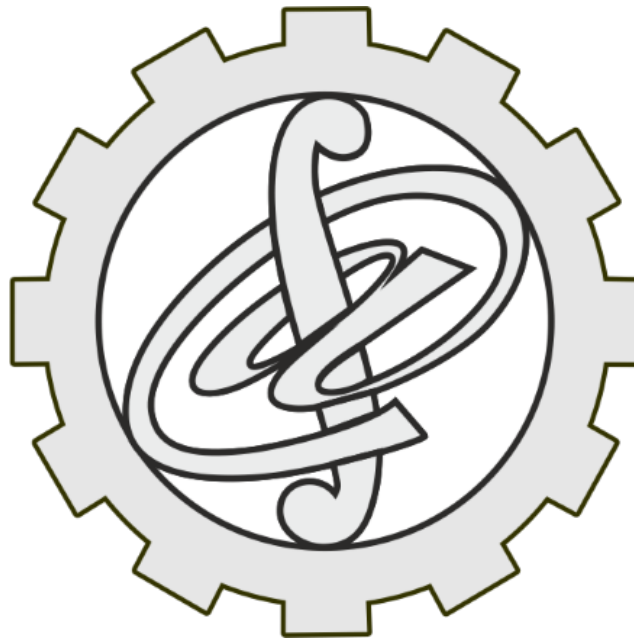


Systemy sztucznej inteligencji

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.

```
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_crossi
0.2723272,0.09016607,0.044159543,0.00025018366,1282.9845011767718,77574.32251806105,1726.0138135656548,66589.36635207309,2511.2684044471152,676298.8771939419,0.05169396033653846,
0.26111427,0.085881434,0.038686298,0.0005414083,1307.1235404772044,59359.08688722552,1663.8335582064083,52173.14077666943,2517.148625300481,458048.16384220275,0.05347430889423077,
0.22769697,0.07900824,0.100510195,0.0009806289,1638.3832766667697,38396.36271340232,1702.2630998912157,10387.256979915059,3023.0132587139424,174835.92884600518,0.078125,0.0002387,
0.22878258,0.084998496,0.056821927,0.00039984507,1280.554156575394,65485.40952065723,1773.9042716040617,60407.331079484975,2479.962439903846,730366.4909193502,0.04875676081730769,
0.25124887,0.091251194,0.07329995,0.0011551806,1824.5425224140272,292679.21314842126,1739.3299531840353,18175.078267879508,3325.886042668269,998895.1969279341,0.10946514423076924,
0.2803955,0.09216021,0.038523056,0.00020710824,1295.011182171602,131819.78612800705,1666.509304986024,42845.734033255234,2503.4833233173076,584226.8016083,0.06132061298076923,0.0,
0.21878338,0.082384214,0.08497642,0.0013451182,1580.7089179788338,109875.25963268403,1721.4881945444374,47457.078220126794,3047.5279822716348,494642.4384329093,0.0790301983173076,
0.23756126,0.08965432,0.056573384,0.00058733387,1749.8269331641159,84381.71146548055,1962.0365399048855,57846.66143620123,3555.214655949510,435122.8121216128,0.08340594951923076,
0.28028622,0.097626865,0.06930908,0.0028846865,1484.7077067544235,67389.61058753535,1769.253335437665,85713.56821714446,2837.330791766827,570353.8394725181,0.07291917067307692,0.
0.25307417,0.08239199,0.039156035,0.00025110634,1240.6357962800002,83731.37170341353,1703.272837256291,92600.36583300885,2443.1075345552886,832175.1704585976,0.05318509615384615,
```

Rysunek 1: Plik csv z wyodrębnionymi cechami dźwięku

Dodatkowe informacje

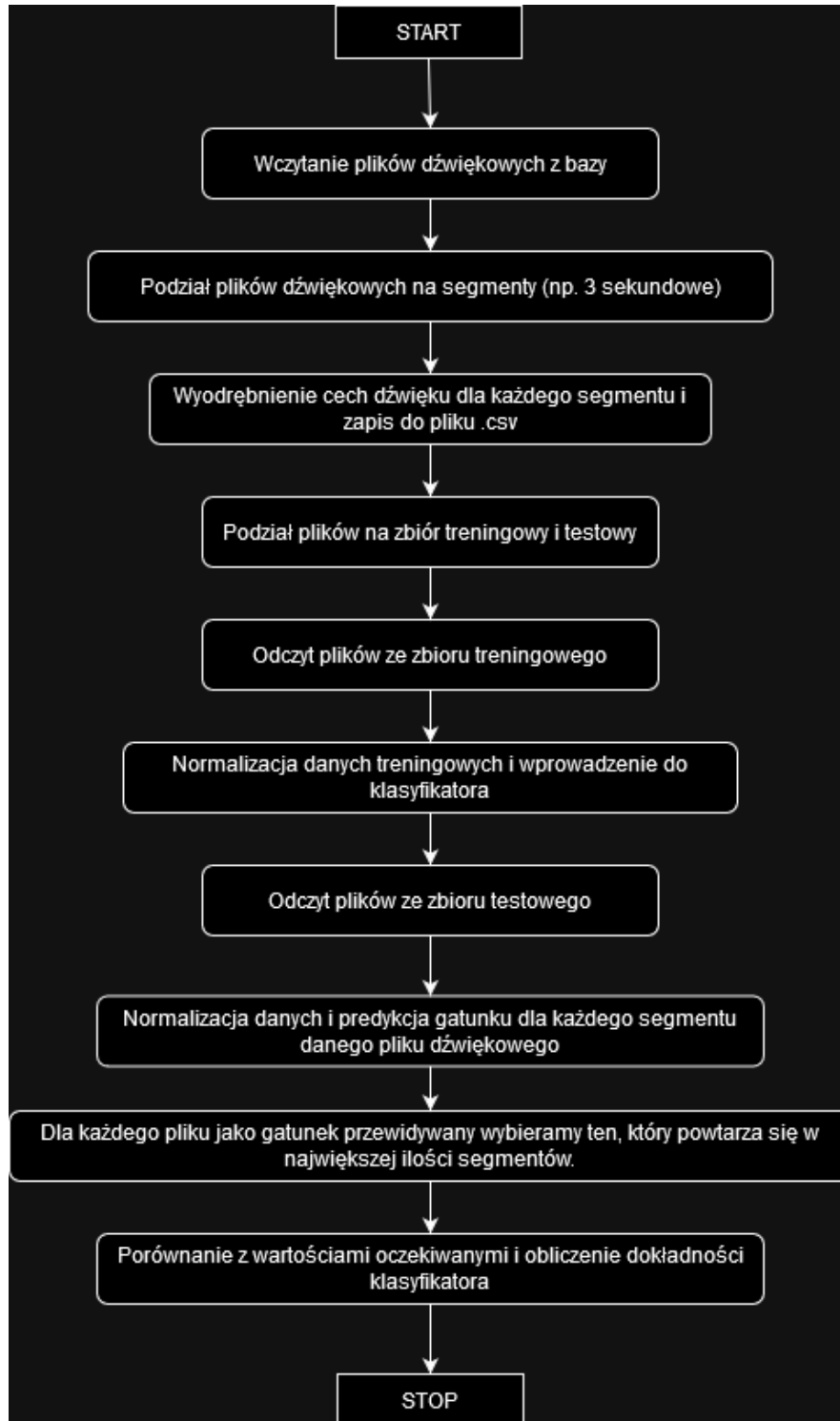
Do poprawnego działania wymagany jest Python 3.11 oraz następujące biblioteki:

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

Część II

Opis działania

Poniżej znajduje się schemat, opisujący sposób działania systemu.



