Report Homework 1

Computer Music Representation and models

Alice Portentoso

November 25, 2023

1 Introduction

The aim of this homework is to build a music genre classificator, that is able to classify music tracks into ten music genres using rhytmic features. We extract a precise feature vector and we use a dataset to train the classificator and to test it.

2 Implementation and analysis

2.1 Question 1

2.1.1 Load the dataset

The algorithm starts with loading the dataset. This is done with deeplake library, and the dataset used is GTZAN dataset.

```
import deeplake
dataset = deeplake.load("hub://activeloop/gtzan-genre")
```

2.1.2 Analyze the dataset

This dataset maps some audio files with their genre. It has two keys: `'audio'` and `'genre'`. `'audio'` is a tensor contains 1000 audio files. Each file is a 30 seconds duration music track sampled at 22050Hz Mono 16-bit in wav format. `'genre'` is a class label tensor which classify the music tracks into 10 classes, mapped in integer numbers as:

```
genre_names = ['pop', 'metal', 'classical', 'rock', 'blues', 'jazz',
    'hiphop', 'reggae', 'disco', 'country', 'pop']
```

2.1.3 Preprocess the dataset

There are 100 tracks for every genre in the dataset. We extract from this dataset 100 tracks for the training set, save them in `audio_train`, 20 tracks for the test set, saved in `audio_test`, and its relative genre classification label, saved in `genre_train` and `genre_test`.

The `audio_train` and `audio_test` numpy array have shape `(100, 639450)` and `(20, 639450)`. 639450 is the number of samples in each audio files, obained by `22050 * 29` where 22050 is the sampling frequency `Fs`.

```
genre_train = dataset['genre'][train_indices]
genre_test = dataset['genre'][test_indices]

audio_train = []
for i in tqdm(train_indices):
    audio = dataset['audio'][i].numpy()
    audio_train.append(audio[:n_samples])

audio_test = []
for i in tqdm(test_indices):
    audio = dataset['audio'][i].numpy()
    audio_test.append(audio[:n_samples])
```

We plot the first wav file as example, saved in `audio_train[0]`

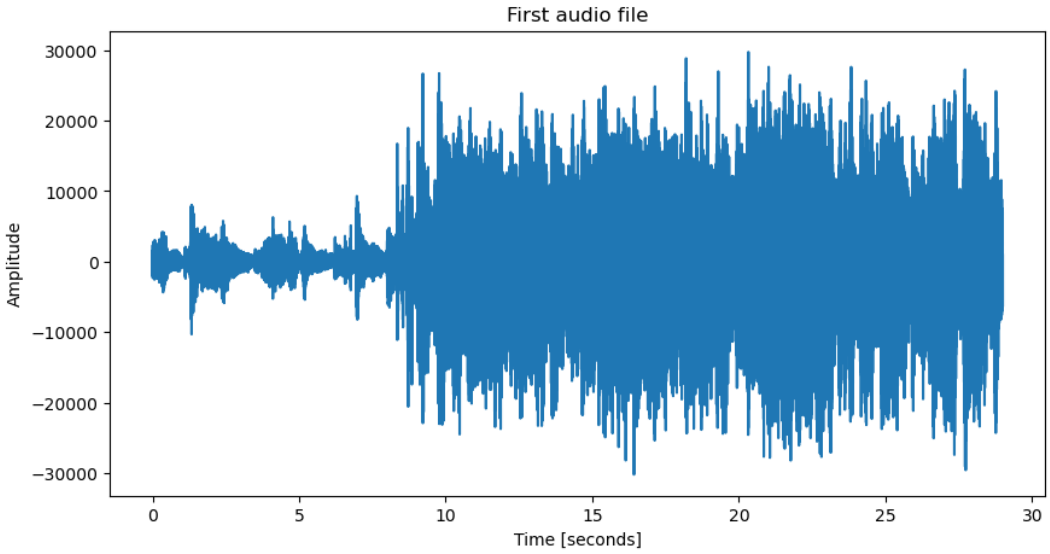


Figure 1: First wav file, audio_train[0]

2.2 Question 2

2.2.1 Process the dataset

In order to deal with different dynamic range of different music tracks we apply normalization of the data. We use the `sklearn` library, with its `MinMaxScaler` function with `feature_range = (-1,1)`. We scaled every audio files. As an example, the first wav file in the previus image became this:

