PyMeno

Programowanie w języku Python

Aplikacje, 7

Autorzy: *Klaudia Olejniczak, Oleksandr Kuzhel*

Spis treści

Wstęp

Opis ogólny tematu :

Aplikacja która dzięki przeszukaniu biblioteki muzycznej użytkownika odnajduje podobne do nich i wyświetla je użytkownikowi wraz z możliwością otworzenia odnośnika bezpośrednio do ich utworów na YouTube. Wyszukiwanie podobieństwa opiera się na analizie tekstów utworów.

Informacje wstępne:

Projekt PyMeno realizowany jest w ramach przedmiotu Programowanie w języku Python. Implementacja odbyła się w języku Python 3.4 . Dokumentacja ta ma na celu opisać program, by ułatwić użytkowanie oraz zapoznanie się z kodem.

Wymagane dodatkowe biblioteki :

W przypisach linki do bibliotek lub komendy do instalowania z cmd.

PyLyrics[1], nltk[2], mutagen[3], Youtube API[4]

1 Komenda : pip install PyLyrics **(**W razie problemów : https://pypi.python.org/pypi/PyLyrics/)
2 Strona : https://pypi.python.org/pypi/nltk
3 Strona : https://pypi.python.org/pypi/mutagen
4 Komenda : pip install --upgrade google-api-python-client (W razie problemów : https://developers.google.com/api-client-library/python/start/installation)

Opis aplikacji :

Projekt składa się z elementów zajmujących się tworzeniem biblioteki muzycznej do porównywania posiadanych przez użytkownika utworów, analizy biblioteki użytkownika, wyszukiwania podobieństw oraz GUI.

Funkcjonalność :

Nowe źródła porównań

W aplikacji mamy możliwość uaktualnienia bazy porównań. (Data → Download artists list) Dzięki temu można uaktualnić swoją listę scrobble.xml, względem toplisty z Last.fm.

Tworzenie i poszerzanie biblioteki utworów do porównań

Bibliotekę można tworzyć od zera lub kontynuować jej rozbudowywanie. Poprzez (*Data → Parse artists information to database), gdzie podaje się od którego do którego artysty chce się ściągnąć.* Minimalna liczba to 0, maksymalna 1000. W wersji użytkowej gromadzenie danych nie powinno być dostępne dla użytkownika, ale nie mamy dostępu do takiej bazy.

Wyszukiwanie podobnych utworów

Wyszukiwanie podobnych utworów dzieli się na trzy etapy, niezależnie od algorytmu, który się wybierze (dostępne są 3).

1. Przeszukiwanie biblioteki użytkownika i przetwarzanie tekstów
2. Zastosowanie algorytmu i wyszukanie podobieństw.
    - Algorytm I

Na podstawie wektorów dla każdego artysty w bazie i wektora wszystkich słów w naszej bibliotece wybieramy najbliższe 1 wyniki, po czym porównujemy je ze względu na średnią ilość słów na utwór. (Naszym zdaniem istnieje tutaj zależność względem tempa utworu, im więcej, tym dynamiczniejsza). Po czym wyszukuje się ponownie za pomocą miary podobieństw wektorów, tylko na bazie artysta, album ,z zachowaniem najlepszych wyników z poprzedniej eliminacji.

- Algorytm II

Na podstawie słowników z bibliotek i zbioru słów powstaje przecięcie pomiędzy nimi. Wybieramy te zbiory które są największe, ograniczamy poprzez ilość słów(analogicznie jak w alg I) i na wyniku wykorzystujemy podobieństwo wektorów z wykorzystaniem artysta,album.

- Algorytm III

Ten algorytm w stosunku poprzednich nie opiera się na wspólnym słowniku dla całej biblioteki użytkownika, tylko za pomocą podobieństwa wektorów wyszukuje najbardziej podobne artysty do każdego znalezionego wykonawcy. Po czym opiera się o kryterium ze średnią słów w utworze o powiększonym lekko zakresie niż średnia wynikająca z obliczeń.

3. Wylosowanie utworu z wybranego autora oraz albumu.

Odsyłanie do utworu na serwisie YouTube

Po dwukrotnym kliknięciu na zespół : utwór na liście zaproponowanych użytkownikowi zostanie otworzona nowa zakładka w przeglądarce wraz z utworem, który został mu zaproponowany.

Oprowadzka po kodzie

Wstęp

W projekcie klasy pogrupowane są w 2 moduły :

- GUI_DIR
- CORE

oraz katalog z plikami pickle i plikiem scrobble :

- DATA

Moduł CORE :

Ten moduł zawiera wszystkie funkcjonalności naszej aplikacji.

