Progetto - Spotify API

Progetto di Francesco Petrosino X81000533

Introduzione

Questo programma, grazie alle informazioni che è possibile ricavare dalle API REST di Spotify, permette di creare una playlist con album e canzoni di autori preferiti dagli utenti che la compongono, arricchendola tramite opportuni algoritmi ed infinte verranno aggiunte alcune delle canzoni più in voga del momento selezionate dagli autori suggeriti dal programma.

Prima di iniziare, è opportuno precisare che tutte le informazioni, dati e quant'altro ricavato e/o elaborato dal programma stesso, NON ottiene (o conserva) in alcun modo tutti quei dati e informazioni che Spotify ritiene private per l'utente (tranne pre quelle di chi fornisce la coppia di chiavi d'accesso per le API)

Come funziona

Tramite la sottoscrizione all'account Spotify developer e con l'utilizzo di opportune librerie, in Python è possibile realizzare moltissimi tool come il progetto che segue..

Questo progetto inizialmente, richiede l'inserimento di una lista di utenti, iscritti semplicemente alla piattaforma e che abbiano - non necessariamente - creato già delle loro playlist pubbliche.
Dopo aver definito il gruppo di utenti, il tool procederà alla raccolta dei dati, memorizzando informazioni come: nome utente, nomi delle playlist, le relative canzoni inserite, e molto altro.
Successivamente, durante la fase dell'analisi dei dati, grazie al supporto di alcune librerie, come pandas e numpy, e all'implementazione di un algoritmo di uno dei metodi di sistemi di raccomandazione dei dati, verranno processate tutte le informazioni raccolte nel precedente step.
Infine, verrà creata una playlist con una parte dei dati elaborati dall'algoritmo, e per rendere più creativa e unica la playlist, tramite le informazioni in possesso di generi e autori, verrà effettuata una ricerca sulla piattaforma stessa di alcune delle canzoni più in voga del momento, per inserirle dunque sulla playlist.

Raccolta dei dati

Setup

Esegui questo comando nel tuo terminale:

- `py -m pip install --upgrade pip`
- `py -m pip install --upgrade -r .\requirements.txt`

In [ ]:
```
# Import delle librerie

import time
import pandas
import numpy
import numpy as np
import sklearn
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics.pairwise import cosine_similarity
import json
import spotipy
from operator import length_hint
from datetime import datetime

from spotipy.oauth2 import SpotifyOAuth
from user import User
from group import Group

import main
```

In [ ]:
```
# Variabili globali

# client_id = "YOUR_CLIENT_ID"
# client_secret = "YOUR_CLIENT_SECRET"
# redirect_uri = "YOUR_REDIRECT_URI"
client_id = "b48898f561ab4533b7396439e6610ff4"
client_secret = "bbf061893e8f4ff8924eed41c39c6a90"
redirect_uri = "http://localhost:8080"

# Il token servirà per..
scope = [
    "user-library-read",
    "user-top-read",
    "user-follow-read",
    "user-read-private",
    "user-read-recently-played",
    "user-read-playback-state",
    "user-modify-playback-state",
    "user-read-playback-position",
    "streaming",
    "app-remote-control",
    "playlist-read-private",
    "playlist-modify-private",
    "playlist-modify-public",
    "playlist-read-collaborative"
]

# Autenticazione

auth_manager = SpotifyOAuth(
    client_id=client_id,
    client_secret=client_secret,
    redirect_uri=redirect_uri,
    scope=scope
)

sp = spotipy.Spotify(auth_manager=auth_manager)
```

Una volta ottenuto il token per poter richiamare le API di Spotify, sarà possibile iniziare la spiegazione e la raccolta dei dati, per il conseguimento del progetto.

Creazione del gruppo di utenti

Nello step successivo, viene illustrato come verranno raccolte le informazioni, relative agli utenti e le relative playlist.
Sono state definite due classi di comodo che ci consentiranno di semplificare la lettura e l'analisi dei dati.
Queste classi sono: **User** e **Group**

User class

```python
class User(object):

    def __init__(self, user):
        self.display_name = user['display_name']
        self.user_id = user['id']
        self.following_playlists = []

    # Stampa il nome dell'utente e il relativo id
    def print(self):
        print("L'username di {:s} è {:s}".format(self.display_name, self.user_id))

    # Imposta delle playlist all'utente
    def set_following_playlists(self, playlists):
        json_p = json.dumps(playlists)
        playlist_json = json.loads(json_p)
        public_playlist_json = filter(lambda p: p.get('public') == True, playlist_json)
        self.following_playlists = list(public_playlist_json)

    # Ritorna le playlist dell'utente
    def get_following_playlists(self):
        return self.following_playlists
```

