Klasyfikacja utworów muzycznych do różnych gatunków

Cel projektu

Celem projektu było stworzenie i zaimplementowanie rozwiązania pozwalającego na klasyfikację utworów muzycznych do poszczególnych gatunków muzycznych.

Rozwiązanie

W ramach projektu wykonano program realizujący algorytm *k najbliższych sąsiadow* służący do klasyfikacji, którego poprawnośc testowano na ogólnodostępnym zbiorze danych GTZAN (https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification). W skład wykorzystanego datasetu wchodzą fragmenty utworów o długości 30 sekund, obrazy pokazujące spektrogram mel plików audio oraz 2 pliki w formacie *.csv z wyodrębnionymi cechami poszczególnych utworów o długości 30 s lub podzielonych na 3 sekundowe fragmenty. Utwory podzielone są na 10 różnych gatunków:

- blues
- muzyka klasyczna
- country
- disco
- hiphop
- jazz
- metal
- pop
- reggae
- rock

Do każdego gatunku przypisane jest 100 30-sekundowych utworów w formacie *.wav, które wykorzystano do testowania napisanego programu. Ze wszystkich utworów wyodrębniono parametry mel-cepstralne (*MFCC - Mel-Frequency Cepstral Coefficients*) za pomocą funkcji mfcc z biblioteki python_speech_features, wyniki zapisano w pliku genres_data.dmp. Każdy utwór został podzielony na 20 ms fragmenty, a fragment kodu odpowiedzialny za tę część projektu znajduje się poniżej.

```
(rate, sig) = wav.read(path_to_dataset + "/" + folder + "/" + file)
mfcc_feat = mfcc(sig, rate, winlen=0.02, winstep=0.01, numcep=15, nfft=1200, appendEnergy=False)
covariance = np.cov(np.matrix.transpose(mfcc_feat))
mean_matrix = mfcc_feat.mean(0)
feature = (mean_matrix, covariance, i)
pickle.dump(feature, handle)
```

Na algorytm knn (*k-nearest-neighbours*) składają się 3 funkcje:

a) funkcja obliczająca dystans między dwiema instancjami,
b) funkcja sortująca sąsiadów pod względem dystansu,

```python
def get_neighbours(train_data, considered_point, k):
    distances = []
    for data in train_data:
        dist = check_distance(data, considered_point, k) + check_distance(considered_point, data, k)
        distances.append((data[2], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbours = []
    for neighbour in range(k):
        neighbours.append(distances[neighbour][0])
    return neighbours
```

Parametr train_data to dane treningowe, a k określa liczbę sąsiadów, którzy będą brani pod uwagę. Sprawdzany jest w pętli dystans między wszystkimi punktami / cechami ze zbioru treningowego, wszystkie dystanse są sortowane, a następnie wybierane jest k najbliższych sąsiadów zgodnie z nazwą algorytmu.

c) funkcja zwracająca najbliższego sąsiada

```python
def get_nearest_neighbour(neighbours):
    classes = {}
    for neighbour in neighbours:
        if neighbour in classes:
            classes[neighbour] += 1
        else:
            classes[neighbour] = 1
    # print(classes)
    sorted_classes = sorted(classes.items(), key=operator.itemgetter(1), reverse=True)
    return sorted_classes[0][0]
```

Funkcja opisana w podpunkcie b zwraca k najbliższych sąsiadów, a wszystkie dane z datasetu są odpowiednio oznaczone, czyli mają przypisany gatunek. By odnaleźć najbliższego sąsiada zdefiniowany został zbiór, do którego dodawani są najbliżsi sąsiedzi podzieleni na klasy / gatunki, następnie klasy sortowane są według liczby wystąpień.

Klasyfikacja własnych utworów (w formacie *.wav) odbywa się za pomocą funkcji predict_sample_data(), której budowę przedstawiono poniżej. Podobnie jak przy ekstraktowaniu danych z datasetu, również i tu wyodrębnia się parametry mel-cepstralne badanego utworu i porównuje się z parametrami ze zbioru danych treningowych, na podstawie którego odbywa się klasyfikacja.