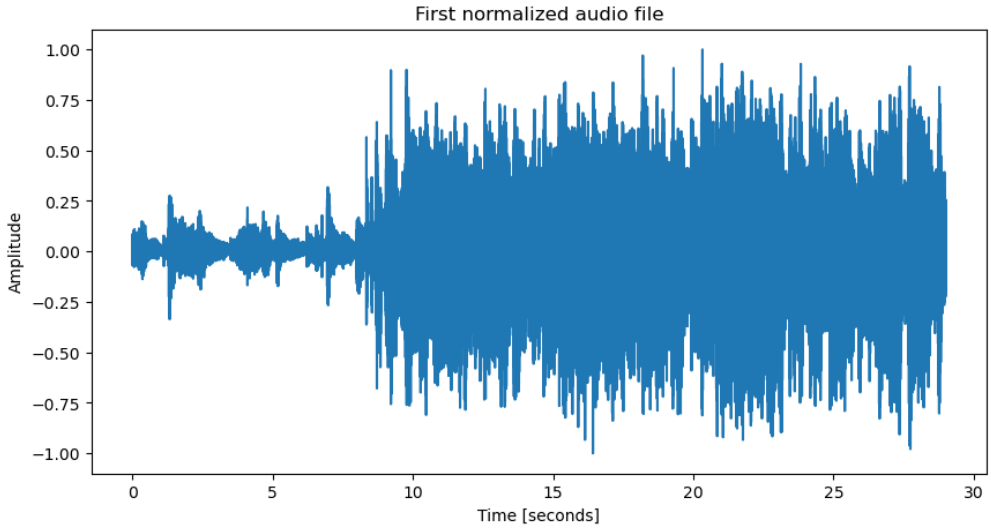


Figure 2: First audio file after normalization, audio_train[0]

2.2.2 Compute local averages and principal argument functions

Now we can define `compute_local_average` which implement this function, where n is the sample index and M is the size of the averaging window.

$$\mu(n) := \frac{1}{2M+1} \sum_{m=-M}^{M} x(n+m)$$

```
    def compute_local_average(x, M, Fs):
        local_average = np.zeros(len(x))
        for i in range(len(x)):
            start = max(0, i - M)
            end = min(len(x), i + M + 1)
            local_average[i] = np.mean(x[start:end])
            return local_average
```

In order to deal with the periodicity of the phase, we define the `principal_argument` function which maps phase differences into the range `[-0.5, 0.5]`.

```
def principal_argument(x):
    y = (x + 0.5) % 1 - 0.5
    return y
```

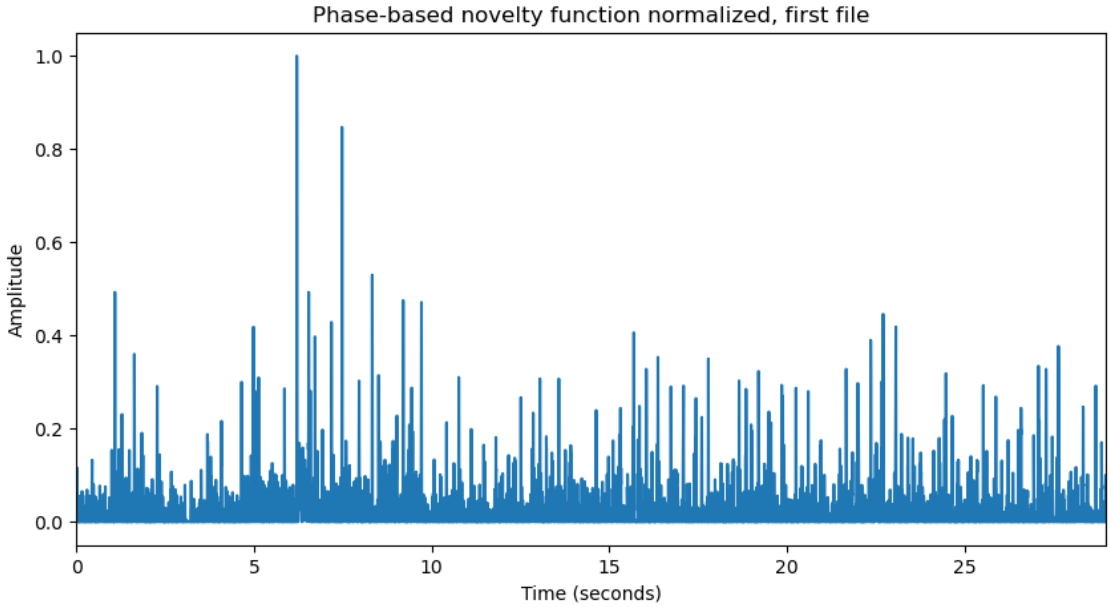
2.2.3 Compute phase novelty

We define the `compute_phase_novelty` function to extract the phase informations from the audio signals.