Group class

```python
class Group(object):

    def __init__(self):
        self.list = {}
        self.playlist_dataframe = pandas.DataFrame(columns=['user_id', 'user_display_name', 'playlist_id', 'playlist_name', 'playlist_description', 'playlist_uri', 'playlist_public', 'playlist_total_tracks'])
        self.playlist_tracks_dataframe = pandas.DataFrame(columns=['user_id', 'user_display_name', 'playlist_id', 'playlist_name', 'album_id', 'album_name', 'album_release_date', 'album_total_tracks', 'artist_id',
                                                                   'artist_name', 'track_disc_number', 'track_duration_ms', 'track_explicit', 'track_id', 'track_name', 'track_popularity'])
        self.artists_top_tracks = []
        self.albums_top_tracks = []
        self.top_tracks = []
        self.recommended_playlist = []

    # Aggiunge un nuovo user al gruppo
    def append(self, user: User):
        self.list[len(self.list)] = user

    # Ritorna il numero di utenti presenti in lista
    def count(self) -> int:
        return len(self.list.items())

    # Stampa le informazioni degli utenti del gruppo
    def print(self):
        print('Il gruppo è composto da: ')
        for user in self.list.values():
            print(user.display_name)

    # Ritorna l'utente del gruppo tramite indice
    def get_from_list(self, index: int) -> User:
        return self.list[index]

    def __by_display_name(self, user: User, display_name: str):
        return user.display_name == display_name

    # Ritorna un uitente tramite display_name
    def find(self, display_name: str) -> User:
        res = list(filter(lambda user: self.__by_display_name(user, display_name), self.list.values()))
        if len(res) > 0:
            return res[0]
        else:
            return None

    # Concateno con il dataframe di tutte le playlists del gruppo
    def append_to_playlist_dataframe(self, dataframe):
        self.playlist_dataframe = pandas.concat([self.playlist_dataframe, dataframe], ignore_index=True)

    # Ritorna il dataframe delle playlist del gruppo
    def get_playlist_dataframe(self):
        return self.playlist_dataframe
```

```python
    # Concateno con il dataframe di tutte le tracks delle playlists del gruppo
    def append_to_playlist_tracks_dataframe(self, dataframe):
        self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe], ignore_index=True)

    # Ritorna il dataframe di tutte le tracks del gruppo
    def get_playlist_tracks_dataframe(self):
        return self.playlist_tracks_dataframe

    # Aggiunge una colonna al dataframe con la funzione di aggregazione sum
    def __append_sum_column(self, pivot_table: pandas.DataFrame) -> pandas.DataFrame:
        pivot_table = pivot_table.assign(Sum=pivot_table.sum(axis=1))
        return pivot_table.sort_values(by='Sum', ascending=False, inplace=False)

    # Crea una tabella pivot con il numero di volte in cui compare una track nelle playlist di un utente
    def create_tracks_pivot_table(self):
        self.tracks_pivot_table = pandas.pivot_table(self.playlist_tracks_dataframe, index=['track_id'], columns=['user_id'], values=['playlist_id'], aggfunc=pandas.Series.count, fill_value=0.0)
        self.tracks_pivot_table = self.__append_sum_column(self.tracks_pivot_table)

    # Crea una tabella pivot con il numero di volte in cui compare un album nelle tracks di un utente
    def create_albums_pivot_table(self):
        self.albums_pivot_table = pandas.pivot_table(self.playlist_tracks_dataframe, index=['album_id'], columns=['user_id'], values=['track_id'], aggfunc=pandas.Series.count, fill_value=0.0)
        self.albums_pivot_table = self.__append_sum_column(self.albums_pivot_table)

    # Crea una tabella pivot con il numero di volte in cui compare un artista nelle tracks di un utente
    def create_artists_pivot_table(self):
        self.artists_pivot_table = pandas.pivot_table(self.playlist_tracks_dataframe, index=['artist_id'], columns=['user_id'], values=['track_id'], aggfunc=pandas.Series.count, fill_value=0.0)
        self.artists_pivot_table = self.__append_sum_column(self.artists_pivot_table)

    # Ritorna la tabella pivot
    def get_tracks_pivot_table(self) -> pandas.DataFrame:
        return self.tracks_pivot_table

    # Ritorna la tabella pivot
    def get_albums_pivot_table(self) -> pandas.DataFrame:
        return self.albums_pivot_table

    # Ritorna la tabella pivot
    def get_artists_pivot_table(self) -> pandas.DataFrame:
        return self.artists_pivot_table

    # Aggiunge le tracks nella lista delle top tracks di un artista
    def append_to_artists_top_tracks(self, list):
        for item in list:
            self.artists_top_tracks.append(item)
```

```
    # Aggiunge le tracks nella lista delle top tracks di un album
    def append_to_albums_top_tracks(self, list):
        for item in list:
            self.albums_top_tracks.append(item)

    # Aggiunge la track nella lista delle top tracks generale
    def append_to_top_tracks(self, item):
        self.top_tracks.append(item)

    # Ritorna la lista delle top tracks di un artista
    def get_artists_top_tracks(self) -> list:
        return self.artists_top_tracks

    # Ritorna la lista delle top tracks di un album
    def get_albums_top_tracks(self) -> list:
        return self.albums_top_tracks

    # Ritorna la lista delle top tracks generale
    def get_top_tracks(self) -> list:
        return self.top_tracks

    # Aggiunge una track nella recommended playlist
    def append_to_recommend_playlist(self, item):
        for obj in self.recommended_playlist:
            if obj['track_id'] == item['track_id']:
                return
        self.recommended_playlist.append(item)

    # Ritorna la recommended playlist
    def get_recommended_playlist(self) -> list:
        return self.recommended_playlist
```

