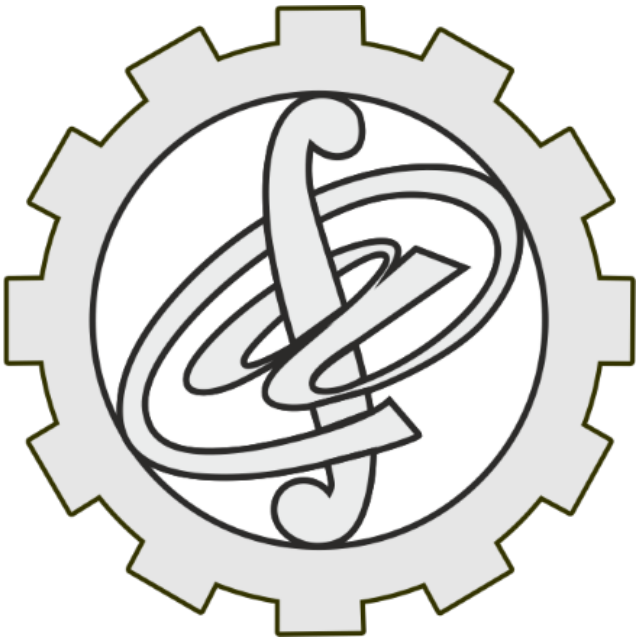
Systemy sztucznej inteligencji

dokumentacja projektu "Klasyfikacja gatunków muzyki"

Patryk Gamrat, Radosław Olesiński, Radosław Szwed, grupa 2/4
Politechnika Śląska, Wydział Matematyki Stosowanej

5 czerwca 2024



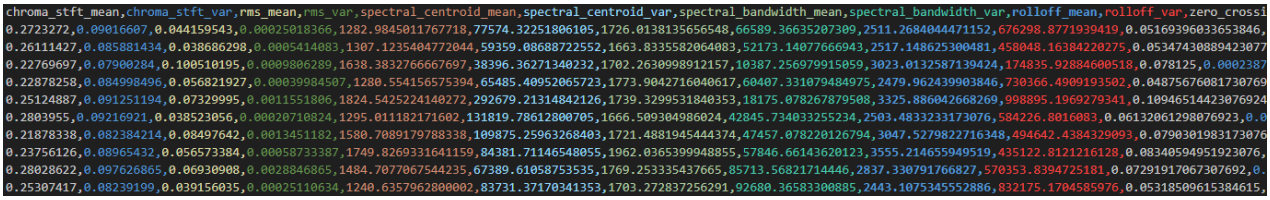
Część I

Opis projekt

Głównym założeniem projektu było utworzenie systemu, który klasyfikuje gatunki muzyki. Aby zrealizować te zadanie, należy odczytać pliki dźwiękowe i wyodrębnić z nich cechy, na podstawie których można klasyfikować utwory muzyczne. W projekcie wykorzystano zbiór danych **GTZAN Music Genre Classification**, który zawiera 1000 utworów muzycznych, podzielonych na 10 gatunków. Klasyfikacja jest realizowana za pomocą algorytmu K najbliższych sąsiadów.

Instrukcja obsługi

Pierwszym krokiem, jaki należy wykonać, jest wyodrębnienie cech dźwięku z plików dźwiękowych znajdujących się w bazie (możemy wykorzystać zbiór GTZAN lub własne pliki dźwiękowe). Aby przeanalizować utwory, należy uruchomić plik **Analyze_Audio.ipynb**. Skrypt przetwarza pliki dźwiękowe i zapisuje wyniki do plików csv. Pliki wejściowe i wyjściowe są podzielone w katalogach według gatunków.
Po wyodrębnieniu cech możemy uruchomić plik **Classify_Music.ipynb**, który klasyfikuje utwory muzyczne na podstawie utworzonych plików csv.



Rysunek 1: Plik csv z wyodrębnionymi cechami dźwięku

Dodatkowe informacje

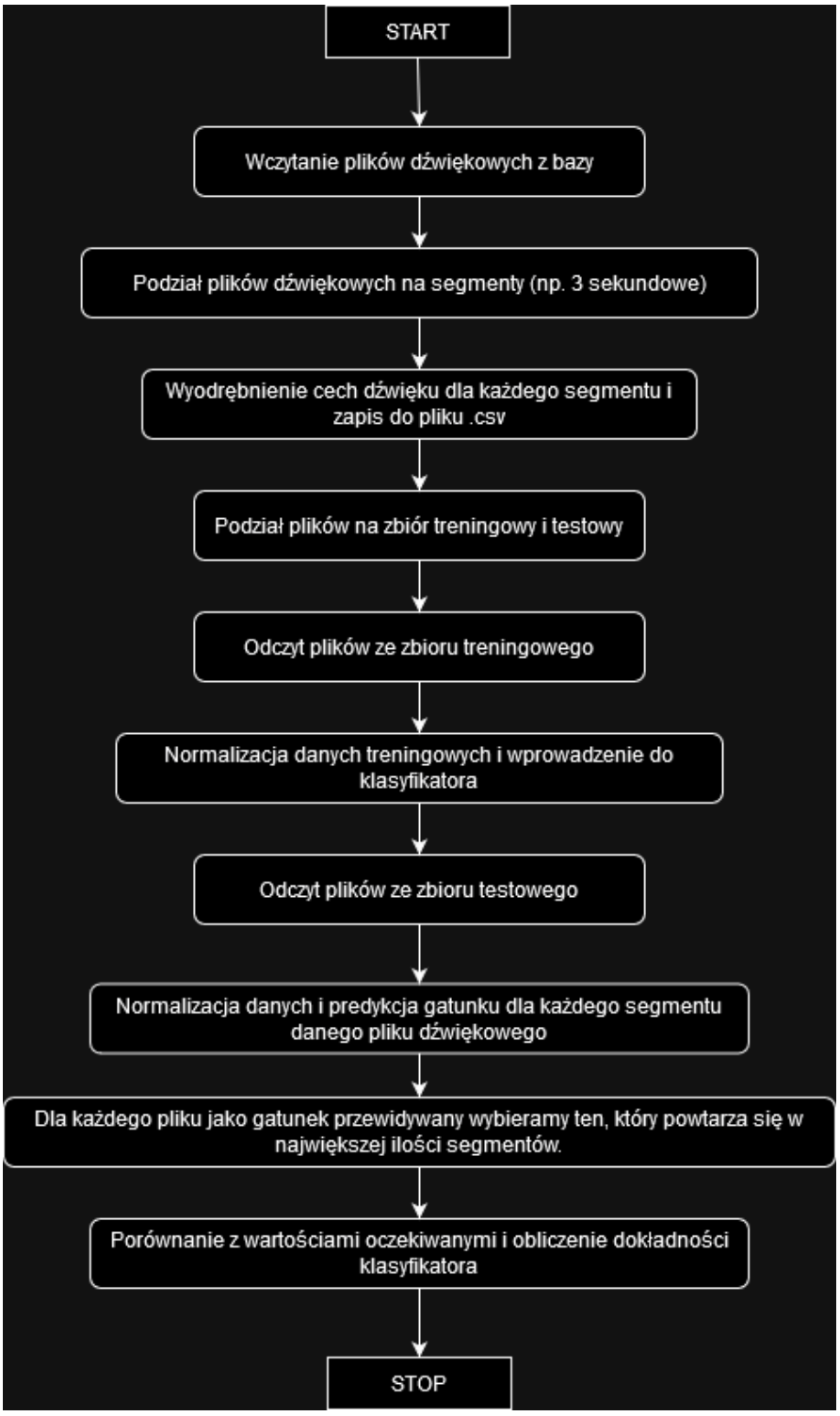
Do poprawnego działania wymagany jest Python 3.11 oraz następujące biblioteki:

- Pandas
- Numpy
- Matplotlib
- Seaborn
- Scikit-learn
- Librosa

Część II

Opis działania

Poniżej znajduje się schemat, opisujący sposób działnia systemu.