Rysunek 2: Schemat działania systemu klasyfikacji muzyki

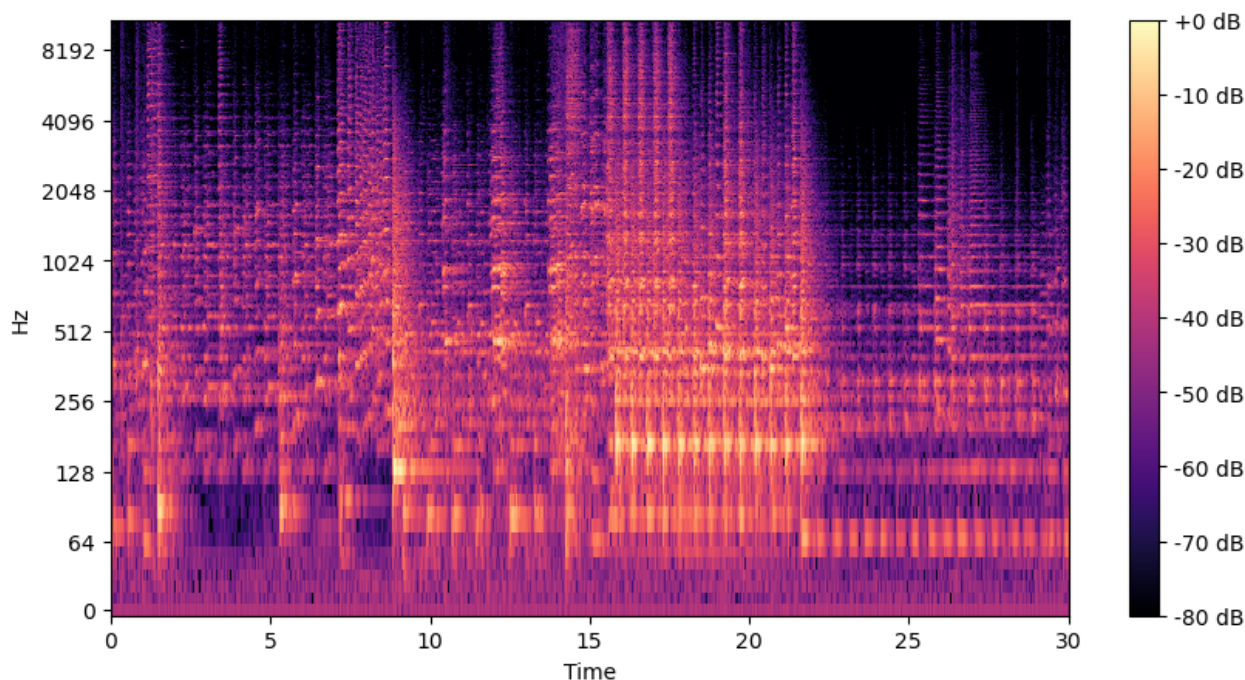
Analiza dźwięku

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.

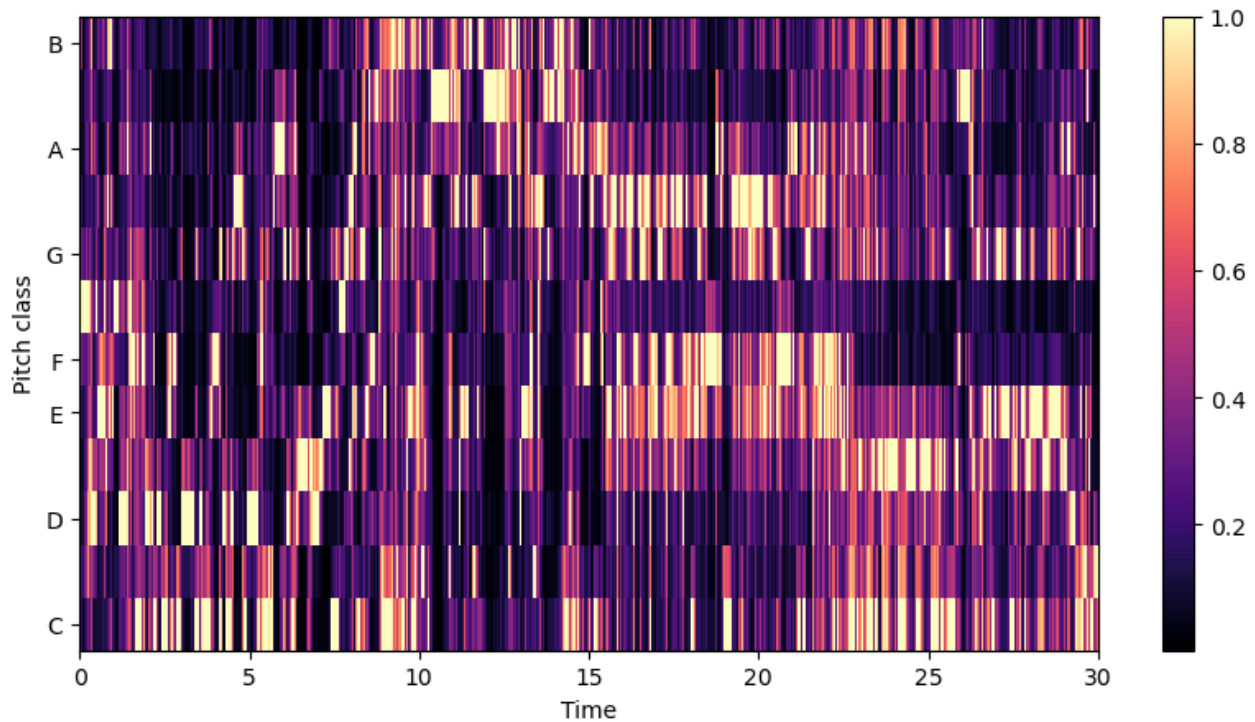
Aby zwiększyć ilość danych treningowych oraz dokładność modelu, pliki dźwiękowe są poddawane podziałowi na krótsze segmenty, dla których z osobna obliczane są cechy. Każdy segment dźwiękowy charakteryzowany jest przez 57 różnych wartości liczbowych.