In [ ]:
```
print("Creo il gruppo di utenti...")

# Istanza del gruppo di utenti
group = Group()

# Lista di username dei vari utenti
id_list = [
    '21ktslj6lkgx6wswmfwu5cwuq',
    'rcin690qmgjd32d088agw6xyr',
    '1171706601',
    '6ua7zipvurza24psy4h2rvumy',
    '21vfi35alpob2afo2vtoybthi',
    '392q67hc38jnsgjg2hqrjyuhc',
    '11135404841'
]

len = len(id_list)

# Per ogni username in lista, prendo le loro informazioni
# e le inserisco nel gruppo
for i, id in enumerate(id_list):
    # Cerco su Spotify le informazioni dell'utente
    result = sp.user(id)

    # Aggiungo l'utente al gruppo
    user = User(result)
    group.append(user)

    if i+1 < len:
        time.sleep(3)

group.print()
```

```
Creo il gruppo di utenti...
Il gruppo è composto da:
Francesco Petrosino
Ale
Philip Giovanni Emmer
Renato
Elio Pampolini
Scalòppina
Aldo Fiorito
```

Recupero le playlist e le relative tracks

Di seguito andremo a collezionare tutte le playlist e le relative tracks, tenendo traccia solamente delle informazioni più importanti, che ci consentiranno di analizzare al meglio le informazioni nello step successivo tramite un algoritmo basato su uno dei sistemi di raccomandazione

In [ ]:
```
print("Recupero le playlists dal server...")

len = group.count()

# Per ogni utente del gruppo
# colleziono le sue playlists pubbliche
for i in range(len):
    # Prendo l'utente corrente
    user = group.get_from_list(i)

    print(user.display_name)

    # Cerco la playlist dell'utente
    results = sp.user_playlists(user=user.user_id, limit=10)

    # Colleziono la playlist dell'utente
    user.set_following_playlists(results['items'])

    if i+1 < len:
        time.sleep(5)
```

```
Recupero le playlists dal server...
Francesco Petrosino
Ale
Philip Giovanni Emmer
Renato
Elio Pampolini
Scalòppina
Aldo Fiorito
```

In [ ]:
```
print("Creo un dataframe con le playlists...")

len = group.count()

for i in range(len):
    # Prendo l'utente corrente
    user = group.get_from_list(i)

    # Creo una lista temporanea delle playlist dell'utente corrente
    playlist_list = list(map(lambda playlist: {
        'user_id': user.user_id,
        'user_display_name': user.display_name,
        'playlist_id': playlist.get('id'),
        'playlist_name': playlist.get('name'),
        'playlist_description': playlist.get('description'),
        'playlist_uri': playlist.get('uri'),
        'playlist_public': playlist.get('public'),
        'playlist_total_tracks': playlist.get('tracks', {}).get('total')
    }, user.get_following_playlists()))

    # Creo un dataframe con la playlist dell'utente corrente
    playlist_user_dataframe = pandas.DataFrame(playlist_list)

    # Concateno con il dataframe finale
    group.append_to_playlist_dataframe(playlist_user_dataframe)
```

```
Creo un dataframe con le playlists...
```

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:49: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_dataframe = pandas.concat([self.playlist_dataframe, dataframe], ignore_index=
True)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:49: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_dataframe = pandas.concat([self.playlist_dataframe, dataframe], ignore_index=
True)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:49: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_dataframe = pandas.concat([self.playlist_dataframe, dataframe], ignore_index=
True)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:49: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_dataframe = pandas.concat([self.playlist_dataframe, dataframe], ignore_index=
True)
```

In [ ]:

```python
print('Per ogni playlist cerco le relative tracks')

# Per ogni playlist collezionata, cerco le relative tracks
for index, playlist in group.get_playlist_dataframe().iterrows():
    print(playlist.get('playlist_name'))

    # Prendo le tracks in una playlist
    results = sp.playlist_items(playlist_id=playlist.get('playlist_id'), limit=100)

    # Converto la risposta in un json
    items_raw = json.dumps(results['items'])
    items_json = json.loads(items_raw)
    items_list = list(items_json)

    # Creo una lista temporanea delle playlist dell'utente corrente
    playlist_tracks_list = list(map(lambda track: {
        'user_id': playlist.get('user_id'),
        'user_display_name': playlist.get('user_display_name'),
        'playlist_id': playlist.get('playlist_id'),
        'playlist_name': playlist.get('playlist_name'),
        'album_id': track.get('track', {}).get('album', {}).get('id'),
        'album_name': track.get('track', {}).get('album', {}).get('name'),
        'album_release_date': track.get('track', {}).get('album', {}).get('release_date'),
        'album_total_tracks': track.get('track', {}).get('album', {}).get('total_tracks'),
        'artist_id': track.get('track', {}).get('artists', {})[0].get('id'),
        'artist_name': track.get('track', {}).get('artists', {})[0].get('name'),
        'track_disc_number': track.get('track', {}).get('disc_number'),
        'track_duration_ms': track.get('track', {}).get('duration_ms'),
        'track_explicit': track.get('track', {}).get('explicit'),
        'track_id': track.get('track', {}).get('id'),
        'track_name': track.get('track', {}).get('name'),
        'track_popularity': track.get('track', {}).get('popularity')
    }, items_list))

    # Creo un dataframe con la playlist dell'utente corrente
    playlist_tracks_dataframe = pandas.DataFrame(playlist_tracks_list)

    # Concateno con il dataframe finale
    group.append_to_playlist_tracks_dataframe(playlist_tracks_dataframe)
```

```
Per ogni playlist cerco le relative tracks
Recommend playlist - 2023-02-12
Car
```

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Guitar
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
2021
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
- MUSIC (R)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
2020 Look up
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
La mia playlist n. 9
La mia playlist n. 8
La mia playlist n. 7
el latino
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Artemis
Macchina
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Indie pier
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Allenamento
```

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
```

Alan

thnx workout

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
```

Travel

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
```

Pump

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
```

In(die)

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
```

Mix

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
```

Fun time

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
```

🎹🎹🎹🎹

D Mood

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
```

Aggiunto ai preferiti

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Febbraio23
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Novembre 2022
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
OTTOBRE 2022
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Time
LUGLIO
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Estate 2022
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
ROCK
Dance90
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
2022
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
New playlist
```