Rysunek 2: Schemat działania systemu klasyfikacji muzyki

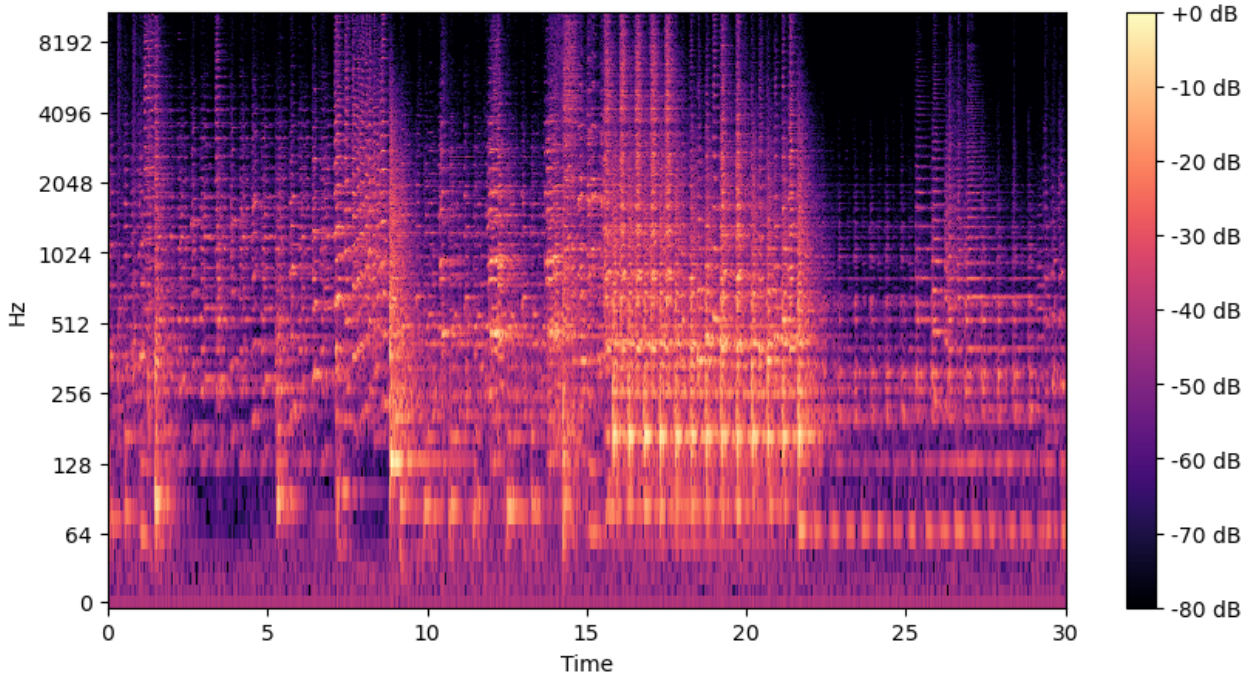
Analiza dźwięku

Każdy plik z bazy z muzyką jest poddawany analizie, na podstawie której wyodrębniane są cechy dźwiękowe. Dla każdej cechy (z wyjątkiem tempa) obliczana jest wartość średnia oraz wariancja, z wartości obliczonych na przestrzeni całego segmentu. Dla każdego pliku dźwiękowego cechy zapisywane są do pliku .csv, dzięki czemu można je później wykorzystać do przewidywania gatunku muzycznego utworu.

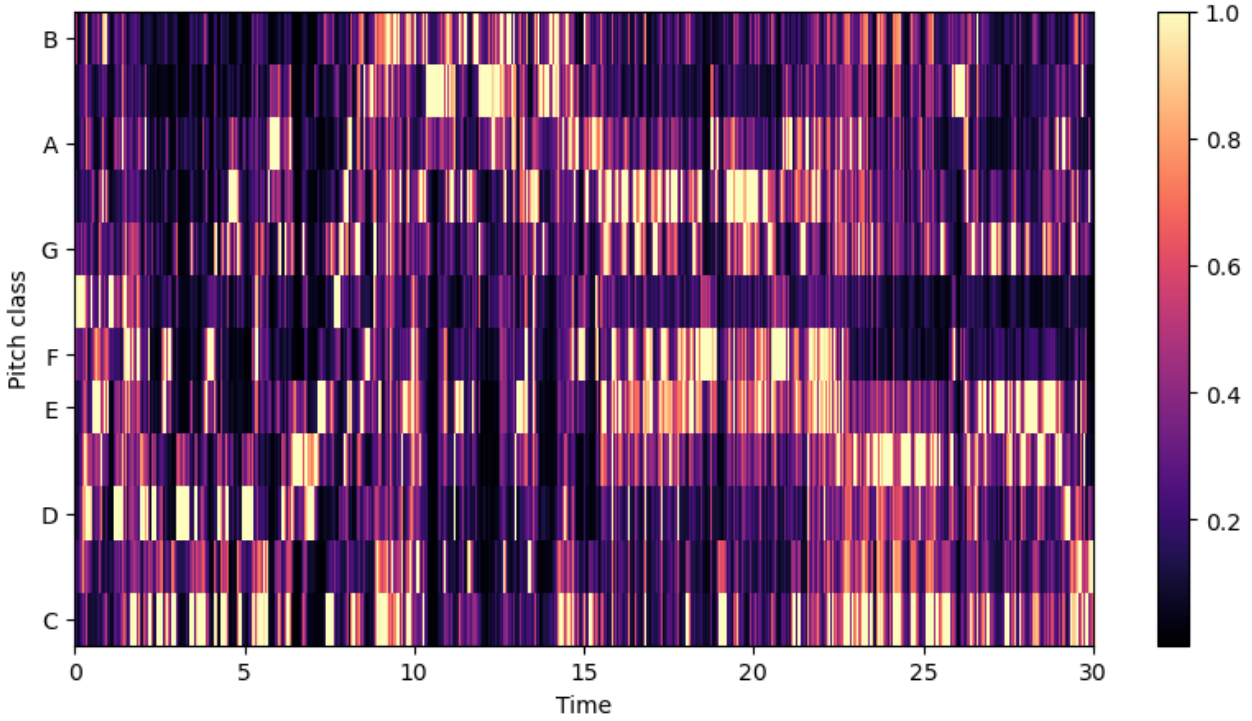
Aby zwiększyć ilość danych treningowych oraz dokładność modelu, pliki dźwiękowe są poddawane podziałowi na krótsze segmenty, dla których z osobna obliczane są cechy. Każdy segment dźwiękowy charakteryzowany jest przez 57 różnych wartości liczbowych.

Chromogram

Chromogram to wektorowa reprezentacja częstotliwości występujących w próbce dźwiękowej. Uzyskiwany jest poprzez przetworzenie sygnału audio za pomocą Krótkoczasowej transformacji Fouriera (STFT), a następnie przypisanie występujących częstotliwości do jednego z dwunastu półtonów występujących w oktawie.



Rysunek 3: Spektogram dla pliku jazz.00000.wav



Rysunek 4: Chromogram dla pliku jazz.00000.wav

Średnia kwadratowa energii sygnału

Średnia kwadratowa energii sygnału jest wskaźnikiem dynamiki dźwięku, pozwala na określenie głośności oraz zmian w intensywności dźwięku. W tym przypadku liczymy średnią kwadratową amplitud sygnału.

$$\sqrt{\frac{1}{N}\sum_{n=0}^{N} x[n]^2}$$

gdzie $x[n]$ to próbki sygnału, a N to liczba próbek sygnału.

Centroid spektralny

Centroid spektralny to kolejna cecha wyznaczana na podstawie STFT. Centroid spektralny jest w znacznym stopniu powiązany z barwą dźwięku, czyli subiektywną cechą dzięki której możemy odróżnić brzmienie różnych instrumentów albo charakteryzować ludzki głos. Im wyższa wartość centroida, tym "jaśniejsze" jest brzmienie.
Wartość tą obliczamy jako średnią ważoną częstotliwości sygnału, gdzie wagi to amplitudy składowych częstotliwości.

$$\text{Centroid spektralny} = \frac{\sum_{k=0}^{N} f_k |X(k)|}{\sum_{k=0}^{N} |X(k)|}$$

gdzie f_k to częstotliwości, a $X(k)$ to amplitudy tych częstotliwości

Szerokość pasma
Szerokość pasma to różnica pomiędzy wysokimi a niskimi częstotliwościami sygnału. Szerokie pasmo wskazuje na złożone, bogate widmo, natomiast wąskie pasmo oznacza prostszy, bardziej tonalny dźwięk. Znając spektralny centroid, pasmo przenoszenia w danym czasie możemy obliczyć jako odchylenie standardowe wokół tej wartości.

$$\text{Szerokość pasma} = \sqrt{\frac{\sum_{k=0}^{N}(f_k - \text{Centroid})^2|X(k)|}{\sum_{k=0}^{N}|X(k)|}}$$

gdzie f_k to częstotliwości, a $X(k)$ to amplitudy tych częstotliwości

Spadek spektralny
Spadek spektralny (spectral rolloff) to punkt częstotliwościowy, poniżej którego znajduje się określony procent (zwykle 85%) całkowitej energii widma.

$$S_r = \sum_{k=0}^{R}|X(k)| = 0.85\sum_{k=0}^{N}|X(k)|$$

Częstotliwość przejść przez zero
Częstotliwość przejść przez zero (zero crossing rate) mierzy częstotliwość, z jaką sygnał przechodzi przez oś poziomą (wartość zero). Wysoka wartość ZCR jest charakterystyczna dla dźwięków perkusyjnych.
Cecha obliczana jest jako liczba przejść przez zero w danym oknie czasowym, dzielona przez długość okna.

$$\text{ZCR} = \frac{1}{N}\sum_{n=0}^{N}\mathbf{1}_{\{x[n]\cdot x[n-1]<0\}}$$

gdzie $x[n]$ to sygnał, N to długość okna, a $\mathbf{1}$ to funkcja wskaźnikowa.

Składowe harmoniczne i perkusyjne
Na odbierany przez nas dźwięk składa się wiele elementów, w szczególności istotne są dla nas składowe harmoniczne (czyli np. dźwięki fortepianu, gitary albo śpiew) oraz składowe perkusyjne (czyli np. perkusja i inne instrumenty niemelodyjne). Składowe te można odseparować za pomocą algorytmu HPS wbudowanego w bibliotekę librosa.

Tempo
Tempo określa liczbę uderzeń na minutę (BPM) w utworze muzycznym, czyli inaczej mówiąc szybkość wykonywania utworu. Tempo może być obliczane poprzez wykrywanie szczytów w sygnale czasowym i mierzenie odstępów czasowych między nimi.