First of all we compute the STFT of the signal, extract the phase out and normalize it by 2π.

Then we consider the phase differences between two consecutive frames and map them into the range [-0.5,0.5] applying the `principal_argument` function as defined.

In the end we perform accumulation by summing over the frequency axis, subtract the local average using the `compute_local_average` function, and apply half-wave rectification.



We can recognize the peaks in the plot but the function appears noisy. This is because the initial signal is complex, and the rhythmic features are not very clear.

The aim of the novelty function is to capture the degree of novelty or salience at different time points, to highlight points in time where there are significant changes or events in the audio, such as the onset of musical notes, beats, or in this case rythmic features and changes.

2.3 Question 3

The aim of this part is to define the `compute_feature_vector` function. A feature vector is a way to represent an object by its particular characteristics. In our case, we can consider an audio file as its rythmic features like phase-based novelty function with its standard deviation and the mean, the tempogram, the zero crossing with its standard deviation and the mean, the spectral flux with its standard deviation and the mean and the tempo, computed as below:

```
    nov, Fs_nov = compute_phase_novelty(y=x, Fs=Fs, norm=1, plot=0)
    novelty_std = np.std(nov)
    novelty_mean = np.mean(nov)

    tempogram = librosa.feature.tempogram(y=nov, sr=Fs)
    zero_crossings = librosa.feature.zero_crossing_rate(x)
    zero_crossings_std = np.std(zero_crossings)
    zero_crossings_mean = np.mean(zero_crossings)

    spectral_flux = np.diff(librosa.onset.onset_strength(y=x, sr=Fs))
    spectral_flux_std = np.std(spectral_flux)
    spectral_flux_mean = np.mean(spectral_flux)
    tempo = librosa.feature.rhythm.tempo(onset_envelope=spectral_flux,
            y=x, sr=Fs, hop_length=H, tg=tempogram)[0]
```

In particular, zero crossing represents the rate at which the signal cross the x axis so the rate at which the signal changes its sign over time. It is a measure of the rapidity of variations in the waveform and can provide information about the frequency and periodicity of the signal. Spectral flux quantifies the changes in the energy distribution of a signal over time, providing information about the dynamics and variations in the spectral content. The spectral flux quantifies how much the spectral content of the signal changes from one instant to the next, so we need to apply difference (like derivative) of the onset computed by the difference of `onset_strength` function. Considering different rhythmic features is important to analyze all the similar aspects that songs belonging to the same genre might have in common.

We obtain the feature vector by concatenating them. Finally we compute feature vector for all the audio files inside the training set and the same for test set as shown below.

```
    train_fvector = []
    for i in tqdm(range(100)):
```

```
        f_v = compute_feature_vector(audio_train[i], Fs, N, H)
        train_fvector.append(f_v)
    train_fvector = np.array(train_fvector)

    test_fvector = []
    for i in tqdm(range(20)):
        f_v = compute_feature_vector(audio_test[i], Fs, N, H)
        test_fvector.append(f_v)
    test_fvector = np.array(train_fvector)
```

2.4 Question 4

Now we have to select a model for the classification training a Support Vector Machine (SVM).

2.4.1 Support Vector Machine and hyperparameters

A SVM is a supervised machine learning algorithm used for classification. Its primary goal is to find a hyperplane that best separates data points into different classes while maximizing the margin. Hyperparameters are external configuration settings for a machine learning model that are not learned from the data but are set prior to the training process. We consider two hyperparameter:
1. Kernel: kernel function determines how the input features are transformed into a higher-dimensional space, where the SVM can find a hyperplane that separates the data into different classes. Default kernel is `'linear'`, other common kernels are polynomial, radial basis function, sigmoid.
2. C: We can also work with the C parameter. A higher value of C allows the model to correctly classify a greater number of points in the training set, but could lead to over fitting, especially if the problem is noisy or not perfectly linearly separable. A lower value of C makes the model more tolerant to errors in the training set, promoting greater generalization.