```
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Rock/Metal
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Eminem
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Pistol Whipping 'n' Hoes
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
[G]old
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Chill and Memes
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
Acoustic Mood
💊
c:\Users\franc\Documents\Università\ProgettoSMM\Progetto2\group.py:57: FutureWarning: In a fu
ture version, object-dtype columns with all-bool values will not be included in reductions wi
th bool_only=True. Explicitly cast to bool dtype instead.
  self.playlist_tracks_dataframe = pandas.concat([self.playlist_tracks_dataframe, dataframe],
ignore_index=True)
```

Analisi dei dati

Andremo ad analizzare tutti i dati raccolti precedentemente, tramite un algoritmo basato sul metodo **Content-based filtering** come sistema di raccomandazione, per creare infine una playlist che accomuna le preferenze degli utenti in lista.

Content-based filtering

Il **Content-based filtering** è un metodo del sistema di raccomandazione per formulare raccomandazioni agli utenti in base alle caratteristiche o agli attributi degli elementi a cui gli utenti hanno mostrato

interesse. Il sistema utilizza questi attributi per identificare elementi simili e quindi formula raccomandazioni agli utenti in base alla somiglianza.

L'idea generale alla base di questo metodo è che se a un utente piace un elemento, gli piaceranno anche elementi simili, messi a fattor comune da un dataset globale come risultato ottenuto dalla raccolta dei dati. Per implementare un sistema di content-based filtering, il primo passo è estrarre "feature" o attributi chiave dagli elementi in quesitone. Queste caratteristiche possono essere tutto ciò che caratterizza l'oggetto, come il suo genere, l'artista, il numero di "mi piace", ecc.. Successivamente, viene utilizzata una misura di somiglianza, come la somiglianza del coseno, per confrontare i vettori delle caratteristiche degli elementi per identificare elementi simili. Infine, il sistema utilizza gli elementi simili identificati per formulare raccomandazioni agli utenti.

Questo modello può essere diviso in due tipi principali: - **Item-content based**: Questo metodo utilizza le caratteristiche dell'elemento per consigliare elementi simili all'utente. Ad esempio, se a un utente piace una canzone che è classificata come "rock" e ha "Jimi Hendrix" come artista, il sistema consiglierà altre canzoni che sono anche classificate come "rock" e hanno "Jimi Hendrix" come artista. - **User-content based**: Questo metodo utilizza le features dell'utente per consigliare elementi simili alle preferenze dell'utente. Ad esempio, se a un utente piacciono i generi "rock" e "pop", il sistema consiglierà elementi che provengono da questi generi.
È importante notare che il content-based filtering è limitato dalla qualità e dalla quantità delle caratteristiche dell'elemento, se le caratteristiche non sono informative o pertinenti, le raccomandazioni non saranno accurate.
Il metodo **Content-based filtering** prevede in genere i seguenti passaggi:

- **Raccogliere e preelaborare le caratteristiche dell'oggetto**: il primo passo è raccogliere le caratteristiche degli oggetti, come il genere, l'artista, gli album, ecc.. e preelaborarle per l'analisi. Questo passaggio può includere la "pulizia" dei dati, la rimozione dei valori mancanti e la conversione di feature categoriche in valori numerici.
- **Creare un vettore di feature per ogni elemento**: una volta che le feature sono preelaborate, il passo successivo consiste nel creare un vettore di feature per ogni elemento. Questo vettore rappresenta le caratteristiche dell'elemento in forma numerica e viene utilizzato per confrontare la somiglianza tra gli elementi.
- **Calcolare la somiglianza tra gli elementi**: dopo aver creato i vettori di feature, il sistema utilizza una misura di somiglianza come la somiglianza del coseno, la somiglianza di Jaccard o la distanza euclidea per calcolare la somiglianza tra gli elementi.
- **Fare raccomandazioni**: una volta calcolata la somiglianza tra gli elementi, il sistema può formulare raccomandazioni agli utenti in base alla somiglianza. Ad esempio, se un utente ha mostrato interesse per un elemento specifico, il sistema può raccomandare altri elementi simili all'utente.
- **Rivalutare e aggiornare il modello**: il passo finale è rivalutare le prestazioni del modello e aggiornarlo se necessario. Questo passaggio può includere l'aggiunta di nuovi elementi, la rimozione di elementi irrilevanti e la regolazione del peso delle feature.

Vale la pena notare che il metodo di filtering basato sui contenuti può essere migliorato includendo informazioni aggiuntive, come le informazioni demografiche dell'utente, o integrandolo con altre tecniche di raccomandazione, come il Collaborative filtering.

Inoltre, mentre il Content-based filtering è adatto per raccomandare elementi agli utenti con una storia di interazioni, non è bravo a raccomandare elementi a nuovi utenti senza alcuna cronologia, a questo

scopo, può essere utilizzata una tecnica di filtraggio ibrido che combina i punti di forza sia del filtraggio basato sui contenuti che del filtraggio collaborativo.

Raccolta dei dati

Di seguito andremo ad inizializzare il dataset grazie alle informazioni raccolte negli step precedenti, così da poterlo partizionare in Training set e Test set, e successivamente verificarne il risultato.

In [ ]:
```
print('Creo le tabelle pivot...')

# Creo una tabella pivot con il numero di tracks per playlist di ogni utente
group.create_tracks_pivot_table()

# Creo una tabella pivot con il numero di tracks per album di ogni utente
group.create_albums_pivot_table()

# Creo una tabella pivot
group.create_artists_pivot_table()
```

```
Creo le tabelle pivot...
```

In [ ]:
```
# Prendi i top 7 artisti e cerca le loro 10 canzoni più in voga

limit = 7
take_tracks = 10
i = 0

for key, row in group.get_artists_pivot_table()[:limit].iterrows():

    # Cerco le top tracks dell'artista
    result = sp.artist_top_tracks(artist_id=key)

    # Converto la risposta in json
    items_raw = json.dumps(result['tracks'])
    items_json = json.loads(items_raw)
    items_list = list(items_json)

    # Creo una lista temporanea delle tracks dell'artista corrente
    artist_tracks = list(map(lambda track: {
        'album_id': track.get('album', {}).get('id'),
        'album_name': track.get('album', {}).get('name'),
        'album_release_date': track.get('album', {}).get('release_date'),
        'artist_id': track.get('artists', {})[0].get('id'),
        'artist_name': track.get('artists', {})[0].get('name'),
        'track_disc_number': track.get('disc_number'),
        'track_duration_ms': track.get('duration_ms'),
        'track_explicit': track.get('explicit'),
        'track_popularity': track.get('popularity'),
        'track_id': track.get('id'),
        'track_name': track.get('name')
    }, items_list))