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: Spektrogram 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

```
1 for genre in audio_files:
2     i = 0
3     for f in os.listdir(audio_files[genre]):
4         csv = copy.deepcopy(csv_template)
5         f = audio_files[genre] + "/" + f
6         print("Processing:", f)
7
8         audio, sample_rate = librosa.load(f)
9         audio_duration = librosa.get_duration(y=audio, sr=sample_rate)
10        num_segment = int(audio_duration/SEGMENT_DURATION)
11        samples_per_segment = int(sample_rate*audio_duration/num_segment)
12
13        for n in range(num_segment):
14            audio_seg = audio[samples_per_segment*n: samples_per_segment*(n
15                +1)]
16
17            # Chromagram
18            chromagram = librosa.feature.chroma_stft(y=audio_seg, sr=
19                sample_rate)
20            csv["chroma_stft_mean"].append(chromagram.mean())
21            csv["chroma_stft_var"].append(chromagram.var())
22
23            # Root Mean Square Energy
24            RMSEn= librosa.feature.rms(y=audio_seg)
25            csv["rms_mean"].append(RMSEn.mean())
26            csv["rms_var"].append(RMSEn.var())
27
28            # Spectral Centroid
29            spec_cent=librosa.feature.spectral_centroid(y=audio_seg)
30            csv["spectral_centroid_mean"].append(spec_cent.mean())
31            csv["spectral_centroid_var"].append(spec_cent.var())
32
33            #Spectral Bandwidth
34            spec_band=librosa.feature.spectral_bandwidth(y=audio_seg,sr=
35                sample_rate)
36            csv["spectral_bandwidth_mean"].append(spec_band.mean())
37            csv["spectral_bandwidth_var"].append(spec_band.var())
38
39            # Spectral Rolloff
40            spec_roll=librosa.feature.spectral_rolloff(y=audio_seg,sr=
41                sample_rate)
42            csv["rolloff_mean"].append(spec_roll.mean())
43            csv["rolloff_var"].append(spec_roll.var())
44
45            # Zero Crossing Rate
46            zero_crossing=librosa.feature.zero_crossing_rate(y=audio_seg)
```

```

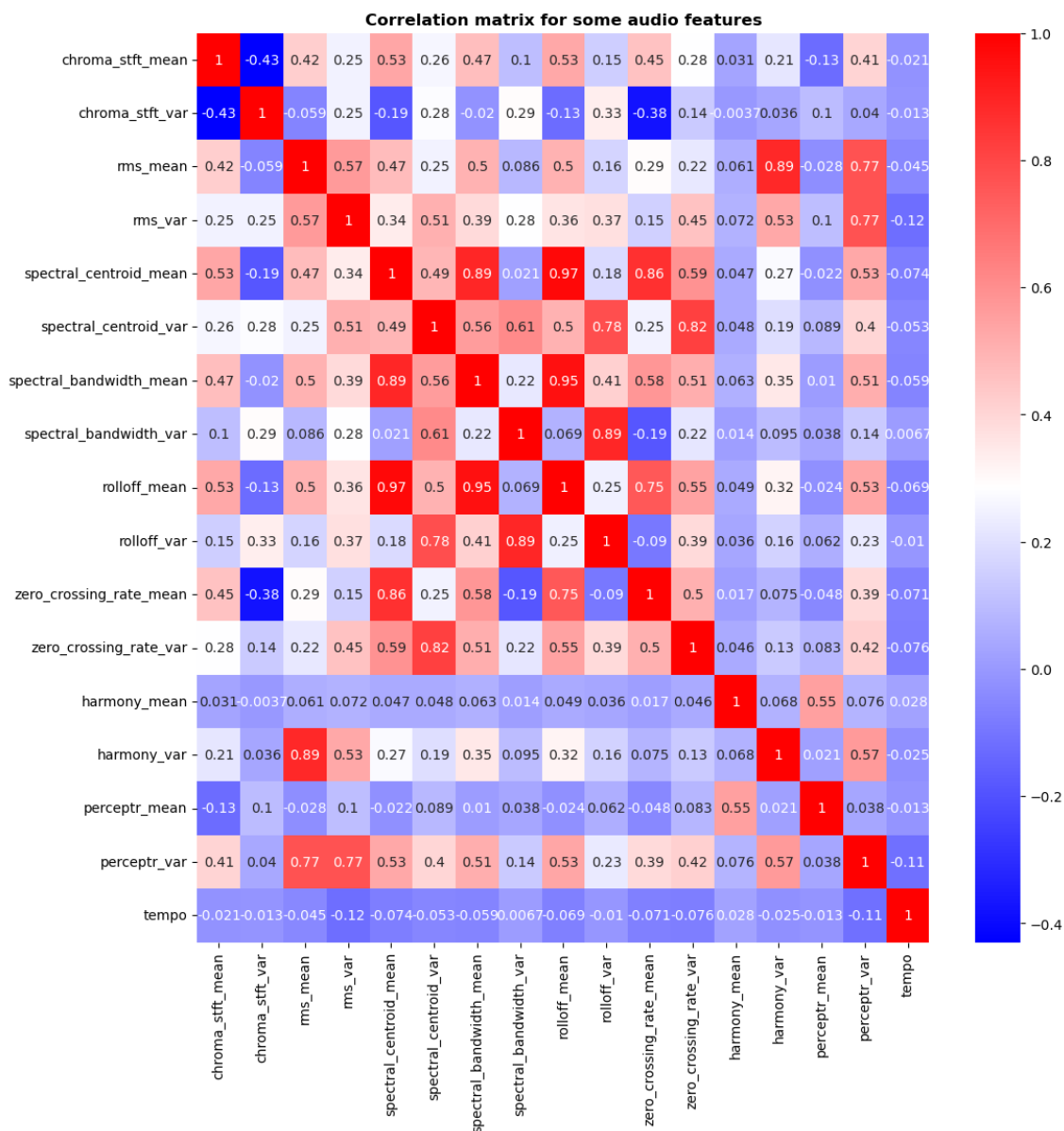
43     csv["zero_crossing_rate_mean"].append(zero_crossing.mean())
44     csv["zero_crossing_rate_var"].append(zero_crossing.var())
45
46     # Harmonics and Perceptual
47     harmony, perceptr = librosa.effects.hpss(y=audio_seg)
48     csv["harmony_mean"].append(harmony.mean())
49     csv["harmony_var"].append(harmony.var())
50     csv["perceptr_mean"].append(perceptr.mean())
51     csv["perceptr_var"].append(perceptr.var())
52
53     # Tempo
54     tempo = librosa.feature.tempo(y = audio_seg, sr = sample_rate)
55     csv["tempo"].append(tempo.item())
56
57     # Mfcc
58     mfcc=librosa.feature.mfcc(y=audio_seg,sr=sample_rate)
59     mfcc=mfcc.T
60     for x in range(20):
61         feat1 = "mfcc" + str(x+1) + "_mean"
62         feat2 = "mfcc" + str(x+1) + "_var"
63         csv[feat1].append(mfcc[:,x].mean())
64         csv[feat2].append(mfcc[:,x].var())
65
66     df = pd.DataFrame(csv)
67     df.to_csv(f"{SAVE_LOCATION}/{genre}/{genre}{i}.csv", index=False)
68     i += 1

```


Analiza danych

Macierz korelacji

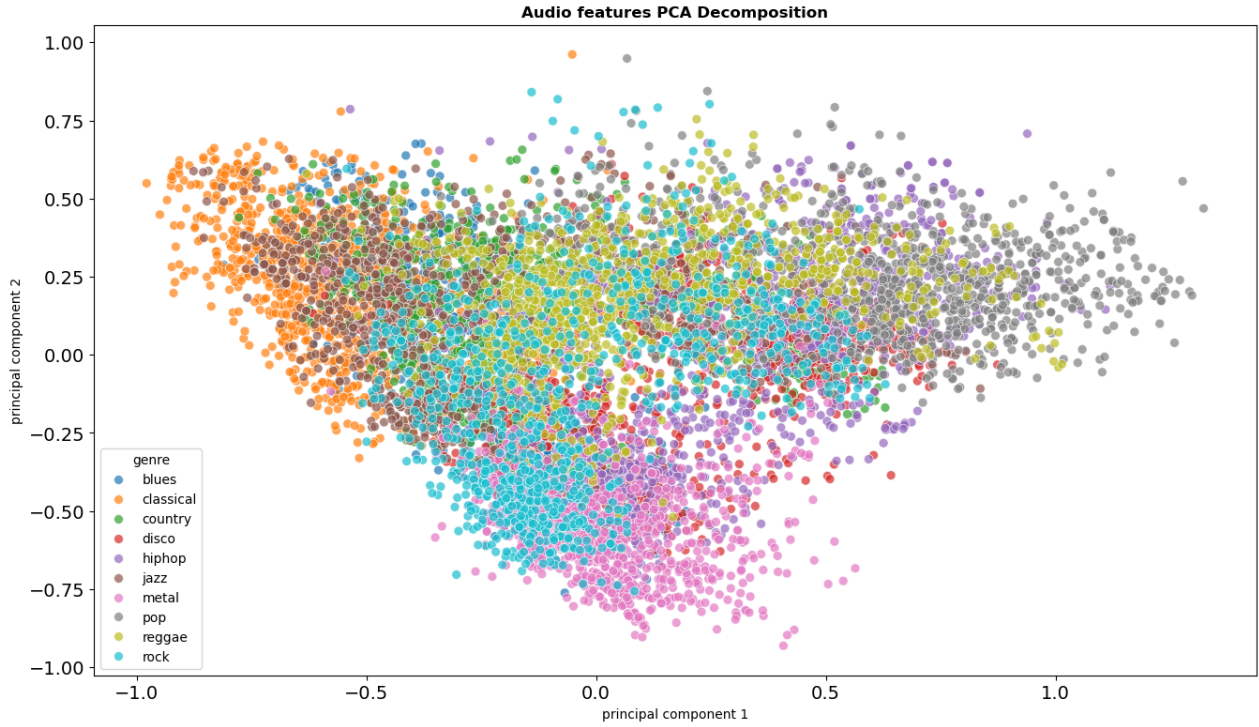
Poniżej przedstawiono macierz korelacji dla niektórych cech dźwięku, wyraźnie widać, że większość z nich jest od siebie w mniejszym lub większym stopniu zależna. W praktyce zachowanie wszystkich cech przynosiło jednak najlepsze rezultaty.



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 komponentów przedstawione jest zaledwie około 45% całkowitej wariancji, dlatego poniższy wykres należy traktować jedynie jako ciekawostkę.