Współczynniki Mel-cepstralne
Współczynniki mel-cepstralne (mfcc) powstają z cepstrum sygnału przedstawionego w skali melowej. Współczynniki te reprezentują charakterystykę widma dźwięku, dzięki czemu znajdują szerokie zastosowanie w wielu dziedzinach uczenia maszynowego (np. rozpoznawanie mowy). Skala melowa została określona w 1937 roku, definiuje ona subiektywne odczucie dźwięku przez ludzkie ucho. Proces obliczania współczynników jest dosyć złożony, pierwszym

krokiem jest przekształcenie sygnału za pomocą STFT. Następnie spektrum jest filtrowane z zastosowaniem banku filtrów skali melowej, a wyniki są logarytmowane. Ostatnim krokiem jest zastosowanie dyskretnej transformacji kosinusowej i obliczenie żądanej ilości współczynników. Nie podajemy tutaj konkretnych wzorów, ponieważ dokładna implementacja powyższego algorytmu różni się w zależności od zastosowań.

Ekstrakcja cech dźwięku

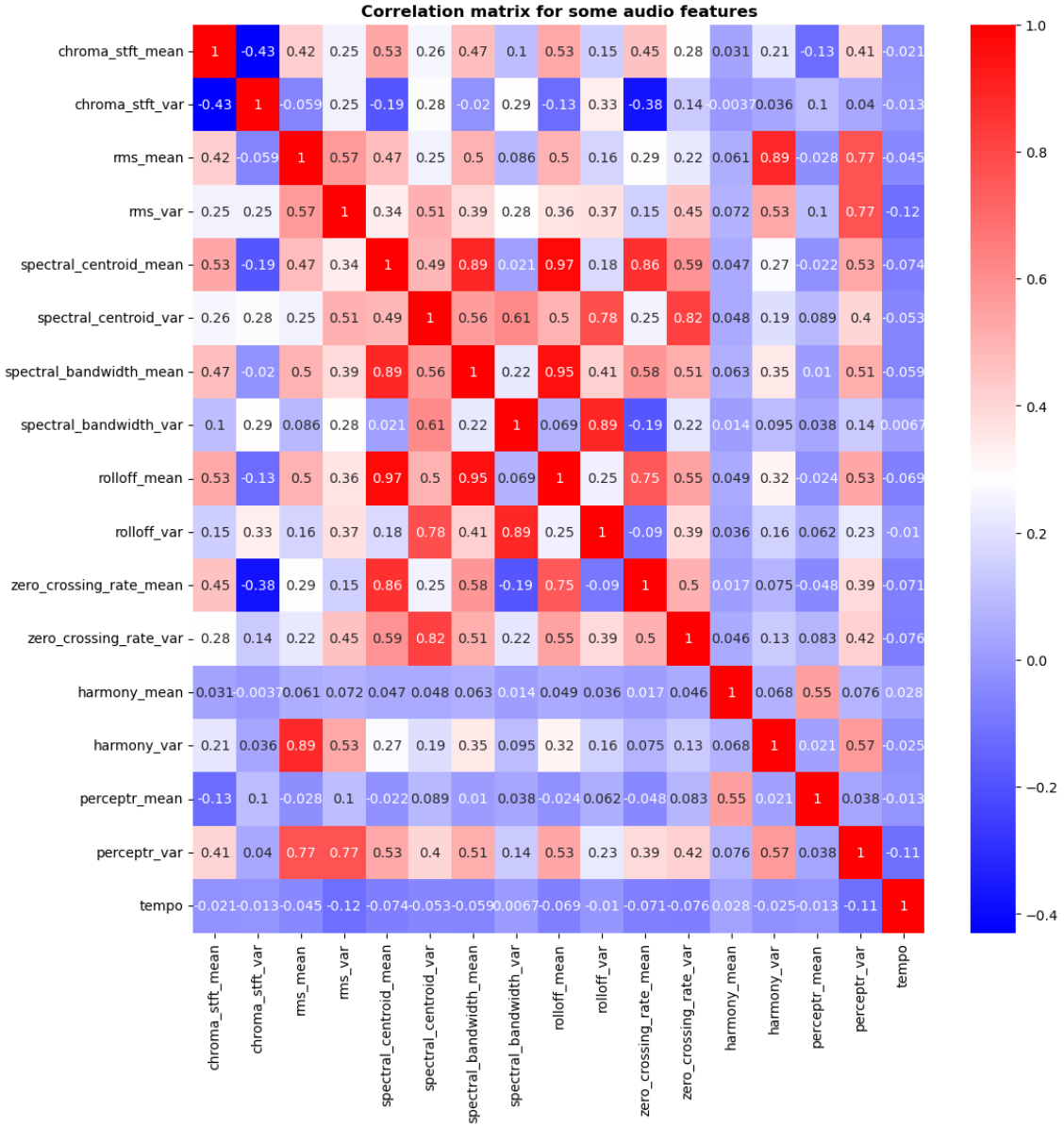
```
for genre in audio_files:
    i = 0
    for f in os.listdir(audio_files[genre]):
        csv = copy.deepcopy(csv_template)
        f = audio_files[genre] + "/" + f
        print("Processing:", f)

        audio, sample_rate = librosa.load(f)
        audio_duration = librosa.get_duration(y=audio, sr=sample_rate)
        num_segment = int(audio_duration/SEGMENT_DURATION)
        samples_per_segment = int(sample_rate*audio_duration/num_segment)

        for n in range(num_segment):
            audio_seg = audio[samples_per_segment*n: samples_per_segment*(n
    +1)]

            # Chromagram
            chromagram = librosa.feature.chroma_stft(y=audio_seg, sr=
    sample_rate)
            csv["chroma_stft_mean"].append(chromagram.mean())
            csv["chroma_stft_var"].append(chromagram.var())

            # Root Mean Square Energy
            RMSEn= librosa.feature.rms(y=audio_seg)
            csv["rms_mean"].append(RMSEn.mean())
            csv["rms_var"].append(RMSEn.var())

            # Spectral Centroid
            spec_cent=librosa.feature.spectral_centroid(y=audio_seg)
            csv["spectral_centroid_mean"].append(spec_cent.mean())
            csv["spectral_centroid_var"].append(spec_cent.var())

            #Spectral Bandwith
            spec_band=librosa.feature.spectral_bandwidth(y=audio_seg,sr=
    sample_rate)
            csv["spectral_bandwidth_mean"].append(spec_band.mean())
            csv["spectral_bandwidth_var"].append(spec_band.var())

            # Spectral Rolloff
            spec_roll=librosa.feature.spectral_rolloff(y=audio_seg,sr=
    sample_rate)
            csv["rolloff_mean"].append(spec_roll.mean())
            csv["rolloff_var"].append(spec_roll.var())

            # Zero Crossing Rate
            zero_crossing=librosa.feature.zero_crossing_rate(y=audio_seg)
```

```
            csv["zero_crossing_rate_mean"].append(zero_crossing.mean())
            csv["zero_crossing_rate_var"].append(zero_crossing.var())

            # Harmonics and Perceptrual
            harmony, perceptr = librosa.effects.hpss(y=audio_seg)
            csv["harmony_mean"].append(harmony.mean())
            csv["harmony_var"].append(harmony.var())
            csv["perceptr_mean"].append(perceptr.mean())
            csv["perceptr_var"].append(perceptr.var())

            # Tempo
            tempo = librosa.feature.tempo(y = audio_seg, sr = sample_rate)
            csv["tempo"].append(tempo.item())

            # Mfcc
            mfcc=librosa.feature.mfcc(y=audio_seg,sr=sample_rate)
            mfcc=mfcc.T
            for x in range(20):
                feat1 = "mfcc" + str(x+1) + "_mean"
                feat2 = "mfcc" + str(x+1) + "_var"
                csv[feat1].append(mfcc[:,x].mean())
                csv[feat2].append(mfcc[:,x].var())

        df = pd.DataFrame(csv)
        df.to_csv(f"{SAVE_LOCATION}/{genre}/{genre}{i}.csv", index=False)
        i += 1
```

Analiza danych

Macierz korelacji

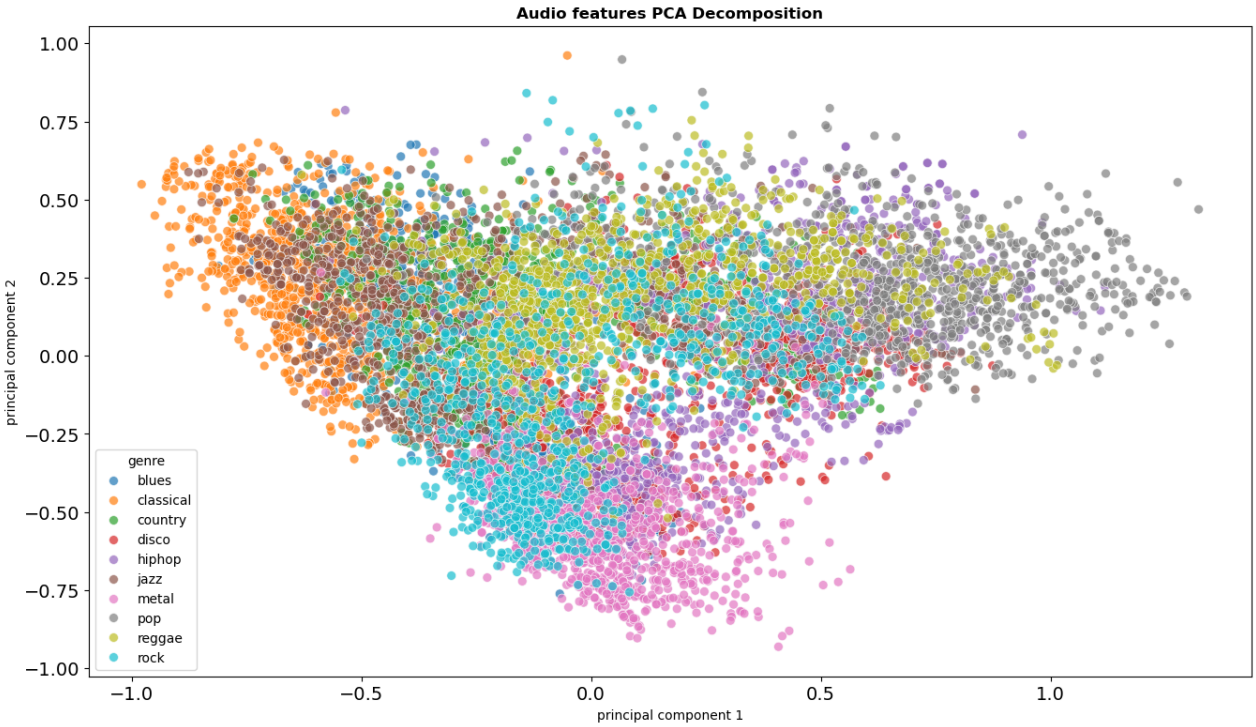
Poniżej przedstawiono macierz korelacji dla niektórych cech dźwięku, wyraźnie widać, że większość z nich jest od siebie w mniejszym lub większym stopniu zależna. W praktyce zachowanie wszystkich cech przynosiło jednak najlepsze rezultaty.



