxima in the spectrogram `S` and save their position, i.e. their time-frequency coordinates. For instance:

   ```
   >>> from utils_projet import get_maxima
   >>> maxima = get_maxima(S)
   ```

3. We will take each maximum as an *anchor point* and select a small region called *Target Zone* not far away from it, as shown in Fig. 1 (Third Figure). Implement a function called `get_maxima_in_tz(S,maxima,anchor)` that receives the spectrogram, a list with all the maxima locations and the anchor point. This function should do the following:

   (a) For the anchor point `anchor`, find a Target Zone of 300 pixels in time and 20 pixels in frequency.

   (b) Return a list with the 10 highest maxima in the Target Zone.

4. We need to create a *hash* that translates the maxima pairs into searchable objects. Create a function called `get_hashes(anchor,maxima_in_tz)` that receives an anchor point location

and the list of maxima in its Target Zone. The function should return a tuple following the next:

```
>>> hash = anchor[0]*1000000+point[0]*1000+point[1]-anchor[1]
>>> return hash, anchor[1]
```

where `point` is a tuple with the two coordinates of a maximum within the Target Zone corresponding to `anchor`.

5. Finally, create a function called `process_signal(x)` that does the following:
   (a) Get the spectrogram using the `compute_spectrogram(x)` function.
   (b) Get the maxima of the spectrogram using the `get_maxima(S)` function.
   (c) For each maxima:
      i. Get the maxima pairs in its Target Zone using the `get_maxima_in_tz(S,maxima,anchor)` function.
      ii. Now save the hashes of each maxima obtained by the `get_hashes(anchor,maxima_in_tz)` function.
   (d) In a Python dictionary, save all the hashes corresponding to the song as `hash:anchor[1]`.
   (e) Return the dictionary with the hashes.

Exercise 5: Create a database and search for a song.

1. Create the list of songs loading the files provided. For each song in the dataset apply the function `process_signal(x)` and get the dictionary of hashes. Save all the dictionaries in a list. In the end, you should have a list with one dictionary of hashes per song. You can use the library `pickle` to save and load the dataset in a file:

```
>>> import pickle
>>> with open('dataset.pickle', 'wb') as handle:
>>>    pickle.dump(dataset, handle)
```

   You can then load the dataset to memory from a file using:

```
>>> with open('dataset.pickle', 'rb') as handle:
>>>   dataset = pickle.load(handle)
```

2. Searching for a song
   (a) Get a short excerpt of a song (5 to 15 seconds).
   (b) Compute the hashes of the excerpt.
   (c) Using the function `search_song(db_hashes,song_hashes)` given in the file `utils_projet`, pass the list of hashes of the dataset and the hashes of the excerpt. It will return the indices of the list corresponding to the three more similar songs. Is the correct song among these three guesses? You can read more about how this works in Appendix C and [1].

Remark. For the second part of the project you have to hand in 2 files (file names including your names separated with "_").

1. A `.py` file including your implementation of the function `process_signal(...)` and related functions.
2. A short example showing the creation of the dataset of hashes from the songs, and how to search for an excerpt in the dataset (could be `.py` or `.ipynb`). Make sure to add comments in your code that show your understanding of what it is doing.

Bonus exercise: uncertainty in machine learning

How can we evaluate the uncertainty in our prediction? This is quite a relevant question in machine learning. For instance, we could ask ourselves how good is the prediction of the genre of a song that we have done in Exercise 1. One way of answering this question is by *conformal prediction*[2], which is a technique that allows one to produce predictions *intervals* based on the training data. For example, we can have a 95% prediction interval for the genre of a song that contains with probability 95% the right genre.

Conformal prediction is based on an *unconformity measure*. This is a function that measures how different a new example is from the training data and is based on the classification model. For this exercise, we will consider a K-nearest neighbors (KNN) classifier trained with the training data in the Spotify dataset (use the first 1500 examples, no more).

1. Create an object of the `ConformalPrediction` class by passing a KNN model from `sklearn` already trained with the training data as well as the training features and labels. This step will take a few minutes. For instance:

   ```
   >>> from utils_projet import ConformalPrediction
   >>> predictor = ConformalPrediction(KNNmodel,X,y)
   ```

2. Make it compute the predictions using the class method `predict` on a data point from the test dataset:

   ```
   >>> predictor.predict(x_new)
   ```

3. Now you can generate (1-`eps`)% intervals of prediction for the new data point using the class method `compute_interval(eps)`. For instance:

   ```
   >>> interval = predictor.compute_interval(eps)
   ```

4. Analyze the returned interval for `eps=0.05`. Does it include many genres ?

5. Increase the value of the input parameter `eps`. This will give you (1-`eps`)% intervals of prediction. Does the interval includes more or less genres as you increase `eps`? Are these genres "similar"?

6. Finally, increase `eps` until the interval includes only one genre. You can now say this prediction has (1-`eps`)% of confidence. Apply this procedure to several songs from the test set and evaluate the results.

A Features extracted for Exercises 1 to 3.