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 normalizację 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 ← read data from file f
| X_test ← scale data using scaler
| predictions ← clf.predict(X_test)
| // Finding the most frequent genre
| possible_genres, counts ← np.unique(predictions, return_counts = True)
| genre ← possible_genres[counts.argmax()]
| y_pred.append(genre)
end
// Calculating accuracy
```

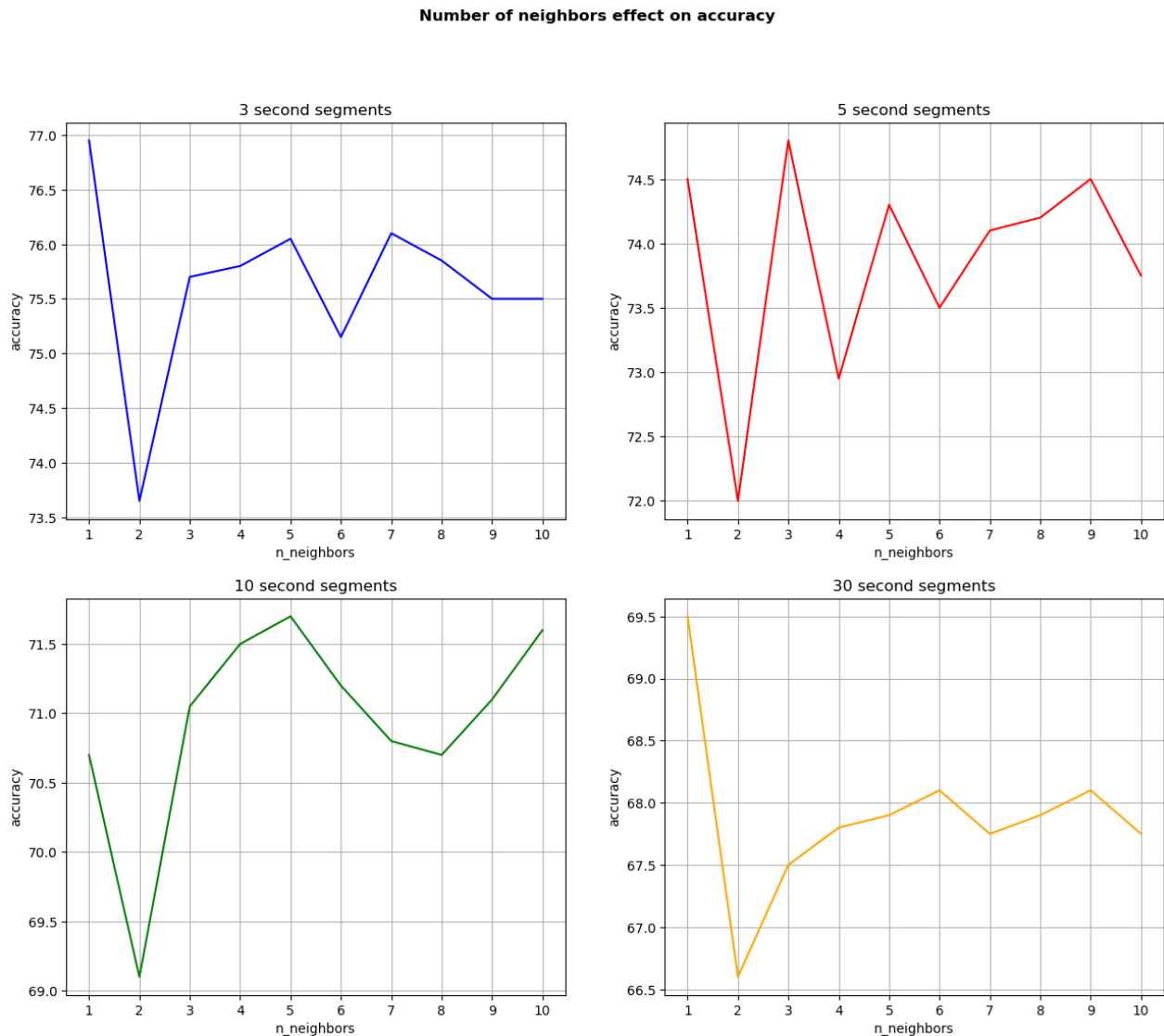
Implementacja