    # Ordino la lista temporanea per l'attributo track_populatiry
    artist_tracks = sorted(artist_tracks, key=lambda track: track['track_popularity'], revers

    # Aggiungo la lista temporanea alle informazioni del gruppo
    group.append_to_artists_top_tracks(artist_tracks[:take_tracks])

    i += 1
    if i < limit:
        time.sleep(5)
```

In [ ]:
```
# Prendi i top 8 album e cerca le loro 10 canzoni più in voga

limit = 8
take_tracks = 10
i = 0

for key, row in group.get_albums_pivot_table()[:limit].iterrows():

    # Cerco le top tracks dell'artista
    result = sp.album_tracks(album_id=key)

    # Converto la risposta in json
    items_raw = json.dumps(result['items'])
    items_json = json.loads(items_raw)
    items_list = list(items_json)

    # Creo una lista temporanea delle tracks dell'album corrente
    album_tracks = list(map(lambda track: {
        'album_id': None,
        'album_name': None,
        'album_release_date': None,
        'artist_id': track.get('artists', {})[0].get('id'),
        'artist_name': track.get('artists', {})[0].get('name'),
        'track_disc_number': track.get('disc_number'),
        'track_duration_ms': track.get('duration_ms'),
        'track_explicit': track.get('explicit'),
        'track_popularity': None,
        'track_id': track.get('id'),
        'track_name': track.get('name')
    }, items_list))

    # Aggiungo la lista temporanea alle informazioni del gruppo
    group.append_to_albums_top_tracks(album_tracks[:take_tracks])

    i += 1
    if i < limit:
        time.sleep(5)
```

Training set & Test set

Di seguito andremo a partizionare il dataset in Trainig set (al 70%) e in Test set (al 30%) in maniera randomica. Successivamente ne verificheremo il risultato confrontandone la similarità tra alcune features.

In [ ]:
```
# Suddivido il dataframe con la tabella pivot di tutte le tracks ottenute in Training set e T

df = group.get_tracks_pivot_table()

train_df, test_df = train_test_split(df, test_size=0.3, random_state=42)
```

Training set

In [ ]:
```
# Definisco la recommended tracks playlist dal Training set

i = 0
limit = 40
__tracks_list = group.get_playlist_tracks_dataframe()

# Dataset di supporto
dataset = train_df

# Inizializzo la Recommended playlist
```

```
group.recommended_playlist = []

# Fintantoché la lista non raggiunge il limit desiderato
while length_hint(group.get_recommended_playlist()) < limit:

    # Prondo le informazioni della track
    track_id = dataset.iloc[i].name
    results = __tracks_list.loc[__tracks_list['track_id'] == track_id]
    item = results.iloc[0]

    track = {
        'album_id': item.get('album_id'),
        'album_name': item.get('album_name'),
        'album_release_date': item.get('album_release_date'),
        'artist_id': item.get('artist_id'),
        'artist_name': item.get('artist_name'),
        'track_disc_number': item.get('track_disc_number'),
        'track_duration_ms': item.get('track_duration_ms'),
        'track_explicit': item.get('track_explicit'),
        'track_popularity': item.get('track_popularity'),
        'track_id': item.get('track_id'),
        'track_name': item.get('track_name')
    }

    # Appendo l'item in lista
    group.append_to_recommend_playlist(track)

    i += 1
```

In [ ]:
```
# Riporto le recommended tracks (ottenute dal training set) su una lista di supporto

recommended_playlist_train_df = group.get_recommended_playlist()
```

Test set

In [ ]:
```
# Definisco la recommended tracks playlist sul Test set

i = 0
limit = 40
__tracks_list = group.get_playlist_tracks_dataframe()

# Dataset di supporto
dataset = test_df

# Inizializzo la Recommended playlist
group.recommended_playlist = []

# Fintantoché la lista non raggiunge il limit desiderato
while length_hint(group.get_recommended_playlist()) < limit:

    # Prondo le informazioni della track
    track_id = dataset.iloc[i].name
    results = __tracks_list.loc[__tracks_list['track_id'] == track_id]
    item = results.iloc[0]

    track = {
        'album_id': item.get('album_id'),
        'album_name': item.get('album_name'),
        'album_release_date': item.get('album_release_date'),
        'artist_id': item.get('artist_id'),
        'artist_name': item.get('artist_name'),
        'track_disc_number': item.get('track_disc_number'),
        'track_duration_ms': item.get('track_duration_ms'),
        'track_explicit': item.get('track_explicit'),
```