Rysunek 5: Macierz korelacji danych

Analiza głównych składowych

Dzięki analizie głównych składowych (PCA), możemy zwizualizować poszczególne gatunki muzyczne oraz zagęszczenie danych na wykresie. Warto jednak zaznaczyć, że dla dwóch komponetów przedstawione jest zaledwie około 45% całkowitej wariancji, dlatego poniższy wykres należy traktować jedynie jako ciekawostke.



Rysunek 6: Analiza głównych składowych dla gatunków muzycznych

Algorytm

Pierwszy algorytm to implementacja metody k najbliższych sąsiadów (KNN). Algorytm ten jest używany do klasyfikacji gatunku muzycznego danego segmentu. Dla danego punktu, który ma zostać sklasyfikowany, algorytm KNN szuka k najbliższych sąsiadów w zbiorze treningowym. Następnie, na podstawie tych sąsiadów, przewiduje klasę dla danego punktu.
W algorytmie do obliczania odległości zastosowano klasyczną metrykę euklidesową:

$$d(v,u) = \sqrt{\sum_{i=1}^{n} |v_i - u_i|^2}$$

Do normalizacji danych wykorzystano normalizacje względem wielkości skrajnych (Min-Max) w zakresie od 0 do 1:

$$x' = \frac{x - min}{max - min}$$

Algorithm 1 KNN Algorithm

Input : k - number of neighbors, X_train, y_train - training data, $X\_predict$ - test data to predict
Output: Predictions for test data

```
class KNN()
    def __init__(self, k=3)
        // Class initialization
        self.k ← k
    def fit(self, X_train, y_train)
        // Model training
        self.X_train ← X_train  self.y_train ← y_train
    def predict(self, X_test)
        // Prediction
        foreach X in X_test do
            // Distance calculation
            dst ← self.X_train - X  dst ← dst * dst  dst ← dst.sum(axis='columns') ** 0.5
            // DataFrame creation
            dst ← pd.DataFrame({'distance': dst, 'value': self.y_train})
            dst ← dst.sort_values('distance')
            // Choosing nearest neighbors
            voters ← dst.head(self.k).value
            // Voting
            predictions.append(voters.mode()[0])
        end
        return predictions
```

Drugi algorytm to metoda predykcji gatunków muzycznych. Dla plików testowych zawierających cechy dźwięku dla każdego segmentu algorytm wczytuje dane, normalizuje je i przewiduje gatunek muzyczny za pomocą klasyfikatora. Następnie, najczęściej przewidywany gatunek spośród wszystkich segmentów jest wybierany jako wynik dla danego pliku. Na końcu, algorytm oblicza dokładność predykcji, porównując przewidziane gatunki z prawdziwymi gatunkami.

Algorithm 2 Music Genre Prediction Algorithm

Input : List of test files $test\_files$
Output: Accuracy of music genre predictions

foreach f ***in*** $test\_files$ **do**
    $X\_test \leftarrow$ read data from file f
    $X\_test \leftarrow$ scale data using $scaler$
    $predictions \leftarrow clf.predict(X\_test)$
    // Finding the most frequent genre
    $possible\_genres, counts \leftarrow np.unique(predictions, return\_counts = True)$
    $genre \leftarrow possible\_genres[counts.argmax()]$
    $y\_pred.append(genre)$
end
// Calculating accuracy

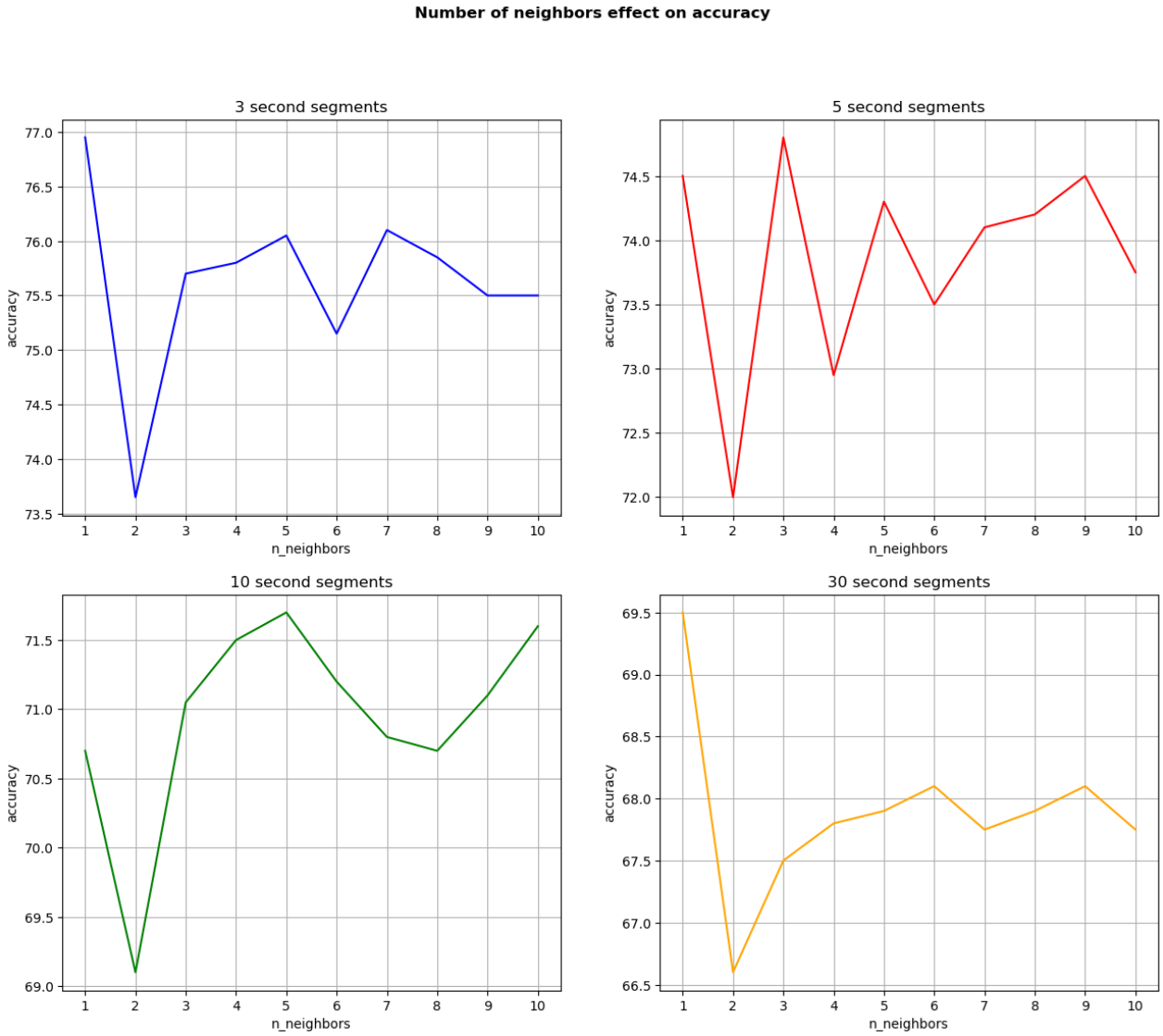
Implementacja

```
X_train_data = []
for i, f in enumerate(train_files):
    csv_df = pd.read_csv(f, header = 0)
    csv_df["genre"] = train_files_genre[i]
    X_train_data.append(csv_df)

X_train = pd.concat(X_train_data, axis = 0, ignore_index = True)
y_train = X_train.pop("genre")
scaler = MinMaxScaler()
scaler.set_output(transform="pandas")
X_train = scaler.fit_transform(X_train)
clf = KNN(1)
clf.fit(X_train, y_train)

y_test = test_files_genre
y_pred = []
for f in test_files:
    X_test = pd.read_csv(f)
    X_test = scaler.transform(X_test)
    predictions = clf.predict(X_test)
    possible_genres, counts = np.unique(predictions, return_counts=True)
    genre = possible_genres[counts.argmax()]
    y_pred.append(genre)

print(f"Dokładność: {accuracy_score(y_test, y_pred)*100}%")
```