- Primary:
  - – id (Id of track generated by Spotify)
- Numerical:
  - acousticness (Ranges from 0 to 1)
  - danceability (Ranges from 0 to 1)
  - energy (Ranges from 0 to 1)
  - duration_ms (Integer typically ranging from 200k to 300k)
  - instrumentalness (Ranges from 0 to 1)
  - valence (Ranges from 0 to 1)
  - popularity (Ranges from 0 to 100)
  - tempo (Float typically ranging from 50 to 150)
  - liveness (Ranges from 0 to 1)
  - loudness (Float typically ranging from -60 to 0)
  - speechiness (Ranges from 0 to 1)
  - year (Ranges from 1921 to 2020)
- Binary:
  - mode (0 = Minor, 1 = Major)
  - explicit (0 = No explicit content, 1 = Explicit content)
- Categorical:
  - key (All keys on octave encoded as values ranging from 0 to 11, starting on C as 0, C# as 1 and so on)
  - artist_name (List of artists mentioned)
  - release_date (Date of release mostly in yyyy-mm-dd format, however precision of date may vary)
  - track_name (Name of the song)
  - genres (Multiple musical genres)
  - genre (Musical genre)

B The short-time Fourier transform and the spectrogram

For a signal $x \in L^2(\mathbb{R})$ and a window $g \in L^2(\mathbb{R})$, the STFT of f with window g is defined as

$$V_g^x(t,f) = \int_{-\infty}^{+\infty} x(t)\overline{g(u-t)}\, e^{-2i\pi uf} du$$

where the variables $t, f \in \mathbb{R}$ are usually referred as *time* and *frequency*.

The *spectrogram* of x is then given by:

$$S_g^x(u,v) = |V_g(f)(u,v)|^2$$

where S_g can be considered as a measure of how the *energy* of the signal $\|f\|^2 = \|V_g^x\|^2$ is spread across the time-frequency plane.

In practice, however, the spectrogram is represented as a matrix, i.e. an *image*,

$$\mathbf{S} = (s_{mn}) \in \mathbb{R}_+^{M\times N},$$

where $m \in \{0, 1, \ldots, M-1\}$ is the frequency bin and $n \in \{0, 1, \ldots, N-1\}$ is the time index. Thus $\mathbf{S} = s_{mn} \approx S_g^x(\frac{n}{N}T_\mathrm{s}, \frac{m}{2MT_\mathrm{s}})$ for a sufficiently small sampling period T_s.

If local maxima are *paired*, more data is available (trade-off between memory space and fast execution). Pairs also produce more specific tokens, that are easier to sort and quicker to search. So our song fingerprinting system will be based on looking for coinciding maxima pairs in the spectrogram.

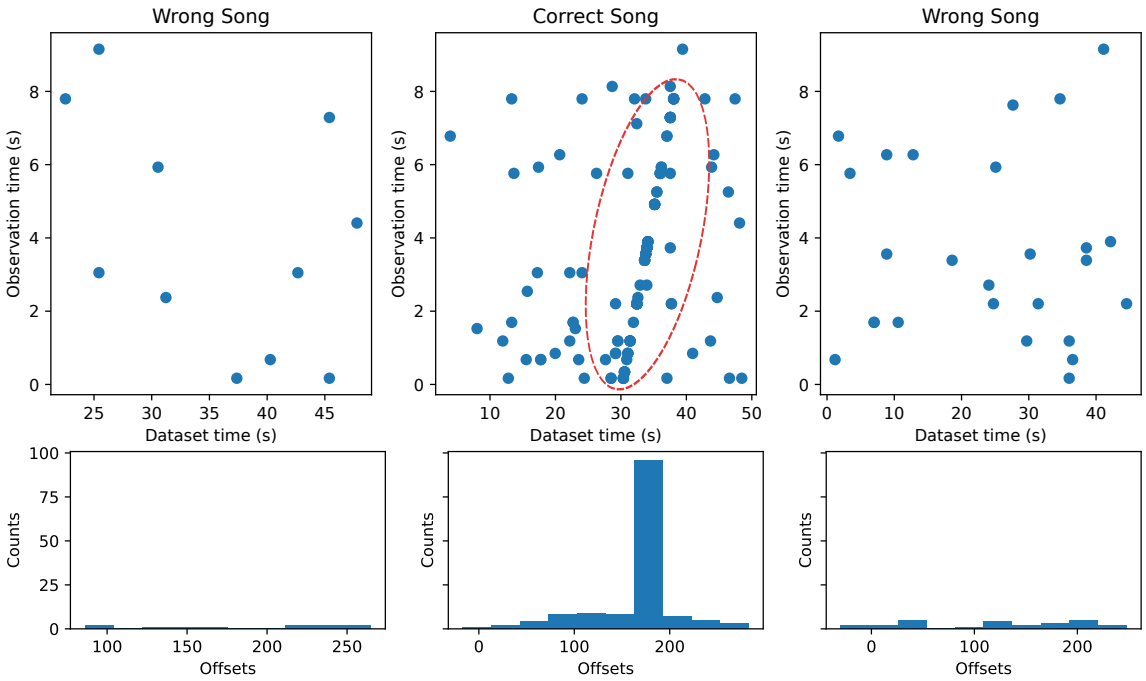

Figure 2: Each point in the plots show the location of a hash that has been found both in the sample excerpt (y axis) and the original song (in the x axis). For the correct song, a line of points appears. These points share the same offset, which is visible in the histogram below as a tall bar.

C Why it works?

- As shown in Fig. 2, a song excerpt can have hashes in common with any song in the dataset.
- However, for the correct song, a line of points appear in the plot, showing that the coincidences of hashes are correlated with the song.
- This means that the offsets corresponding to these points, i.e. the difference between the two coordinates of each point, is the same.
- In order to find many points with the same offset, one can make a histogram like those shown in Fig. 2. A tall bar is expected to appear for the correct song, while a mostly flat histogram is expected for the incorrect ones.
- In practice, this means that we can take the highest bar in the histograms as a score that indicates how well the excerpt matches a given song.
- Finally, taking the highest of these scores is tantamount to finding the song that matches the best with the excerpt.

References

[1] Wang, Avery Li-Chun "An Industrial-Strength Audio Search Algorithm." 2003.

[2] Shafer, Glenn and Vovk, Vladimir "A Tutorial on Conformal Prediction." 2008.