```
        'track_popularity': item.get('track_popularity'),
        'track_id': item.get('track_id'),
        'track_name': item.get('track_name')
    }

    # Appendo l'item in lista
    group.append_to_recommend_playlist(track)

    i += 1
```

In [ ]:
```
# Riporto le recommended tracks (ottenute dal training set) su una lista di supporto

recommended_playlist_test_df = group.get_recommended_playlist()
```

Verifica della qualità del risultato

In [ ]:
```
# Analisi Training set

tracks_features_train_list_energy = []
tracks_features_train_list_acousticness = []
tracks_features_train_list_tempo = []
tracks_features_train_list_loudness = []
tracks_features_train_list_valence = []
tracks_features_train_list_speechiness = []

# Per ogni elemento della lista di supporto ricavata dal Training set
for item in recommended_playlist_train_df:

    # Cerco le features della track corrente
    result = sp.audio_features(tracks=[item['track_id']])

    # Converto la risposta in json
    items_raw = json.dumps(result)
    items_json = json.loads(items_raw)
    items_list = list(items_json)

    # Estrapolo alcune delle features che caratterizzano la track corrente
    tracks_features_train_list_energy.append(items_list[0].get('energy'))
    tracks_features_train_list_acousticness.append(items_list[0].get('acousticness'))
    tracks_features_train_list_tempo.append(items_list[0].get('tempo'))
    tracks_features_train_list_loudness.append(items_list[0].get('loudness'))
    tracks_features_train_list_valence.append(items_list[0].get('valence'))
    tracks_features_train_list_speechiness.append(items_list[0].get('speechiness'))
```

In [ ]:
```
# Analisi Test set

tracks_features_test_list_energy = []
tracks_features_test_list_acousticness = []
tracks_features_test_list_tempo = []
tracks_features_test_list_loudness = []
tracks_features_test_list_valence = []
tracks_features_test_list_speechiness = []

# Per ogni elemento della lista di supporto ricavata dal Test set
for item in recommended_playlist_test_df:

    # Cerco le features della track corrente
    result = sp.audio_features(tracks=[item['track_id']])

    # Converto la risposta in json
    items_raw = json.dumps(result)
    items_json = json.loads(items_raw)
    items_list = list(items_json)
```

```
    # Estrapolo alcune delle features che caratterizzano la track corrente
    tracks_features_test_list_energy.append(items_list[0].get('energy'))
    tracks_features_test_list_acousticness.append(items_list[0].get('acousticness'))
    tracks_features_test_list_tempo.append(items_list[0].get('tempo'))
    tracks_features_test_list_loudness.append(items_list[0].get('loudness'))
    tracks_features_test_list_valence.append(items_list[0].get('valence'))
    tracks_features_test_list_speechiness.append(items_list[0].get('speechiness'))
```

In [ ]:
```
# Calcolo la similarità degli elementi delle due liste


# Conversione delle liste in array numpy

tracks_features_train_list_energy = np.array(tracks_features_train_list_energy).reshape(1,-1)
tracks_features_test_list_energy = np.array(tracks_features_test_list_energy).reshape(1,-1)

tracks_features_train_list_acousticness = np.array(tracks_features_train_list_acousticness).r
tracks_features_test_list_acousticness = np.array(tracks_features_test_list_acousticness).res

tracks_features_train_list_tempo = np.array(tracks_features_train_list_tempo).reshape(1,-1)
tracks_features_test_list_tempo = np.array(tracks_features_test_list_tempo).reshape(1,-1)

tracks_features_train_list_loudness = np.array(tracks_features_train_list_loudness).reshape(1
tracks_features_test_list_loudness = np.array(tracks_features_test_list_loudness).reshape(1,-

tracks_features_train_list_valence = np.array(tracks_features_train_list_valence).reshape(1,-
tracks_features_test_list_valence = np.array(tracks_features_test_list_valence).reshape(1,-1)

tracks_features_train_list_speechiness = np.array(tracks_features_train_list_speechiness).res
tracks_features_test_list_speechiness = np.array(tracks_features_test_list_speechiness).resha


# Calcolo del cosine similarity

similarity_energy = cosine_similarity(tracks_features_train_list_energy, tracks_features_test
similarity_acousticness = cosine_similarity(tracks_features_train_list_acousticness, tracks_f
similarity_tempo = cosine_similarity(tracks_features_train_list_tempo, tracks_features_test_l
similarity_loudness = cosine_similarity(tracks_features_train_list_loudness, tracks_features_
similarity_valence = cosine_similarity(tracks_features_train_list_valence, tracks_features_te
similarity_speechiness = cosine_similarity(tracks_features_train_list_speechiness, tracks_fea