```
1 X_train_data = []
2 for i, f in enumerate(train_files):
3     csv_df = pd.read_csv(f, header = 0)
4     csv_df["genre"] = train_files_genre[i]
5     X_train_data.append(csv_df)
6
7 X_train = pd.concat(X_train_data, axis = 0, ignore_index = True)
8 y_train = X_train.pop("genre")
9 scaler = MinMaxScaler()
10 scaler.set_output(transform="pandas")
11 X_train = scaler.fit_transform(X_train)
12 clf = KNN(1)
13 clf.fit(X_train, y_train)
14
15 y_test = test_files_genre
16 y_pred = []
17 for f in test_files:
18     X_test = pd.read_csv(f)
19     X_test = scaler.transform(X_test)
20     predictions = clf.predict(X_test)
21     possible_genres, counts = np.unique(predictions, return_counts=True)
22     genre = possible_genres[counts.argmax()]
23     y_pred.append(genre)
24
25 print(f"Dokładność: {accuracy_score(y_test, y_pred)*100}%")
```

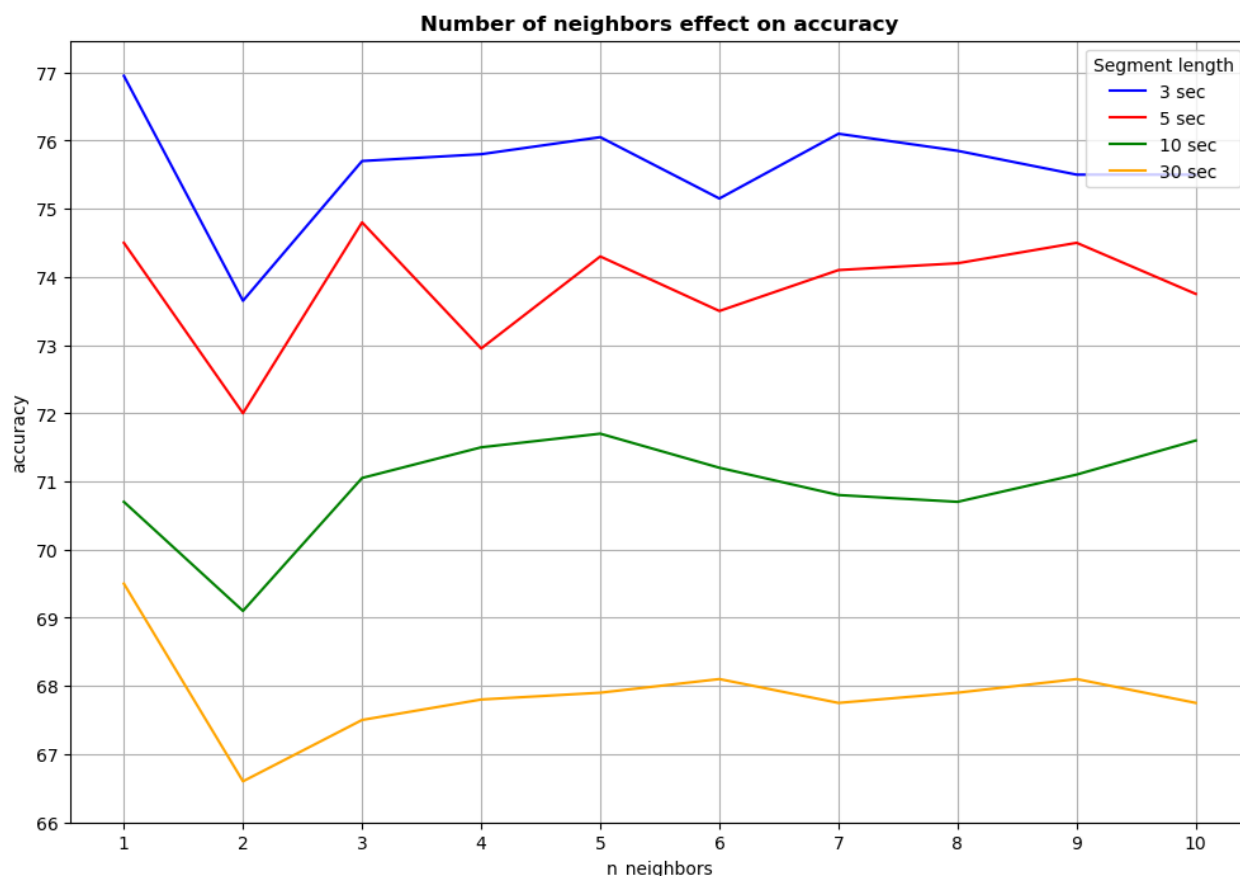
Eksperymenty

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

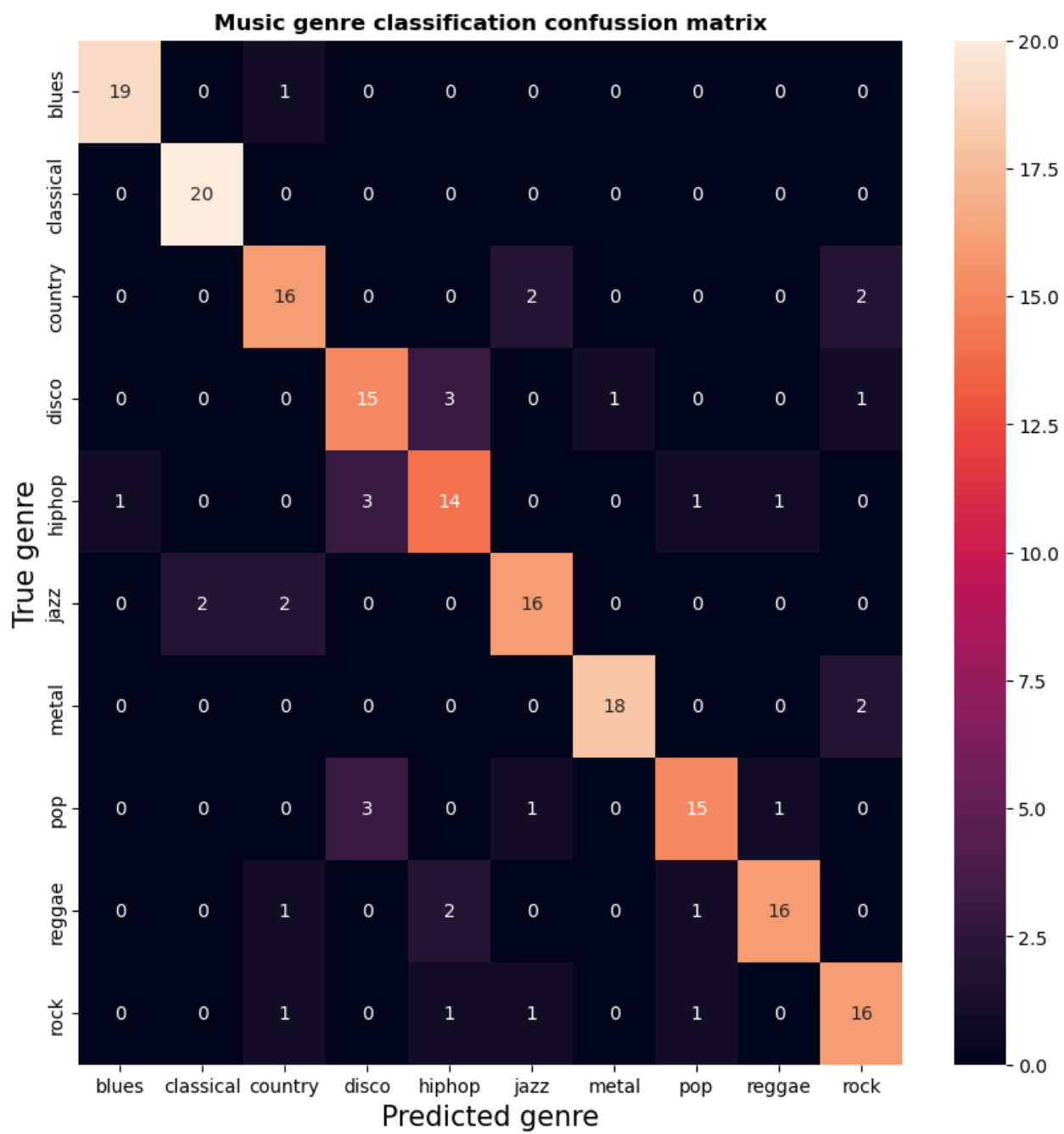


Rysunek 8: Analiza liczby sąsiadów i długości segmentów

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.



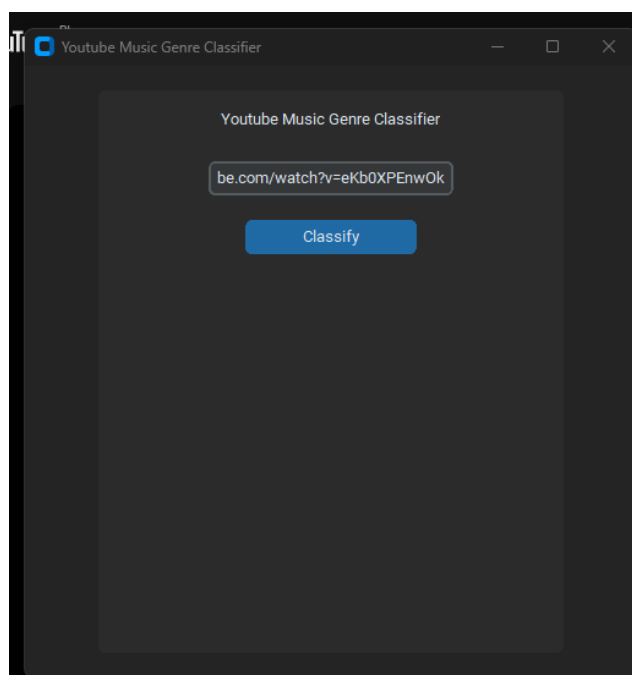
Rysunek 9: Macierz błędów

Testy

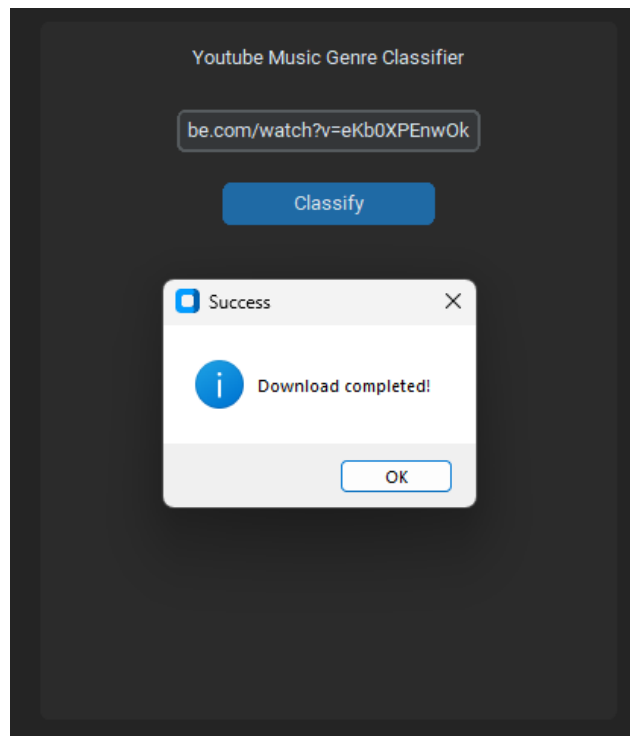
Analiza muzyki spoza bazy

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.

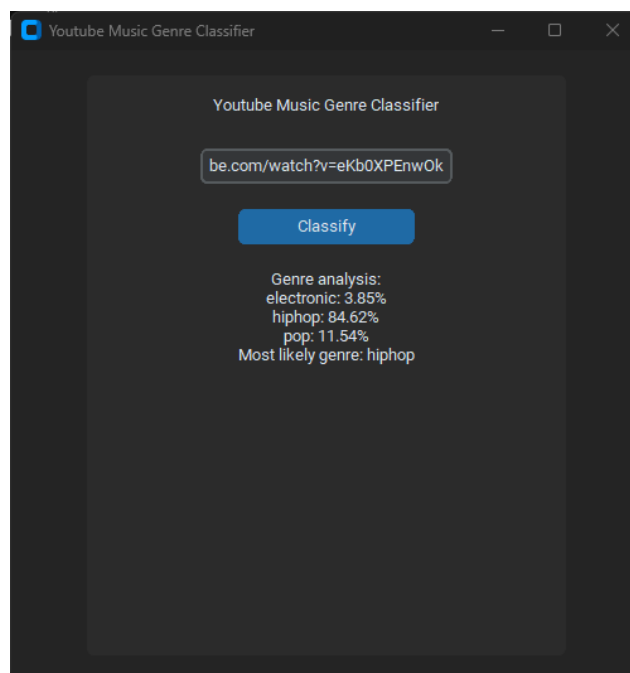
Z powodów wymienionych wcześniej, analiza utworów z bazą danych GTZAN jako zbiór treningowy nie przyniosła najlepszych rezultatów. W związku z tym w ramach testu nasza baza z cechami dźwięku została przez nas rozszerzona o cechy wyciągnięte na podstawie utworów znajdujących się obecnie na listach przebojów. Dodatkowo zamiast przewidywania pojedynczego gatunku, w zastosowaniu praktycznym wypisujemy procentową klasyfikację, która odzwierciedla ilość segmentów utworu przypisanych do konkretnego gatunku. Aby przyspieszyć klasyfikację, 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`