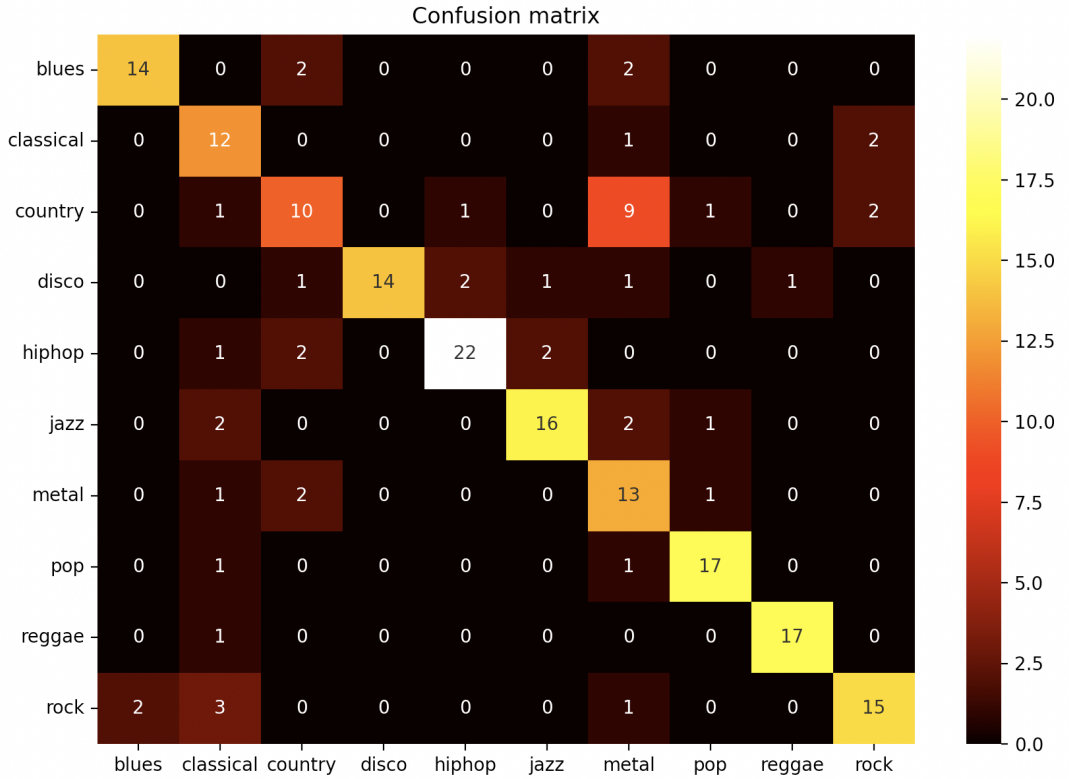
```python
def predict_sample_data(file, genres_names):
    dataset = load_dataset("./genres_data.dmp")
    train_dataset, test_dataset = train_test_split(dataset, test_size=1, random_state=58)
    (rate, sig) = wav.read(file)
    mfcc_features = mfcc(sig, rate, winlen=0.02, winstep=0.01, numcep=15, nfft=1200, appendEnergy=False)
    covariance = np.cov(np.matrix.transpose(mfcc_features))
    mean_matrix = np.round(mfcc_features.mean(axis=0), 3)
    feature = (mean_matrix, covariance, 0)
    pred = get_nearest_neighbour(get_neighbours(train_dataset, feature, 5))
    return genres_names[pred]
```

Wyniki i wnioski

Uzyskane rezultaty okazały się obiecujące: na datasecie GTZAN algorytm osiągnął skuteczność ok. 75 %, przy liczbie sąsiadów równej 5.

```
F1 score is: 0.755478
Precision score is: 0.782946
Average accuracy: 75.00%
```

Tablica pomyłek zbioru testowego (wyodrębnionego z datasetu GTZAN) prezentuje się następująco:



Z widocznej tablicy można wyciągnąć wnioski, że wykonany algorytm działa poprawnie, w zdecydowanej wiekszości przypadków klasyfikacja przebiegła pomyślnie. Widoczną anomalią jest natomiast dość częste klasyfikowanie metalu jako country, co może być związane z przenikaniem się gatunków muzycznych.

Testy na przykładowych utworach niepochodzących z datasetu potwierdziły ten wynik, algorytm działał dobrze w zdecydowanej większości przypadków jednak nie był nieomylny. Często muzykę klasyczną określał jako rock, natomiast jazz jako pop, jest to jednak zrozumiałe, ponieważ utwory posiadają cechy różnych gatunków.