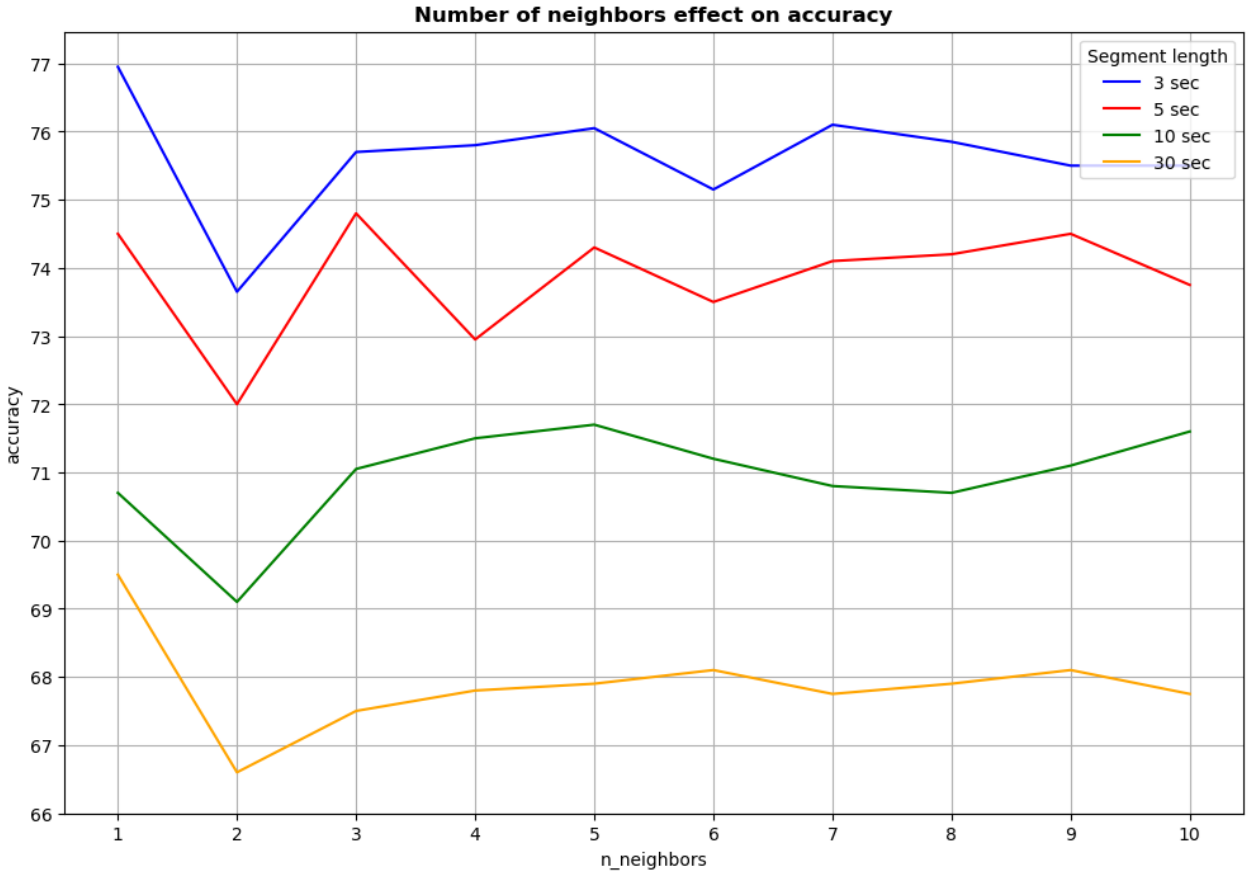
Eksperymenty

KNN

Wykresy przedstawiają wyniki, jakie algorytm KNN uzyskał dla liczby sąsiadów od 1 do 10, dla segmentów 3, 5, 10 i 30 sekundowych. Maksymalnie najwyższą dokładność uzyskujemy dla trzy sekundowych segmentów z jednym sąsiadem. Nasze klasy danych są bardzo różnorodne i mają duże rozproszenie, więc większa liczba sąsiadów może uwzględniać punkty z różnych klas, nawet jeśli punkt testowy jest blisko centrum swojej klasy. Możemy również zauważyć, że długość analizowanych segmentów, na które dzielimy nasz utwór muzyczny, jest odwrotnie proporcjonalna do uzyskiwanej dokładności. Jest to spowodowane tym, że krótsza długość segmentów zapewnia lepszą reprezentację zmienności w czasie, oraz większą dokładność ekstrakcji cech sprawdzanych utworów.



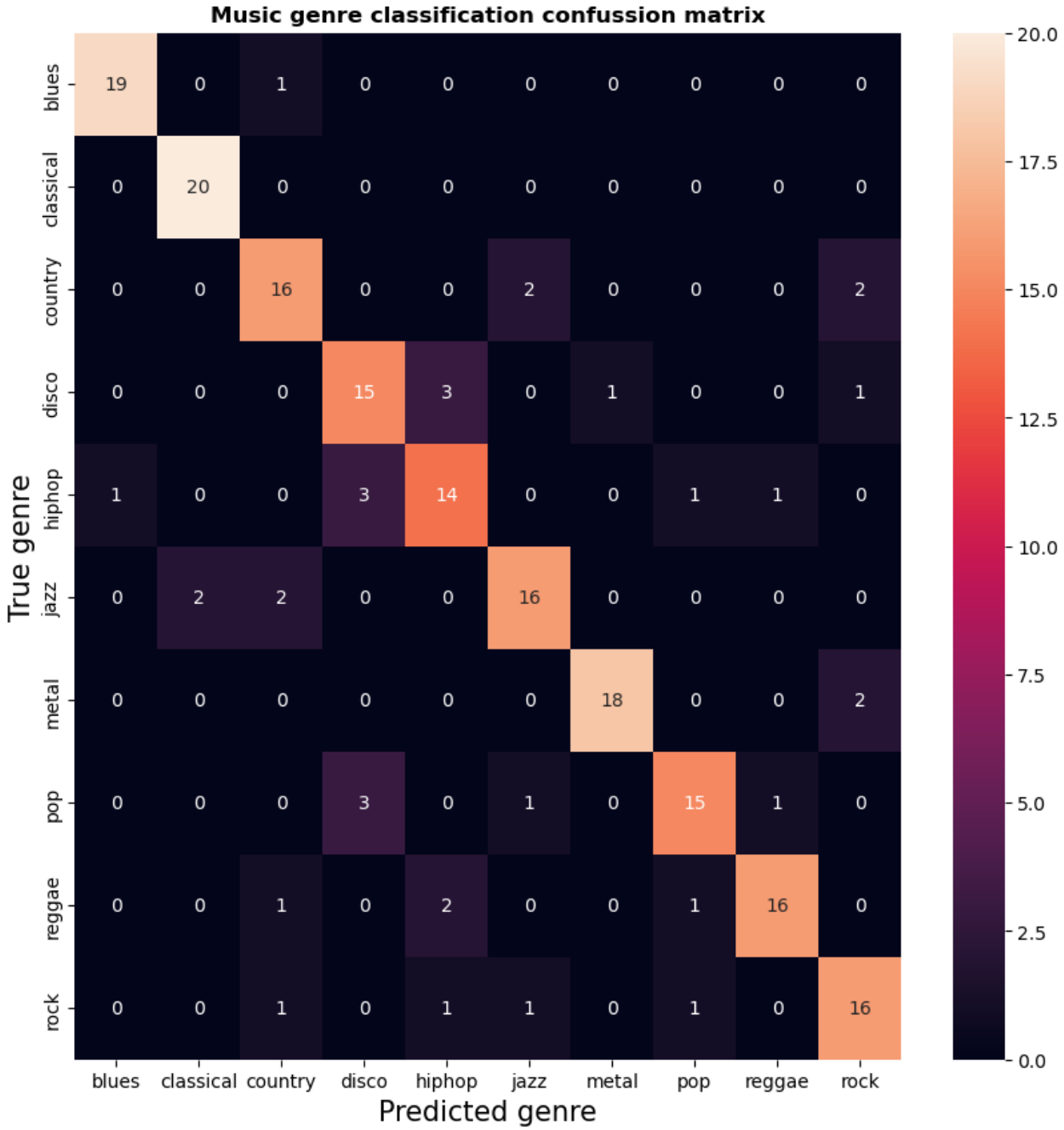
Rysunek 7: Analiza liczby sąsiadów i długości segmentów



Rysunek 8: Analiza liczby sąsiadów i długości segmentów

Analiza wyników

Dla podziału plików dźwiękowych na 3 sekundowe segmenty i jednego sąsiada, klasyfikator KNN uzyskiwał średnią dokładność na poziomie **77%**.
Możemy zauważyć, że algorytm najlepiej radzi sobie z klasyfikacją muzyki klasycznej oraz bluesa. Najprawdopodobniej wynika to ze specyficznej charakterystyki tych gatunków, które ciężko pomylić z czymś innym. Trzeba jednak zauważyć, że pliki dźwiękowe znajdujące się w bazie GTZAN zostały precyzyjnie wyselekcjonowane, i reprezentują najbardziej typowe brzmienie dla każdego gatunku muzyki. W ogólnym przypadku współczesna muzyka może zawierać cechy charakterystyczne dla kilku gatunków, dlatego w przypadku klasyfikacji utworów spoza bazy, możemy spodziewać się innych wyników.