Klasa CreatingDatabase

Opis :

Ta klasa odpowiada za cały proces tworzenia bazy danych do porównań . Biblioteka ta jest zapisywana raz, dla skrócenia czasu przeszukiwania. (Sparsowanie 500 wykonawców zajęło nam łącznie 10h)

Do stworzenia listy autorów użyliśmy udostępnione przez last.fm API, do pobrania albumów i tekstów piosenek biblioteki PyLyrics . Zaś do zapisania ich użyliśmy pickle. Na bibliotekę łącznie składa się 6 plików(pickle200, pickle303, pickle500 zawierają ten sam rodzaj danych, ale z faktu czasu jaką zajmuje ściąganie tekstów rozłożyliśmy to na 3, żeby uniknąć utraty już wcześniej sparsowanych danych) .

Zbiory słów:

PickleLil200, pickleLil303, pickleLil500 – dane [autor,album]

= Counter({ word : amount })

PickleLilEvery.p – dane[autor] = Counter({ word : amount })

Zbiory średniej ilości słów na utwór :

PickleLilWordPerSong.p – dane[autor] = średnia

PickleLilFromArtistPerSong.p - dane[autor,album] = średnia

W trakcie rozkładu piosenek na zbiory słów wykorzystana została jeszcze biblioteka ntlk (Natural Language Tool Kit), dzięki której usunięto wszystko stopwords'y oraz ujednolicono formę słów (np. running → run). Rozkład tekstu powstał na bazie bag_of_words.

Metody :

__init__() – inicjalizuje słowniki wykorzystywane w tworzeniu bazy

download_list_of_artists() – pobiera najnowszą top listę z Last.fm

do_the_dicts(artist, name) – funkcja która ściąga tekst utworu i ujednolica go i tworzy z niego słownik

__log_info() - funkcja która wypisuje aktualne średnie

dict_per_album(label, song) – funkcja która wstawia nowy Counter utworu do słownika jeśli go w nim nie ma, jeśli jest to uzupełnia.

dict_for_artist(current_artist, song) – funkcja robi to samo co poprzednia tylko w ramach innego dictonary

put_into_pickles(number_of) – funkcja ta odpowiada za zapisywanie słowników do plików pickle

parse_file(number_of, number_from) – funkcja ta jest główną w tej klasie, odpowiada za całą prace, czyli tworzenie biblioteki od odpowiedniego numeru artysty do innego i za wypisywanie ewentualnych błędów.

Rzucane we wnętrzu wyjątki są spowodowane głównie brakiem jakiegoś artysty, albumu, piosenek w bazie PyLyrics.

Opis działania programu pod czas wyszukiwania nowych piosenek

Po tym jak użytkownik wybierze algorytm(na przykład ALG 1) wywołuje się metoda new_thread2() pokazująca użytkownikowi okienko dialogowe i proponująca wybrać folder z muzyką. Po wybraniu folderu tworzy nowe okienko App() i blokuje Menu disable_menu() po czym tworzy nowy wątek który będzie wyszukiwał dane. W nowym wątku przechodzi po każdym pliku znajdującym się w folderze wybranym przez użytkownika i jeśli to plik audio pobiera z niego informację o utworze za pomocą ID3, po czym pobiera tekst utworu i usuwając stopwords i lemmatyzując pozostałe słowa tworzy bag of words.

Użytkownik może śledzić cały proces za pomocą nowego okienka w którym widzi dane opracowywane w tej chwili przez program i pasek postępu szacujący ile czasu jeszcze zostało do

otrzymania wyników. Po czym za pomocą algorytmu opisanego wyżej dane z biblioteki użytkownika porównują się z danymi z biblioteki programu po czym 15 najbardziej pasujących wyników zostaje wyświetlonych w lewym listboxie, a okienko pokazujące postęp znika.

Klasa MakeBagOfWords

Opis:

Ta klasa odpowiada za przeszukiwanie biblioteki użytkownika przy pomocy biblioteki mutagen pozwalającej szybko i bezbłędnie otrzymać informację o każdym utworze. Następnie analogicznie jak przy tworzeniu bazy, z różnicą w przechowaniu słownika, w tym wypadku jako zmiennych w klasie.

Zmienne :

LIST_ARTIST_SONGS – lista mająca zapobiec parsowaniu wielokrotnie tych samych utworów.