```
1 import os
2 import pandas as pd
3 import librosa
4 import copy
5
6 SEGMENT_DURATION = 1 # in sec
7 SAVE_LOCATION = f"Audio_Features_{SEGMENT_DURATION}sec"
8
9 csv_template= {"chroma_stft_mean": [], "chroma_stft_var": [], "rms_mean":
10               [], "rms_var": [], "spectral_centroid_mean": [],
11               "spectral_centroid_var": [], "spectral_bandwidth_mean": [], "
12               "spectral_bandwidth_var": [], "rolloff_mean": [], "rolloff_var": [],
13               "zero_crossing_rate_mean": [], "zero_crossing_rate_var": [], "
14               "harmony_mean": [], "harmony_var": [], "perceptr_mean": [],
15               "perceptr_var": [], "tempo": [], "mfcc1_mean": [], "mfcc1_var" :
16               [], "mfcc2_mean" : [], "mfcc2_var" : [],
17               "mfcc3_mean" : [], "mfcc3_var" : [], "mfcc4_mean" : [], "mfcc4_var"
18               : [], "mfcc5_mean" : [],
19               "mfcc5_var" : [], "mfcc6_mean" : [], "mfcc6_var" : [], "mfcc7_mean"
20               : [], "mfcc7_var" : [],
21               "mfcc8_mean" : [], "mfcc8_var" : [], "mfcc9_mean" : [], "mfcc9_var"
22               : [], "mfcc10_mean" : [],
23               "mfcc10_var" : [], "mfcc11_mean" : [], "mfcc11_var" : [], "
24               "mfcc12_mean" : [], "mfcc12_var" : [],
25               "mfcc13_mean" : [], "mfcc13_var" : [], "mfcc14_mean" : [], "
26               "mfcc14_var" : [], "mfcc15_mean" : [],
27               "mfcc15_var" : [], "mfcc16_mean" : [], "mfcc16_var" : [], "
28               "mfcc17_mean" : [], "mfcc17_var" : [],
29               "mfcc18_mean" : [], "mfcc18_var" : [], "mfcc19_mean" : [], "
30               "mfcc19_var" : [], "mfcc20_mean" : [],
31               "mfcc20_var":[]}]
32
33 audio_files = {
34     'blues': "gtzan_dataset/blues",
35     'classical': "gtzan_dataset/classical",
36     'country': "gtzan_dataset/country",
37     'disco': "gtzan_dataset/disco",
38     'hiphop': "gtzan_dataset/hiphop",
39     'jazz': "gtzan_dataset/jazz",
40     'metal': "gtzan_dataset/metal",
41     'pop': "gtzan_dataset/pop",
42     'reggae': "gtzan_dataset/reggae",
43     'rock': "gtzan_dataset/rock"
44 }
45
46 for genre in audio_files:
47     i = 0
48     for f in os.listdir(audio_files[genre]):
49         csv = copy.deepcopy(csv_template)
50         f = audio_files[genre] + "/" + f
51         print("Processing:", f)
```

```

41     audio, sample_rate = librosa.load(f)
42     audio_duration = librosa.get_duration(y=audio, sr=sample_rate)
43     num_segment = int(audio_duration/SEGMENT_DURATION)
44     samples_per_segment = int(sample_rate*audio_duration/num_segment)
45
46     for n in range(num_segment):
47         audio_seg = audio[samples_per_segment*n: samples_per_segment*(n
+1)]
48
49         # Chromagram
50         chromagram = librosa.feature.chroma_stft(y=audio_seg, sr=
sample_rate)
51         csv["chroma_stft_mean"].append(chromagram.mean())
52         csv["chroma_stft_var"].append(chromagram.var())
53
54         # Root Mean Square Energy
55         RMSEn= librosa.feature.rms(y=audio_seg)
56         csv["rms_mean"].append(RMSEn.mean())
57         csv["rms_var"].append(RMSEn.var())
58
59         # Spectral Centroid
60         spec_cent=librosa.feature.spectral_centroid(y=audio_seg)
61         csv["spectral_centroid_mean"].append(spec_cent.mean())
62         csv["spectral_centroid_var"].append(spec_cent.var())
63
64         #Spectral Bandwith
65         spec_band=librosa.feature.spectral_bandwidth(y=audio_seg,sr=
sample_rate)
66         csv["spectral_bandwidth_mean"].append(spec_band.mean())
67         csv["spectral_bandwidth_var"].append(spec_band.var())
68
69         # Spectral Rolloff
70         spec_roll=librosa.feature.spectral_rolloff(y=audio_seg,sr=
sample_rate)
71         csv["rolloff_mean"].append(spec_roll.mean())
72         csv["rolloff_var"].append(spec_roll.var())
73
74         # Zero Crossing Rate
75         zero_crossing=librosa.feature.zero_crossing_rate(y=audio_seg)
76         csv["zero_crossing_rate_mean"].append(zero_crossing.mean())
77         csv["zero_crossing_rate_var"].append(zero_crossing.var())
78
79         # Harmonics and Perceptrueal
80         harmony, perceptr = librosa.effects.hpss(y=audio_seg)
81         csv["harmony_mean"].append(harmony.mean())
82         csv["harmony_var"].append(harmony.var())
83         csv["perceptr_mean"].append(perceptr.mean())
84         csv["perceptr_var"].append(perceptr.var())
85
86         # Tempo
87         tempo = librosa.feature.tempo(y = audio_seg, sr = sample_rate)
88         csv["tempo"].append(tempo.item())
89
90         # Mfcc
91         mfcc=librosa.feature.mfcc(y=audio_seg,sr=sample_rate)

```

```

92         mfcc=mfcc.T
93         for x in range(20):
94             feat1 = "mfcc" + str(x+1) + "_mean"
95             feat2 = "mfcc" + str(x+1) + "_var"
96             csv[feat1].append(mfcc[:,x].mean())
97             csv[feat2].append(mfcc[:,x].var())
98
99         df = pd.DataFrame(csv)
100         df.to_csv(f"{SAVE_LOCATION}/{genre}/{genre}{i}.csv", index=False)
101         i += 1

```

Plik Analyze_data.ipynb

```

1 import os
2 import pandas as pd
3
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6
7 from sklearn.preprocessing import MinMaxScaler
8 from sklearn.decomposition import PCA
9
10 SEGMENT_DURATION = 3 # in sec
11 audio_files = {
12     'blues': f"Audio_Features_{SEGMENT_DURATION}sec/blues/",
13     'classical': f"Audio_Features_{SEGMENT_DURATION}sec/classical/",
14     'country': f"Audio_Features_{SEGMENT_DURATION}sec/country/",
15     'disco': f"Audio_Features_{SEGMENT_DURATION}sec/disco/",
16     'hiphop': f"Audio_Features_{SEGMENT_DURATION}sec/hiphop/",
17     'jazz': f"Audio_Features_{SEGMENT_DURATION}sec/jazz/",
18     'metal': f"Audio_Features_{SEGMENT_DURATION}sec/metal/",
19     'pop': f"Audio_Features_{SEGMENT_DURATION}sec/pop/",
20     'reggae': f"Audio_Features_{SEGMENT_DURATION}sec/reggae/",
21     'rock': f"Audio_Features_{SEGMENT_DURATION}sec/rock/"
22 }
23 data = []
24
25 for genre in audio_files:
26     for f in os.listdir(audio_files[genre]):
27         csv_df = pd.read_csv(f"{audio_files[genre]}{f}", header = 0)
28         csv_df["genre"] = genre
29         data.append(csv_df)
30
31 df = pd.concat(data, axis = 0, ignore_index=True)
32
33 y = df.pop('genre')
34 X = df
35
36 scaler = MinMaxScaler()
37 X = scaler.fit_transform(X)
38
39 pca = PCA(n_components=2)
40 pca_components = pd.DataFrame(data = pca.fit_transform(X), columns = ["
    principal component 1", "principal component 2"])
41 pca_components = pd.concat([pca_components, y], axis = 1)

```