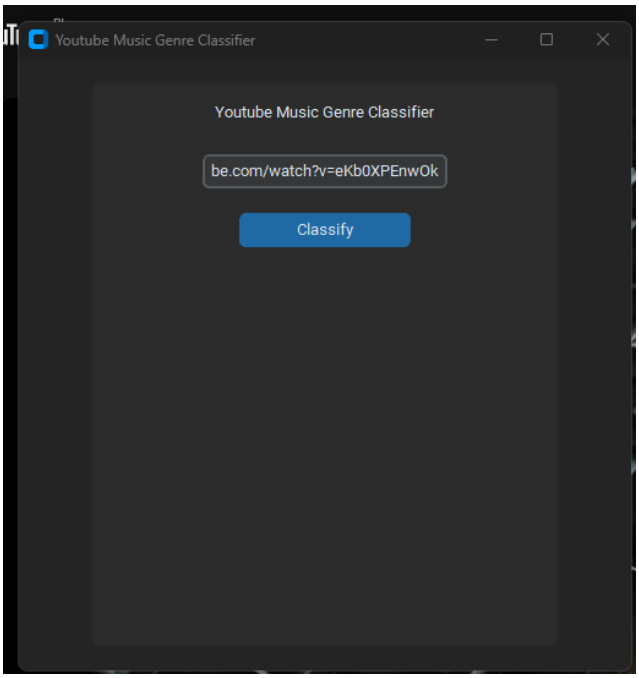
Rysunek 9: Macierz błędów

Testy

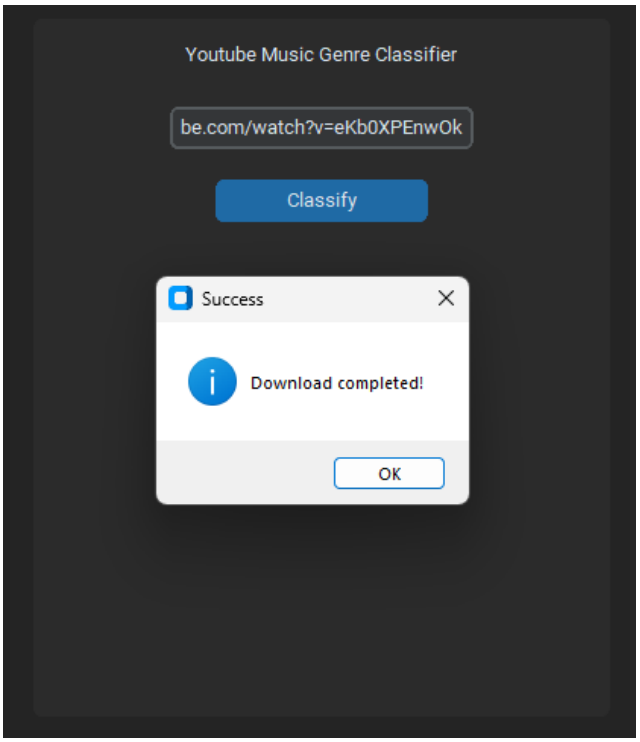
Analiza muzyki spoza bazy

Do analizy możemy również używać utworów muzycznych nie znajdujących się w używanej przez nas bazie danych GTZAN. Na potrzeby testów utworzony został prosty program napisany w języku python z użyciem biblioteki customtkinter i pytube. Aplikacja działa na zasadzie podania linku do utworu z platformy YouTube. Następnie program pobiera ten utwór i wykonuje analizę w oparciu o wcześniej opisane metody. Na ekran wypisywane są wyniki analizy, a pobrany utwór zostaje usunięty.

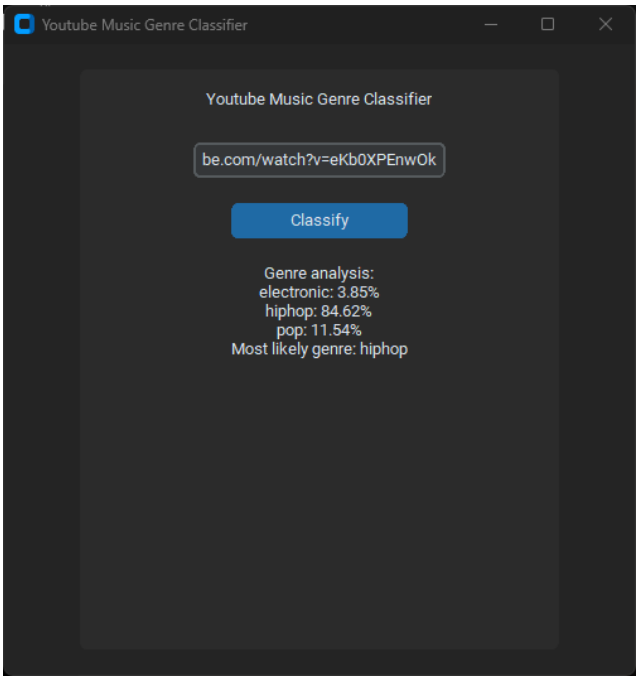
Z powodów wymienionych wcześniej, analiza utworów z bazą danych GTZAN jako zbiór treningowy nie przyniosła najlepszych rezultatów. W związku z tym w ramach testu nasza baza z cechami dźwięku została przez nas rozszerzona o cechy wyciągnięte na podstawie utworów znajdujących się obecnie na listach przebojów. Dodatkowo zamiast przewidywania pojedynczego gatunku, w zastosowaniu praktycznym wypisujemy procentową klasyfikację, która odzwierciedla ilość segmentów utworu przypisanych do konkretnego gatunku. Aby przyspieszyć klasyfikacje, czas pojedynczego segmentu to 5 sekund. Nie przeprowadziliśmy dokładnej analizy, ale po wprowadzonych modyfikacjach w większości przypadków program poprawnie wskazywał na dominujący gatunek muzyczny.



Rysunek 10: Wygląd programu i użycie linku



Rysunek 11: Sukces pobrania utworu



Rysunek 12: Analiza gatunku utworu

Pełen kod aplikacji

Plik Analyze_audio.ipynb

```
import os
import pandas as pd
import librosa
import copy

SEGMENT_DURATION = 1 # in sec
SAVE_LOCATION = f"Audio_Features_{SEGMENT_DURATION}sec"

csv_template= {"chroma_stft_mean": [], "chroma_stft_var": [], "rms_mean": [], "rms_var": [], "spectral_centroid_mean": [],
        "spectral_centroid_var": [], "spectral_bandwidth_mean": [], "spectral_bandwidth_var": [], "rolloff_mean": [], "rolloff_var": [],
        "zero_crossing_rate_mean": [], "zero_crossing_rate_var": [], "harmony_mean": [], "harmony_var": [], "perceptr_mean": [],
        "perceptr_var": [], "tempo": [], "mfcc1_mean": [], "mfcc1_var" : [], "mfcc2_mean" : [], "mfcc2_var" : [],
        "mfcc3_mean" : [], "mfcc3_var" : [], "mfcc4_mean" : [], "mfcc4_var" : [], "mfcc5_mean" : [],
        "mfcc5_var" : [], "mfcc6_mean" : [], "mfcc6_var" : [], "mfcc7_mean" : [], "mfcc7_var" : [],
        "mfcc8_mean" : [], "mfcc8_var" : [], "mfcc9_mean" : [], "mfcc9_var" : [], "mfcc10_mean" : [],
        "mfcc10_var" : [], "mfcc11_mean" : [], "mfcc11_var" : [], "mfcc12_mean" : [], "mfcc12_var" : [],
        "mfcc13_mean" : [], "mfcc13_var" : [], "mfcc14_mean" : [], "mfcc14_var" : [], "mfcc15_mean" : [],
        "mfcc15_var" : [], "mfcc16_mean" : [], "mfcc16_var" : [], "mfcc17_mean" : [], "mfcc17_var" : [],
        "mfcc18_mean" : [], "mfcc18_var" : [], "mfcc19_mean" : [], "mfcc19_var" : [], "mfcc20_mean" : [],
        "mfcc20_var":[]}
audio_files = {
        'blues': "gtzan_dataset/blues",
        'classical': "gtzan_dataset/classical",
        'country': "gtzan_dataset/country",
        'disco': "gtzan_dataset/disco",
        'hiphop': "gtzan_dataset/hiphop",
        'jazz': "gtzan_dataset/jazz",
        'metal': "gtzan_dataset/metal",
        'pop':"gtzan_dataset/pop",
        'reggae': "gtzan_dataset/reggae",
        'rock': "gtzan_dataset/rock"
}

for genre in audio_files:
    i = 0
    for f in os.listdir(audio_files[genre]):
        csv = copy.deepcopy(csv_template)
        f = audio_files[genre] + "/" + f
        print("Processing:", f)