MY_BAG – słownik przechowujący słowa występujące u każdego artysty i ich ilość

MY_BAG_C – słownik przechowujący ilość piosenek na każdego autora w bibliotece użytkownika

Metody:

__init__() - zmienne inicjalizujaca zbiory

change_title(path_to_file) – wywoływana przy znalezieniu każdego utworu, ona dba aby utwory nie były jeszcze raz ściąganie, oraz odpowiada za złapanie wyjątku, gdy plik ma nieodpowiednie właściwości.

bag_of_words(artist_name, song_name) – metoda odpowiedzialna za rozkład tekstu utworu i wpisaniu informacji do słowników

clear_bag_of_words() - metoda która czyści wszystkie zmienne w klasie aby nie wprowadzały błędu gdy user chce sprawdzić inną bibliotekę

check_if_refresh() - metoda ta sprawdza czy ostatnia ścieżka przeszukiwania biblioteki jest taka sama jak ostatnio, jeśli nie to czysci słowniki i wpisuje nową ścieżka

Klasa FindMusic

Klasa ta służy do przetwarzania danych z biblioteki i ze zbioru do porównań, na celu ma zwrócić listę autor,album z tymi które na podstawie tekstów są najbardziej podobne

Zmienne:

Klasa ta korzysta ze zmiennych przekazanych z poprzedniej klasy. Oraz słowniki

Metody:

__init__() - inicjalizacja zmiennych słowników oraz przypisanie wartości podanych z zewnątrz

search_for_similar_ver_1()

search_for_similar_ver_2() - Te dwie funkcje zawierają kroki kolejnego algorytmu. Dla uproszczenia kodu wszystkie metody korzystają z praktycznie tych samych metod, poza tymi w których aktualny algorytm się różnice, dzięki temu oszczędzone jest bardzo dużo niepotrzebnego kodu.

search_for_similar_ver_3() - ta metoda się różni od poprzednich bo przeskakuje część z nich.

made_group_smaller() - ta funkcja oblicza średnie słów na piosenkę u artystów z biblioteki użytkownika, oraz wyznacza minimalna i maksymalną wartość.

first_step() - wczytuje dane z pickle LileEvery.

second_step_ver1(min_max, my_bag_all) – metoda która robi część wspólną na dwóch zbiorach słowników. Te tych które mają największą wartość wyrażeń wspólnych zostają sprawdzone pod względem średniej ilości słów na piosenkę. Ta metoda zwracali listę artystów

second_step_ver2(min_max, my_bag_all) - wersja druga używa podobieństwa wektorów zamiast części wspólnej zbiorów.

second_step_ver3(min_max) – sprawdza podobieństwo wektorów pomiędzy każdym artystą z biblioteki użytkownika, a albumami znajdującymi się w pickle

fourth_step(list_chosen, my_bag_all) – sprawdza podobieństwo pomiędzy wektorami ze słowników artystów,albumów a wspólnym słownikiem z całej biblioteki użytkownika.

fifth_step(list_of_keys, queues) – ta metoda losuje utwory z wybranych albumów i przesyła informacje do wątku głównego.

similarity(vector1, vector2) – liczy podobieństwo pomiędzy dwoma wektorami

Moduł GUI_DIR :

- Klasa gui – odpowiada za główne okno aplikacja, to ona posiada metodę odpowiadającą za wyszukiwanie utworów poprze youtube API po podwójny kliknięci na tytuł w listboxie
- Klasa App – odpowiada za okienko podczas parsowania

GUI

Interfejs został stworzony przy pomocy Tkintera i składa się z dwóch listboxów w których pokazujemy utwory muzyczne znalezione na komputerze użytkownika i menu pozwalającego na

wybór algorytmu, folderu z muzyką i ogólnego sterowania programem. Pod czas parsowania danych pojawia się nowe okienko pokazujące użytkownikowi cały proces i szacujące czas do końca parsowania.