2.4.2 Model

We can modify parameters like C and kernel, default values are `C=1` and `kernel = 'linear'`. As shown below we create the model, train it with our feature vectors and save it.

```
    if not os.path.exists('my_model/'):
        os.mkdir('my_model/')
    model = sklearn.svm.SVC(C=C, kernel=kernel)
```

```
    model.fit(train_fvector, genre_train)
    file = f'my_model/svc_{kernel}_C_{C}_N_{N}_H_{H}.joblib'
    joblib.dump(model, file)
```

Now we check the accuracy of the training set.

```
    train_accuracy = accuracy_score(genre_train,
    model.predict(train_fvector))
    print(f"Accuracy on the training set: {train_accuracy * 100:.2f}%")
```

For `C=1` and `kernel='linear'` the accuracy is 100%. It means that the model is overfitting. Overfit phenomena accours when the model corresponds too closely to a particular set of data (the training set), and find hard to work with new data. In order to deal with this problem, we have to work with the values of the parameters.

2.5 Question 5

Now that we created and trained the model, we are able to perform classification.

2.5.1 Classification

We predict the classification of the test set, and we compare the predictions with the `genre_test`, the known solution of the classification into genres. We print the accuracy.

```
    predictions = model.predict(test_fvector)
    test_accuracy = accuracy_score(genre_test, predictions)
    print(f"Accuracy on the training set: {test_accuracy * 100:.2f}%")
```

After some tries we found the best accuracy for the test set is with the combination of `kernel='poly'` and `C=0.1`. The best accuracy for the test set is 50% and the accuracy of the training set became 41%.

2.5.2 Confusion matrix

Confusion matrix is a table used to evaluate the performance of a classification algorithm on a set of data for which the true values are known. We are classifying audio track into 10 genres, so the confusion matrix is a 10x10 matrix which represents, in the two axis, the known values and the predicted values. The classificator is as good as more values are on the diagonal.

```
    cmatrix = confusion\_matrix(genre\_test, predictions)
    plt.figure(figsize=(8, 6))
```

```
    sns.heatmap(cmatrix, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=genre_names, yticklabels=genre_names)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()
```

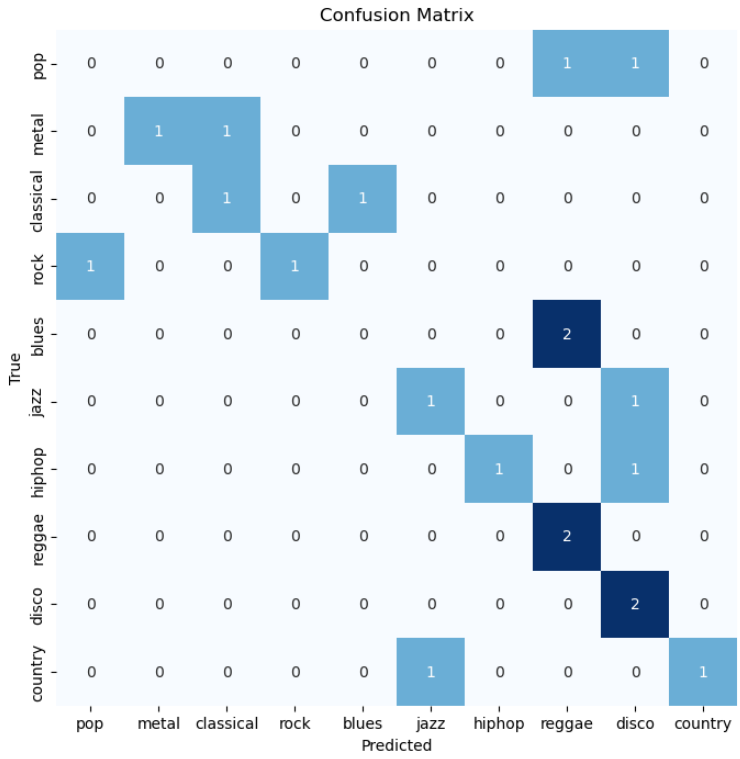


Figure 3: Confusion matrix for the model

3 Conclusion

The final accuracy is obtained is 50%. It is not the accuracy of a perfect classificator, due to the very simple model applied and the low number of audio tracks. However, we can notice that the accuracy obained, is higher than 10% which is the accuracy of a random classifier. A simple way to increase accuracy without changing the process is to increase the number of initial wav files.