```

```
        audio, sample_rate = librosa.load(f)
        audio_duration = librosa.get_duration(y=audio, sr=sample_rate)
        num_segment = int(audio_duration/SEGMENT_DURATION)
        samples_per_segment = int(sample_rate*audio_duration/num_segment)

        for n in range(num_segment):
            audio_seg = audio[samples_per_segment*n: samples_per_segment*(n
    +1)]

            # Chromagram
            chromagram = librosa.feature.chroma_stft(y=audio_seg, sr=
    sample_rate)
            csv["chroma_stft_mean"].append(chromagram.mean())
            csv["chroma_stft_var"].append(chromagram.var())

            # Root Mean Square Energy
            RMSEn= librosa.feature.rms(y=audio_seg)
            csv["rms_mean"].append(RMSEn.mean())
            csv["rms_var"].append(RMSEn.var())

            # Spectral Centroid
            spec_cent=librosa.feature.spectral_centroid(y=audio_seg)
            csv["spectral_centroid_mean"].append(spec_cent.mean())
            csv["spectral_centroid_var"].append(spec_cent.var())

            #Spectral Bandwith
            spec_band=librosa.feature.spectral_bandwidth(y=audio_seg,sr=
    sample_rate)
            csv["spectral_bandwidth_mean"].append(spec_band.mean())
            csv["spectral_bandwidth_var"].append(spec_band.var())

            # Spectral Rolloff
            spec_roll=librosa.feature.spectral_rolloff(y=audio_seg,sr=
    sample_rate)
            csv["rolloff_mean"].append(spec_roll.mean())
            csv["rolloff_var"].append(spec_roll.var())

            # Zero Crossing Rate
            zero_crossing=librosa.feature.zero_crossing_rate(y=audio_seg)
            csv["zero_crossing_rate_mean"].append(zero_crossing.mean())
            csv["zero_crossing_rate_var"].append(zero_crossing.var())

            # Harmonics and Perceptrual
            harmony, perceptr = librosa.effects.hpss(y=audio_seg)
            csv["harmony_mean"].append(harmony.mean())
            csv["harmony_var"].append(harmony.var())
            csv["perceptr_mean"].append(perceptr.mean())
            csv["perceptr_var"].append(perceptr.var())

            # Tempo
            tempo = librosa.feature.tempo(y = audio_seg, sr = sample_rate)
            csv["tempo"].append(tempo.item())

            # Mfcc
            mfcc=librosa.feature.mfcc(y=audio_seg,sr=sample_rate)
```

```
            mfcc=mfcc.T
            for x in range(20):
                feat1 = "mfcc" + str(x+1) + "_mean"
                feat2 = "mfcc" + str(x+1) + "_var"
                csv[feat1].append(mfcc[:,x].mean())
                csv[feat2].append(mfcc[:,x].var())

        df = pd.DataFrame(csv)
        df.to_csv(f"{SAVE_LOCATION}/{genre}/{genre}{i}.csv", index=False)
        i += 1
```

Plik Analyze_data.ipynb

```
import os
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

SEGMENT_DURATION = 3 # in sec
audio_files = {
        'blues': f"Audio_Features_{SEGMENT_DURATION}sec/blues/",
        'classical': f"Audio_Features_{SEGMENT_DURATION}sec/classical/",
        'country': f"Audio_Features_{SEGMENT_DURATION}sec/country/",
        'disco': f"Audio_Features_{SEGMENT_DURATION}sec/disco/",
        'hiphop': f"Audio_Features_{SEGMENT_DURATION}sec/hiphop/",
        'jazz': f"Audio_Features_{SEGMENT_DURATION}sec/jazz/",
        'metal': f"Audio_Features_{SEGMENT_DURATION}sec/metal/",
        'pop': f"Audio_Features_{SEGMENT_DURATION}sec/pop/",
        'reggae': f"Audio_Features_{SEGMENT_DURATION}sec/reggae/",
        'rock': f"Audio_Features_{SEGMENT_DURATION}sec/rock/"
}
data = []

for genre in audio_files:
    for f in os.listdir(audio_files[genre]):
        csv_df = pd.read_csv(f"{audio_files[genre]}{f}", header = 0)
        csv_df["genre"] = genre
        data.append(csv_df)

df = pd.concat(data, axis = 0, ignore_index=True)

y = df.pop('genre')
X = df

scaler = MinMaxScaler()
X = scaler.fit_transform(X)

pca = PCA(n_components=2)
pca_components = pd.DataFrame(data = pca.fit_transform(X), columns = ["
    principal component 1", "principal component 2"])
pca_components = pd.concat([pca_components, y], axis = 1)
```

```
plt.figure(figsize = (16, 9))
sns.scatterplot(x = "principal component 1", y = "principal component 2",
    data = pca_components, hue = "genre", alpha = 0.7, s = 50)
plt.title("Audio features PCA Decomposition", fontweight = "bold")
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.show()

correlation_matrix = df.filter(regex = r'\b(?!\w*mfcc\w*)\w+\b')
correlation_matrix = correlation_matrix.corr()
plt.figure(figsize = (12, 12))
sns.heatmap(correlation_matrix, cmap = "bwr", annot=True)
plt.title("Correlation matrix for some audio features", fontweight = "bold"
    )
plt.show()
```