```

42 plt.figure(figsize = (16, 9))
43 sns.scatterplot(x = "principal component 1", y = "principal component 2",
44               data = pca_components, hue = "genre", alpha = 0.7, s = 50)
45 plt.title("Audio features PCA Decomposition", fontweight = "bold")
46 plt.xticks(fontsize = 14)
47 plt.yticks(fontsize = 14)
48 plt.show()
49
50 correlation_matrix = df.filter(regex = r'\b(?:\w*mfcc\w*)\w+\b')
51 correlation_matrix = correlation_matrix.corr()
52 plt.figure(figsize = (12, 12))
53 sns.heatmap(correlation_matrix, cmap = "bwr", annot=True)
54 plt.title("Correlation matrix for some audio features", fontweight = "bold")
55 plt.show()

```

Plik Test_Classifiers.ipynb

```

1  import random
2  import numpy as np
3  import pandas as pd
4
5  import matplotlib.pyplot as plt
6
7  from sklearn.preprocessing import MinMaxScaler
8  from sklearn.metrics import accuracy_score
9
10 FILES_PER_GENRE = 100
11 TRAIN_SIZE = 0.8
12 TEST_NUMBER = 10
13
14 genres = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', '
15           metal', 'pop', 'reggae', 'rock']
16 audio_files_1sec = {
17     'blues': f"Audio_Features_1sec/blues/",
18     'classical': f"Audio_Features_1sec/classical/",
19     'country': f"Audio_Features_1sec/country/",
20     'disco': f"Audio_Features_1sec/disco/",
21     'hiphop': f"Audio_Features_1sec/hiphop/",
22     'jazz': f"Audio_Features_1sec/jazz/",
23     'metal': f"Audio_Features_1sec/metal/",
24     'pop': f"Audio_Features_1sec/pop/",
25     'reggae': f"Audio_Features_1sec/reggae/",
26     'rock': f"Audio_Features_1sec/rock/"
27 }
28
29 audio_files_3sec = {
30     'blues': f"Audio_Features_3sec/blues/",
31     'classical': f"Audio_Features_3sec/classical/",
32     'country': f"Audio_Features_3sec/country/",
33     'disco': f"Audio_Features_3sec/disco/",
34     'hiphop': f"Audio_Features_3sec/hiphop/",
35     'jazz': f"Audio_Features_3sec/jazz/",
36     'metal': f"Audio_Features_3sec/metal/",
37     'pop': f"Audio_Features_3sec/pop/",
38     'reggae': f"Audio_Features_3sec/reggae/"
39 }

```

```

37         'rock': f"Audio_Features_3sec/rock/"
38     }
39
40     audio_files_5sec = {
41         'blues': f"Audio_Features_5sec/blues/",
42         'classical': f"Audio_Features_5sec/classical/",
43         'country': f"Audio_Features_5sec/country/",
44         'disco': f"Audio_Features_5sec/disco/",
45         'hiphop': f"Audio_Features_5sec/hiphop/",
46         'jazz': f"Audio_Features_5sec/jazz/",
47         'metal': f"Audio_Features_5sec/metal/",
48         'pop': f"Audio_Features_5sec/pop/",
49         'reggae': f"Audio_Features_5sec/reggae/",
50         'rock': f"Audio_Features_5sec/rock/"
51     }
52
53     audio_files_10sec = {
54         'blues': f"Audio_Features_10sec/blues/",
55         'classical': f"Audio_Features_10sec/classical/",
56         'country': f"Audio_Features_10sec/country/",
57         'disco': f"Audio_Features_10sec/disco/",
58         'hiphop': f"Audio_Features_10sec/hiphop/",
59         'jazz': f"Audio_Features_10sec/jazz/",
60         'metal': f"Audio_Features_10sec/metal/",
61         'pop': f"Audio_Features_10sec/pop/",
62         'reggae': f"Audio_Features_10sec/reggae/",
63         'rock': f"Audio_Features_10sec/rock/"
64     }
65
66     audio_files_30sec = {
67         'blues': f"Audio_Features_30sec/blues/",
68         'classical': f"Audio_Features_30sec/classical/",
69         'country': f"Audio_Features_30sec/country/",
70         'disco': f"Audio_Features_30sec/disco/",
71         'hiphop': f"Audio_Features_30sec/hiphop/",
72         'jazz': f"Audio_Features_30sec/jazz/",
73         'metal': f"Audio_Features_30sec/metal/",
74         'pop': f"Audio_Features_30sec/pop/",
75         'reggae': f"Audio_Features_30sec/reggae/",
76         'rock': f"Audio_Features_30sec/rock/"
77     }
78
79     class KNN:
80         def __init__(self, k=3):
81             self.k = k
82
83         def fit(self, X_train, y_train):
84             self.X_train = X_train
85             self.y_train = y_train
86
87         def predict(self, X_test):
88             predictions=[]
89             voters=[]
90             for X in X_test.values:
91                 dst = self.X_train-X

```

```

92         dst = dst * dst
93         dst = dst.sum(axis='columns') ** 0.5
94         dst = pd.DataFrame({'distance': dst,
95                             'value': self.y_train})
96         dst = dst.sort_values('distance')
97         voters = dst.head(self.k).value
98         predictions.append(voters.mode()[0])
99     return predictions
100
101 test_results = {
102     'segment_duration': [],
103     'n_neighbors': [],
104     'accuracy': []
105 }
106
107 for i in range(TEST_NUMBER):
108     print(f"Test nr. {i+1}")
109     train_files = {
110         '1sec': [],
111         '3sec': [],
112         '5sec': [],
113         '10sec': [],
114         '30sec': [],
115     }
116     train_files_genre = []
117     test_files = {
118         '1sec': [],
119         '3sec': [],
120         '5sec': [],
121         '10sec': [],
122         '30sec': [],
123     }
124     test_files_genre = []
125
126     for genre in genres:
127         random_indexes = random.sample(range(FILE_PER_GENRE),
128 FILE_PER_GENRE)
129         train_indexes = random_indexes[:int(TRAIN_SIZE*FILE_PER_GENRE)]
130         test_indexes = random_indexes[int(TRAIN_SIZE*FILE_PER_GENRE):]
131         for n in train_indexes:
132             train_files['1sec'].append(f"{audio_files_1sec[genre]}{genre}{n}
133 }.csv")
134             train_files['3sec'].append(f"{audio_files_3sec[genre]}{genre}{n}
135 }.csv")
136             train_files['5sec'].append(f"{audio_files_5sec[genre]}{genre}{n}
137 }.csv")
138             train_files['10sec'].append(f"{audio_files_10sec[genre]}{genre}
139 }.csv")
140             train_files['30sec'].append(f"{audio_files_30sec[genre]}{genre}
141 }.csv")
142             train_files_genre.append(genre)
143         for n in test_indexes:
144             test_files['1sec'].append(f"{audio_files_1sec[genre]}{genre}{n}
145 }.csv")
146             test_files['3sec'].append(f"{audio_files_3sec[genre]}{genre}{n}

```