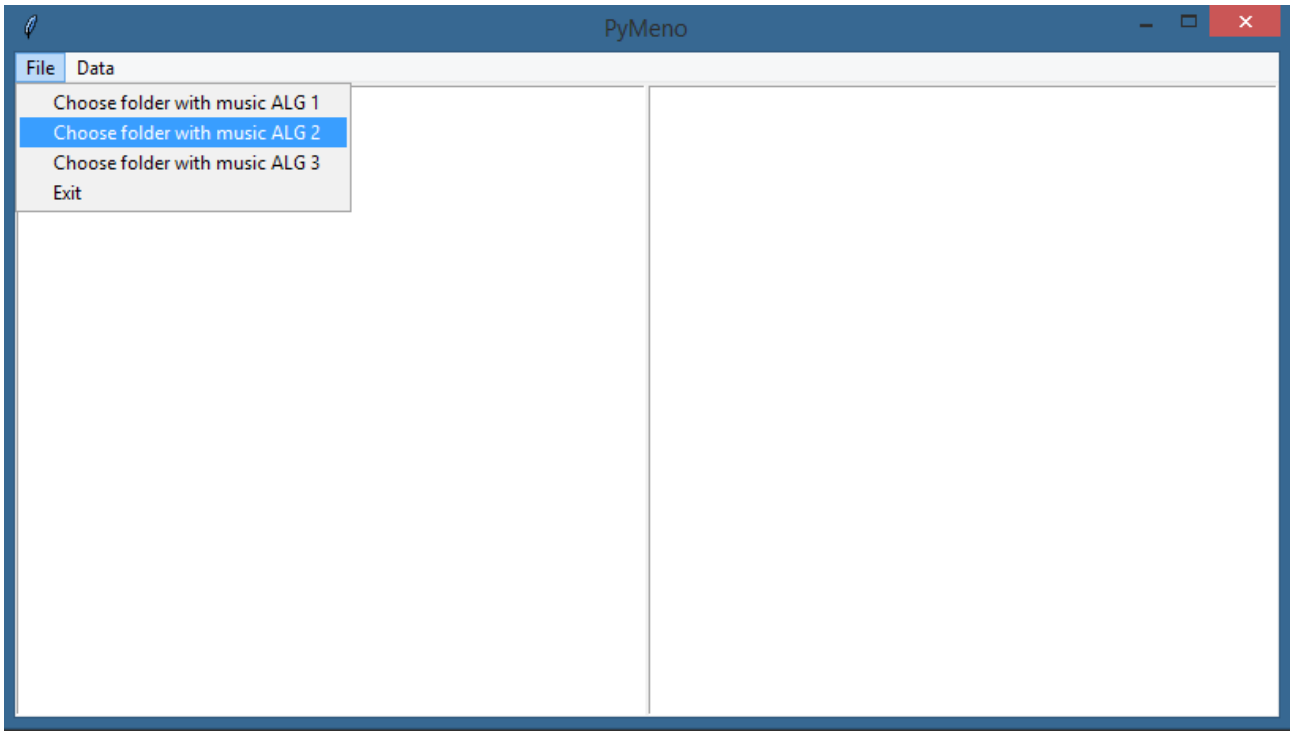
Linki do wybranych utworów.

Wybrane utwory pojawiają się w lewym listboxie, po podwójnym kliknięciu na utworze wyszukujemy na youtubie najbardziej oglądany filmik dotyczące tego utworu i otwieramy go w przeglądarce.

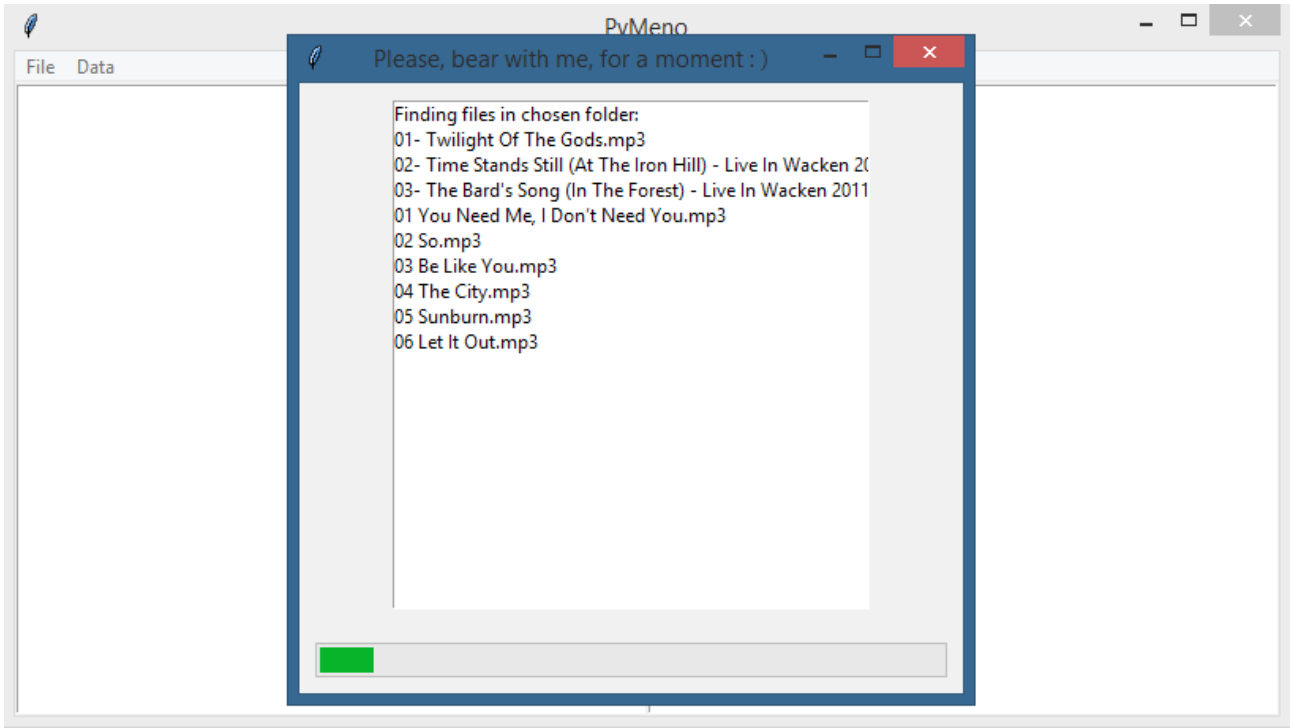
Zrzuty Ekranu:

Aby przedstawić interfejs apliakcji :

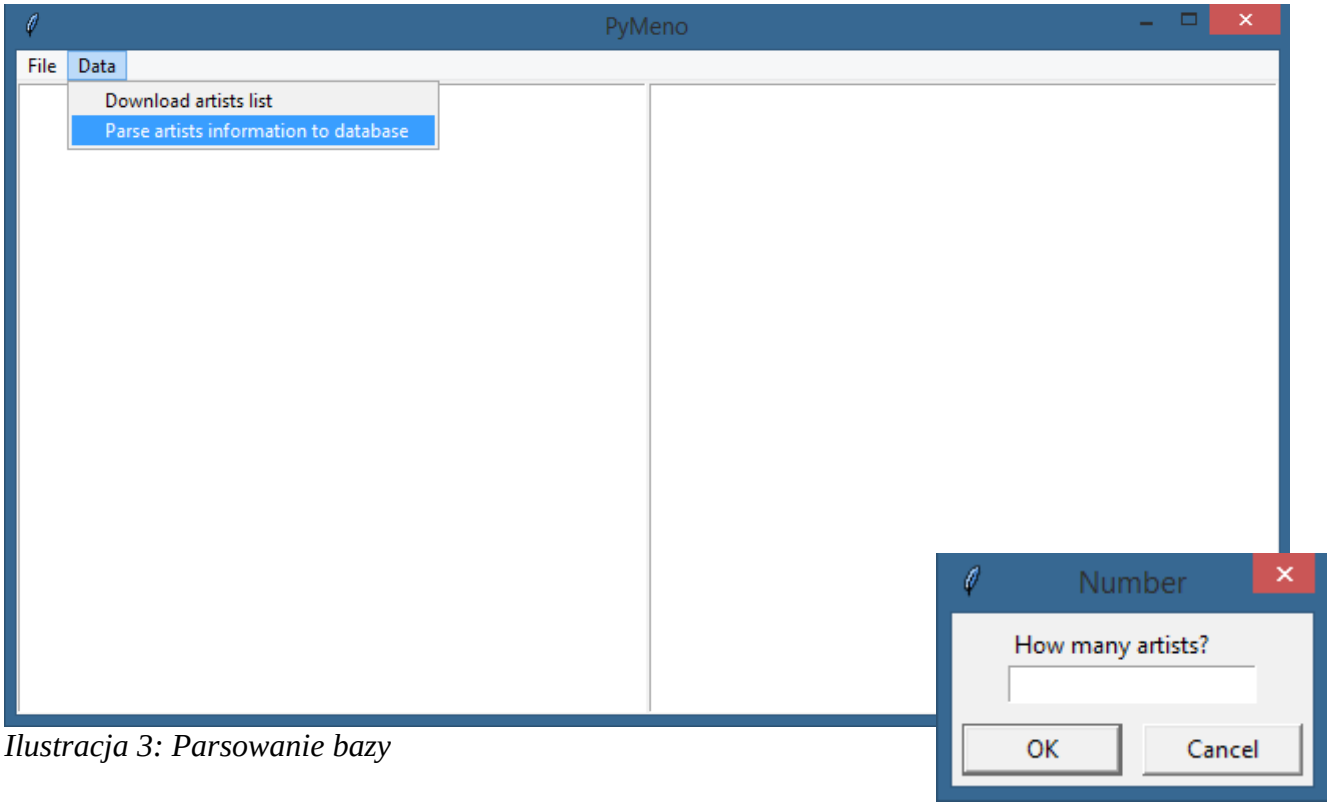
Na ekranie głównym użytkownik może m.in. :



Ilustracja 1: Wybór algorytmu



Ilustracja 2: Wyszukiwanie podobnych utworów



Ilustracja 3: Parsowanie bazy