Plik Test_Classifiers.ipynb

```
import random
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score

FILES_PER_GENRE = 100
TRAIN_SIZE = 0.8
TEST_NUMBER = 10

genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', '
    metal', 'pop', 'reggae', 'rock']
audio_files_1sec = {
        'blues': f"Audio_Features_1sec/blues/",
        'classical': f"Audio_Features_1sec/classical/",
        'country': f"Audio_Features_1sec/country/",
        'disco': f"Audio_Features_1sec/disco/",
        'hiphop': f"Audio_Features_1sec/hiphop/",
        'jazz': f"Audio_Features_1sec/jazz/",
        'metal': f"Audio_Features_1sec/metal/",
        'pop': f"Audio_Features_1sec/pop/",
        'reggae': f"Audio_Features_1sec/reggae/",
        'rock': f"Audio_Features_1sec/rock/"
}
audio_files_3sec = {
        'blues': f"Audio_Features_3sec/blues/",
        'classical': f"Audio_Features_3sec/classical/",
        'country': f"Audio_Features_3sec/country/",
        'disco': f"Audio_Features_3sec/disco/",
        'hiphop': f"Audio_Features_3sec/hiphop/",
        'jazz': f"Audio_Features_3sec/jazz/",
        'metal': f"Audio_Features_3sec/metal/",
        'pop': f"Audio_Features_3sec/pop/",
        'reggae': f"Audio_Features_3sec/reggae/",
```

```
        'rock': f"Audio_Features_3sec/rock/"
}

audio_files_5sec = {
        'blues': f"Audio_Features_5sec/blues/",
        'classical': f"Audio_Features_5sec/classical/",
        'country': f"Audio_Features_5sec/country/",
        'disco': f"Audio_Features_5sec/disco/",
        'hiphop': f"Audio_Features_5sec/hiphop/",
        'jazz': f"Audio_Features_5sec/jazz/",
        'metal': f"Audio_Features_5sec/metal/",
        'pop': f"Audio_Features_5sec/pop/",
        'reggae': f"Audio_Features_5sec/reggae/",
        'rock': f"Audio_Features_5sec/rock/"
}

audio_files_10sec = {
        'blues': f"Audio_Features_10sec/blues/",
        'classical': f"Audio_Features_10sec/classical/",
        'country': f"Audio_Features_10sec/country/",
        'disco': f"Audio_Features_10sec/disco/",
        'hiphop': f"Audio_Features_10sec/hiphop/",
        'jazz': f"Audio_Features_10sec/jazz/",
        'metal': f"Audio_Features_10sec/metal/",
        'pop': f"Audio_Features_10sec/pop/",
        'reggae': f"Audio_Features_10sec/reggae/",
        'rock': f"Audio_Features_10sec/rock/"
}

audio_files_30sec = {
        'blues': f"Audio_Features_30sec/blues/",
        'classical': f"Audio_Features_30sec/classical/",
        'country': f"Audio_Features_30sec/country/",
        'disco': f"Audio_Features_30sec/disco/",
        'hiphop': f"Audio_Features_30sec/hiphop/",
        'jazz': f"Audio_Features_30sec/jazz/",
        'metal': f"Audio_Features_30sec/metal/",
        'pop': f"Audio_Features_30sec/pop/",
        'reggae': f"Audio_Features_30sec/reggae/",
        'rock': f"Audio_Features_30sec/rock/"
}

class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        predictions=[]
        voters=[]
        for X in X_test.values:
            dst = self.X_train-X
```

```
            dst = dst * dst
            dst = dst.sum(axis='columns') ** 0.5
            dst = pd.DataFrame({'distance': dst,
                                'value': self.y_train})
            dst = dst.sort_values('distance')
            voters = dst.head(self.k).value
            predictions.append(voters.mode()[0])
        return predictions

test_results = {
    'segment_duration': [],
    'n_neighbors': [],
    'accuracy': []
}

for i in range(TEST_NUMBER):
    print(f"Test nr. {i+1}")
    train_files = {
        '1sec': [],
        '3sec': [],
        '5sec': [],
        '10sec': [],
        '30sec': [],
    }
    train_files_genre = []
    test_files = {
        '1sec': [],
        '3sec': [],
        '5sec': [],
        '10sec': [],
        '30sec': [],
    }
    test_files_genre = []

    for genre in genres:
        random_indexes = random.sample(range(FILES_PER_GENRE),
    FILES_PER_GENRE)
        train_indexes = random_indexes[:int(TRAIN_SIZE*FILES_PER_GENRE)]
        test_indexes = random_indexes[int(TRAIN_SIZE*FILES_PER_GENRE):]
        for n in train_indexes:
            train_files['1sec'].append(f"{audio_files_1sec[genre]}{genre}{n
    }.csv")
            train_files['3sec'].append(f"{audio_files_3sec[genre]}{genre}{n
    }.csv")
            train_files['5sec'].append(f"{audio_files_5sec[genre]}{genre}{n
    }.csv")
            train_files['10sec'].append(f"{audio_files_10sec[genre]}{genre
    }{n}.csv")
            train_files['30sec'].append(f"{audio_files_30sec[genre]}{genre
    }{n}.csv")
            train_files_genre.append(genre)
        for n in test_indexes:
            test_files['1sec'].append(f"{audio_files_1sec[genre]}{genre}{n
    }.csv")
            test_files['3sec'].append(f"{audio_files_3sec[genre]}{genre}{n
```

```
    }.csv")
140             test_files['5sec'].append(f"{audio_files_5sec[genre]}{genre}{n
    }.csv")
141             test_files['10sec'].append(f"{audio_files_10sec[genre]}{genre}{
    n}.csv")
142             test_files['30sec'].append(f"{audio_files_30sec[genre]}{genre}{
    n}.csv")
143             test_files_genre.append(genre)
144
145     for segment_duation in (3, 5, 10, 30):
146         X_train_files = train_files[f'{segment_duation}sec']
147         X_test_files = test_files[f'{segment_duation}sec']
148         X_train_data = []
149         for i, f in enumerate(X_train_files):
150             csv_df = pd.read_csv(f, header = 0)
151             csv_df["genre"] = train_files_genre[i]
152             X_train_data.append(csv_df)
153
154         X_train = pd.concat(X_train_data, axis = 0, ignore_index = True)
155         y_train = X_train.pop("genre")
156         scaler = MinMaxScaler()
157         scaler.set_output(transform="pandas")
158         X_train = scaler.fit_transform(X_train)
159
160         X_test = []
161         y_test = test_files_genre
162         for f in X_test_files:
163             X = pd.read_csv(f)
164             X = scaler.transform(X)
165             X_test.append(X)
166
167         for neighbors in range(1, 11):
168             print(f"Predicting for: segment_duration = {segment_duation}sec
    , n_neighbors = {neighbors}")
169             clf= KNN(neighbors)
170             clf.fit(X_train, y_train)
171
172             y_pred = []
173             for X in X_test:
174                 predictions = clf.predict(X)
175                 possible_genres, counts = np.unique(predictions,
    return_counts=True)
176                 genre = possible_genres[counts.argmax()]
177                 y_pred.append(genre)
178
179             test_results['segment_duration'].append(segment_duation)
180             test_results['n_neighbors'].append(neighbors)
181             test_results['accuracy'].append(accuracy_score(y_test, y_pred))
182
183         classifier_test_df = pd.DataFrame(test_results)
184         classifier_test_df.to_csv(f"classifier_test_results.csv", index=
    False)
185
186 data = pd.read_csv("classifier_test_results.csv")
187
```

```
data_3sec = data[data['segment_duration'] == 3]
data_3sec = data_3sec.drop(columns=['segment_duration'])
data_3sec = data_3sec.groupby('n_neighbors')['accuracy'].mean()

data_5sec = data[data['segment_duration'] == 5]
data_5sec = data_5sec.drop(columns=['segment_duration'])
data_5sec = data_5sec.groupby('n_neighbors')['accuracy'].mean()

data_10sec = data[data['segment_duration'] == 10]
data_10sec = data_10sec.drop(columns=['segment_duration'])
data_10sec = data_10sec.groupby('n_neighbors')['accuracy'].mean()

data_30sec = data[data['segment_duration'] == 30]
data_30sec = data_30sec.drop(columns=['segment_duration'])
data_30sec = data_30sec.groupby('n_neighbors')['accuracy'].mean()

fig, ax = plt.subplots(2, 2, figsize=(15, 12))
fig.suptitle("Number of neighbors effect on accuracy", fontweight = "bold")
ax[0, 0].plot(data_3sec.keys(), data_3sec.values*100, color = "blue")
ax[0, 0].set_xticks(data_3sec.keys())
ax[0, 0].set_xlabel("n_neighbors")
ax[0, 0].set_ylabel("accuracy")
ax[0, 0].grid(True)
ax[0, 0].title.set_text("3 second segments")

ax[0, 1].plot(data_5sec.keys(), data_5sec.values*100, color = "red")
ax[0, 1].set_xticks(data_5sec.keys())
ax[0, 1].set_xlabel("n_neighbors")
ax[0, 1].set_ylabel("accuracy")
ax[0, 1].grid(True)
ax[0, 1].title.set_text("5 second segments")

ax[1, 0].plot(data_10sec.keys(), data_10sec.values*100, color = "green")
ax[1, 0].set_xticks(data_10sec.keys())
ax[1, 0].set_xlabel("n_neighbors")
ax[1, 0].set_ylabel("accuracy")
ax[1, 0].grid(True)
ax[1, 0].title.set_text("10 second segments")

ax[1, 1].plot(data_30sec.keys(), data_30sec.values*100, color = "orange")
ax[1, 1].set_xticks(data_30sec.keys())
ax[1, 1].set_xlabel("n_neighbors")
ax[1, 1].set_ylabel("accuracy")
ax[1, 1].grid(True)
ax[1, 1].title.set_text("30 second segments")

plt.show()

plt.figure(figsize = (12, 8))
plt.plot(data_3sec.keys(), data_3sec.values*100, color = "blue", label="3
    sec")
plt.plot(data_5sec.keys(), data_5sec.values*100, color = "red", label="5
    sec")
plt.plot(data_10sec.keys(), data_10sec.values*100, color = "green", label="
    10 sec")
```

```
plt.plot(data_30sec.keys(), data_30sec.values*100, color = "orange", label=
    "30 sec")
plt.xticks(data_3sec.keys())
plt.yticks(np.arange(66, 78, 1))
plt.xlabel("n_neighbors")
plt.ylabel("accuracy")
plt.grid(True)
plt.title("Number of neighbors effect on accuracy", fontweight = "bold")
plt.legend(loc="upper right", title = "Segment length")
plt.show()
```

Plik Classify_music.ipynb

```
import random
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, confusion_matrix

SEGMENT_DURATION = 3 # in sec
FILES_PER_GENRE = 100
TRAIN_SIZE = 0.8

audio_files = {
        'blues': f"Audio_Features_{SEGMENT_DURATION}sec/blues/",
        'classical': f"Audio_Features_{SEGMENT_DURATION}sec/classical/",
        'country': f"Audio_Features_{SEGMENT_DURATION}sec/country/",
        'disco': f"Audio_Features_{SEGMENT_DURATION}sec/disco/",
        'hiphop': f"Audio_Features_{SEGMENT_DURATION}sec/hiphop/",
        'jazz': f"Audio_Features_{SEGMENT_DURATION}sec/jazz/",
        'metal': f"Audio_Features_{SEGMENT_DURATION}sec/metal/",
        'pop': f"Audio_Features_{SEGMENT_DURATION}sec/pop/",
        'reggae': f"Audio_Features_{SEGMENT_DURATION}sec/reggae/",
        'rock': f"Audio_Features_{SEGMENT_DURATION}sec/rock/"
}

train_files = []
train_files_genre = []
test_files = []
test_files_genre = []
for genre in audio_files:
    random_indexes = random.sample(range(FILES_PER_GENRE), FILES_PER_GENRE)
    train_indexes = random_indexes[:int(TRAIN_SIZE*FILES_PER_GENRE)]
    test_indexes = random_indexes[int(TRAIN_SIZE*FILES_PER_GENRE):]
    for n in train_indexes:
        train_files.append(f"{audio_files[genre]}{genre}{n}.csv")
        train_files_genre.append(genre)
    for n in test_indexes:
        test_files.append(f"{audio_files[genre]}{genre}{n}.csv")
        test_files_genre.append(genre)

```

```
class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X_train, y_train):
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        predictions=[]
        voters=[]
        for X in X_test.values:
            dst = self.X_train-X
            dst = dst * dst
            dst = dst.sum(axis='columns') ** 0.5
            dst = pd.DataFrame({'distance': dst,
                                'value': self.y_train})
            dst = dst.sort_values('distance')
            voters = dst.head(self.k).value
            predictions.append(voters.mode()[0])
        return predictions

X_train_data = []
for i, f in enumerate(train_files):
    csv_df = pd.read_csv(f, header = 0)
    csv_df["genre"] = train_files_genre[i]
    X_train_data.append(csv_df)

X_train = pd.concat(X_train_data, axis = 0, ignore_index = True)
y_train = X_train.pop("genre")
scaler = MinMaxScaler()
scaler.set_output(transform="pandas")
X_train = scaler.fit_transform(X_train)
clf = KNN(1)
clf.fit(X_train, y_train)

y_test = test_files_genre
y_pred = []
for f in test_files:
    X_test = pd.read_csv(f)
    X_test = scaler.transform(X_test)
    predictions = clf.predict(X_test)
    possible_genres, counts = np.unique(predictions, return_counts=True)
    genre = possible_genres[counts.argmax()]
    y_pred.append(genre)

print(f"Dokładność: {accuracy_score(y_test, y_pred)*100}%")

cm = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize = (10, 10))
s = sns.heatmap(cm, annot=True, xticklabels = audio_files.keys(),
   yticklabels = audio_files.keys(), ax = ax)
s.set_xlabel("Predicted genre", fontsize = 15)
s.set_ylabel("True genre", fontsize = 15)
plt.title("Music genre classification confussion matrix", fontweight = "
```

```
    bold")
plt.show()
```