```

140     }.csv")
141     test_files['5sec'].append(f"{audio_files_5sec[genre]}{genre}{n}
142     }.csv")
143     test_files['10sec'].append(f"{audio_files_10sec[genre]}{genre}{n}
144     }.csv")
145     test_files['30sec'].append(f"{audio_files_30sec[genre]}{genre}{n}
146     }.csv")
147     test_files_genre.append(genre)
148
149     for segment_duration in (3, 5, 10, 30):
150         X_train_files = train_files[f'{segment_duration}sec']
151         X_test_files = test_files[f'{segment_duration}sec']
152         X_train_data = []
153         for i, f in enumerate(X_train_files):
154             csv_df = pd.read_csv(f, header = 0)
155             csv_df["genre"] = train_files_genre[i]
156             X_train_data.append(csv_df)
157
158         X_train = pd.concat(X_train_data, axis = 0, ignore_index = True)
159         y_train = X_train.pop("genre")
160         scaler = MinMaxScaler()
161         scaler.set_output(transform="pandas")
162         X_train = scaler.fit_transform(X_train)
163
164         X_test = []
165         y_test = test_files_genre
166         for f in X_test_files:
167             X = pd.read_csv(f)
168             X = scaler.transform(X)
169             X_test.append(X)
170
171         for neighbors in range(1, 11):
172             print(f"Predicting for: segment_duration = {segment_duration}sec
173             , n_neighbors = {neighbors}")
174             clf= KNN(neighbors)
175             clf.fit(X_train, y_train)
176
177             y_pred = []
178             for X in X_test:
179                 predictions = clf.predict(X)
180                 possible_genres, counts = np.unique(predictions,
181                 return_counts=True)
182                 genre = possible_genres[counts.argmax()]
183                 y_pred.append(genre)
184
185             test_results['segment_duration'].append(segment_duration)
186             test_results['n_neighbors'].append(neighbors)
187             test_results['accuracy'].append(accuracy_score(y_test, y_pred))
188
189         classifier_test_df = pd.DataFrame(test_results)
190         classifier_test_df.to_csv(f"classifier_test_results.csv", index=
191         False)
192
193     data = pd.read_csv("classifier_test_results.csv")

```



```

188 data_3sec = data[data['segment_duration'] == 3]
189 data_3sec = data_3sec.drop(columns=['segment_duration'])
190 data_3sec = data_3sec.groupby('n_neighbors')['accuracy'].mean()
191
192 data_5sec = data[data['segment_duration'] == 5]
193 data_5sec = data_5sec.drop(columns=['segment_duration'])
194 data_5sec = data_5sec.groupby('n_neighbors')['accuracy'].mean()
195
196 data_10sec = data[data['segment_duration'] == 10]
197 data_10sec = data_10sec.drop(columns=['segment_duration'])
198 data_10sec = data_10sec.groupby('n_neighbors')['accuracy'].mean()
199
200 data_30sec = data[data['segment_duration'] == 30]
201 data_30sec = data_30sec.drop(columns=['segment_duration'])
202 data_30sec = data_30sec.groupby('n_neighbors')['accuracy'].mean()
203
204 fig, ax = plt.subplots(2, 2, figsize=(15, 12))
205 fig.suptitle("Number of neighbors effect on accuracy", fontweight = "bold")
206 ax[0, 0].plot(data_3sec.keys(), data_3sec.values*100, color = "blue")
207 ax[0, 0].set_xticks(data_3sec.keys())
208 ax[0, 0].set_xlabel("n_neighbors")
209 ax[0, 0].set_ylabel("accuracy")
210 ax[0, 0].grid(True)
211 ax[0, 0].title.set_text("3 second segments")
212
213 ax[0, 1].plot(data_5sec.keys(), data_5sec.values*100, color = "red")
214 ax[0, 1].set_xticks(data_5sec.keys())
215 ax[0, 1].set_xlabel("n_neighbors")
216 ax[0, 1].set_ylabel("accuracy")
217 ax[0, 1].grid(True)
218 ax[0, 1].title.set_text("5 second segments")
219
220 ax[1, 0].plot(data_10sec.keys(), data_10sec.values*100, color = "green")
221 ax[1, 0].set_xticks(data_10sec.keys())
222 ax[1, 0].set_xlabel("n_neighbors")
223 ax[1, 0].set_ylabel("accuracy")
224 ax[1, 0].grid(True)
225 ax[1, 0].title.set_text("10 second segments")
226
227 ax[1, 1].plot(data_30sec.keys(), data_30sec.values*100, color = "orange")
228 ax[1, 1].set_xticks(data_30sec.keys())
229 ax[1, 1].set_xlabel("n_neighbors")
230 ax[1, 1].set_ylabel("accuracy")
231 ax[1, 1].grid(True)
232 ax[1, 1].title.set_text("30 second segments")
233
234 plt.show()
235
236 plt.figure(figsize = (12, 8))
237 plt.plot(data_3sec.keys(), data_3sec.values*100, color = "blue", label="3
    sec")
238 plt.plot(data_5sec.keys(), data_5sec.values*100, color = "red", label="5
    sec")
239 plt.plot(data_10sec.keys(), data_10sec.values*100, color = "green", label="
    10 sec")

```

```

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

```

Plik Classify_music.ipynb

```

1 import random
2 import numpy as np
3 import pandas as pd
4
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7
8 from sklearn.preprocessing import MinMaxScaler
9 from sklearn.metrics import accuracy_score, confusion_matrix
10
11 SEGMENT_DURATION = 3 # in sec
12 FILES_PER_GENRE = 100
13 TRAIN_SIZE = 0.8
14
15 audio_files = {
16     'blues': f"Audio_Features_{SEGMENT_DURATION}sec/blues/",
17     'classical': f"Audio_Features_{SEGMENT_DURATION}sec/classical/",
18     'country': f"Audio_Features_{SEGMENT_DURATION}sec/country/",
19     'disco': f"Audio_Features_{SEGMENT_DURATION}sec/disco/",
20     'hiphop': f"Audio_Features_{SEGMENT_DURATION}sec/hiphop/",
21     'jazz': f"Audio_Features_{SEGMENT_DURATION}sec/jazz/",
22     'metal': f"Audio_Features_{SEGMENT_DURATION}sec/metal/",
23     'pop': f"Audio_Features_{SEGMENT_DURATION}sec/pop/",
24     'reggae': f"Audio_Features_{SEGMENT_DURATION}sec/reggae/",
25     'rock': f"Audio_Features_{SEGMENT_DURATION}sec/rock/"
26 }
27
28 train_files = []
29 train_files_genre = []
30 test_files = []
31 test_files_genre = []
32 for genre in audio_files:
33     random_indexes = random.sample(range(FILES_PER_GENRE), FILES_PER_GENRE)
34     train_indexes = random_indexes[:int(TRAIN_SIZE*FILES_PER_GENRE)]
35     test_indexes = random_indexes[int(TRAIN_SIZE*FILES_PER_GENRE):]
36     for n in train_indexes:
37         train_files.append(f"{audio_files[genre]}{genre}{n}.csv")
38         train_files_genre.append(genre)
39     for n in test_indexes:
40         test_files.append(f"{audio_files[genre]}{genre}{n}.csv")
41         test_files_genre.append(genre)
42

```

```

43 class KNN:
44     def __init__(self, k=3):
45         self.k = k
46
47     def fit(self, X_train, y_train):
48         self.X_train = X_train
49         self.y_train = y_train
50
51     def predict(self, X_test):
52         predictions=[]
53         voters=[]
54         for X in X_test.values:
55             dst = self.X_train-X
56             dst = dst * dst
57             dst = dst.sum(axis='columns') ** 0.5
58             dst = pd.DataFrame({'distance': dst,
59                                'value': self.y_train})
60             dst = dst.sort_values('distance')
61             voters = dst.head(self.k).value
62             predictions.append(voters.mode()[0])
63         return predictions
64
65 X_train_data = []
66 for i, f in enumerate(train_files):
67     csv_df = pd.read_csv(f, header = 0)
68     csv_df["genre"] = train_files_genre[i]
69     X_train_data.append(csv_df)
70
71 X_train = pd.concat(X_train_data, axis = 0, ignore_index = True)
72 y_train = X_train.pop("genre")
73 scaler = MinMaxScaler()
74 scaler.set_output(transform="pandas")
75 X_train = scaler.fit_transform(X_train)
76 clf = KNN(1)
77 clf.fit(X_train, y_train)
78
79 y_test = test_files_genre
80 y_pred = []
81 for f in test_files:
82     X_test = pd.read_csv(f)
83     X_test = scaler.transform(X_test)
84     predictions = clf.predict(X_test)
85     possible_genres, counts = np.unique(predictions, return_counts=True)
86     genre = possible_genres[counts.argmax()]
87     y_pred.append(genre)
88
89 print(f"Dokładność: {accuracy_score(y_test, y_pred)*100}%")
90
91 cm = confusion_matrix(y_test, y_pred)
92 fig, ax = plt.subplots(figsize = (10, 10))
93 s = sns.heatmap(cm, annot=True, xticklabels = audio_files.keys(),
94                 yticklabels = audio_files.keys(), ax = ax)
95 s.set_xlabel("Predicted genre", fontsize = 15)
96 s.set_ylabel("True genre", fontsize = 15)
97 plt.title("Music genre classification confusion matrix", fontweight = "

```

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