# Stampo i risultati

print('energy similarity: ', similarity_energy)
print('acousticness similarity: ', similarity_acousticness)
print('tempo similarity: ', similarity_tempo)
print('loudness similarity: ', similarity_loudness)
print('valence similarity: ', similarity_valence)
print('speechiness similarity: ', similarity_speechiness)
```

```
energy similarity:  0.9191645269947084
acousticness similarity:  0.2796277768243608
tempo similarity:  0.9642692498474579
loudness similarity:  0.7799224674254003
valence similarity:  0.8116552047901227
speechiness similarity:  0.6307600434783865
```

Le informazioni appena ritornate indicano **features specifiche** della traccia audio, ed in particolare il risultato ottenuto con il **cosine similarity**, dove un valore prossimo ad 1 indica che gli elementi delle due liste sono simili, mentre sono diversi quando il valore è prossimo a -1:

- **Energy**: indica la quantità di energia percepita nella traccia. - **Acousticness**: indica quanto una

traccia è acustica rispetto all'uso di strumenti elettronici. - **Tempo**: indica il ritmo della traccia espresso in BPM (battiti per minuto). - **Loudness**: indica il volume medio della traccia in decibel (dB). - **Valence**: indica la positività o negatività percepita della traccia. - **Speechiness**: indica quanto una traccia contiene parole rispetto alla musica.

Playlist pubblicata!

In [ ]:
```
# Ottengo la data corrente
current_date_time = datetime.now()

# Formato la data corrente
date_format = current_date_time.strftime("%Y-%m-%d")

# Informazioni della playlist
playlist_name = 'Recommend playlist - {}'.format(date_format)
playlist_description = 'SMM'

user_id = group.get_from_list(0).user_id

# Creo la nuova playlist su Spotify e l'aggiungo a quelle dell'utente
playlist_created = sp.user_playlist_create(user=user_id, name=playlist_name, description=play
```

In [ ]:
```
# Creo una lista temporanea delle recommended tracks
track_list = list(map(lambda x: x.get('track_id'), group.get_recommended_playlist()))

playlist_id = playlist_created.get('id')

# Aggiungo tutte le recommended tracks alla playlist appena creata
sp.user_playlist_add_tracks(user=user_id, playlist_id=playlist_id, tracks=track_list)
```

Out[ ]:
```
{'snapshot_id': 'MixkOGFiNGVkZGIzMjlkZTAwYzU4NzM3MzU1YzgxOTAyODJkMGUwODY2'}
```

In [ ]:
```
print('Playlist info')
print('======================')
print(result['name'])
print(result['external_urls']['spotify'])
print('')
print('Playlist owner ' + result['owner']['display_name'])
```

```
Playlist info
======================
Recommend playlist - 2023-04-01
https://open.spotify.com/playlist/659ZHrs8psGPGshWCcOw2b

Playlist owner Francesco Petrosino
```

Metodi di Sistemi di raccomandazione a confronto

Collaborative filtering

Tra i **sistemi di raccomandazione** è noto il metodo **Collaborative filtering** per formulare raccomandazioni agli utenti in base al loro comportamento passato e al comportamento di utenti simili.

> Questo sistema utilizza per identificare modelli e somiglianze tra loro, quindi utilizza questi modelli per formulare raccomandazioni ai nuovi utenti.

Per implementare questo sistema, il primo passo è raccogliere dati sul comportamento passato degli utenti. Questi dati vengono quindi utilizzati per creare una matrice di elementi utente che rappresenta le

valutazioni che gli utenti hanno dato agli elementi. Successivamente, il sistema utilizza una misura di somiglianza (come la somiglianza coseno) per identificare utenti o elementi simili. Infine, il sistema utilizza gli utenti o gli elementi simili identificati per formulare raccomandazioni ai nuovi utenti.

È importante notare che il filtro collaborativo è una tecnica basata sulla memoria, significa che l'algoritmo deve avere accesso a tutte le interazioni utente-elemento per generare raccomandazioni, ecco perché non è adatto per set di dati di grandi dimensioni. Inoltre, può essere influenzato dal problema del cold-start, significa che l'algoritmo non può raccomandare elementi ai nuovi utenti che non hanno ancora interagito con il sistema.

Content-based filtering vs Collaborative filtering

Date le due definizioni scritte nei punti precedenti, abbiamo che: il **Content-based filtering** utilizza le informazioni per creare un profilo di interesse per l'utente e fare le raccomandazioni in base al grado di somiglianza tra i profili degli utenti e i contenuti. Al contrario, il **Collaborative filtering** si basa sulle interazioni tra gli utenti e i contenuti, ma solo se questi hanno contribuito a "popolare" il **Dataset** iniziale.

Feedback sulla playlist

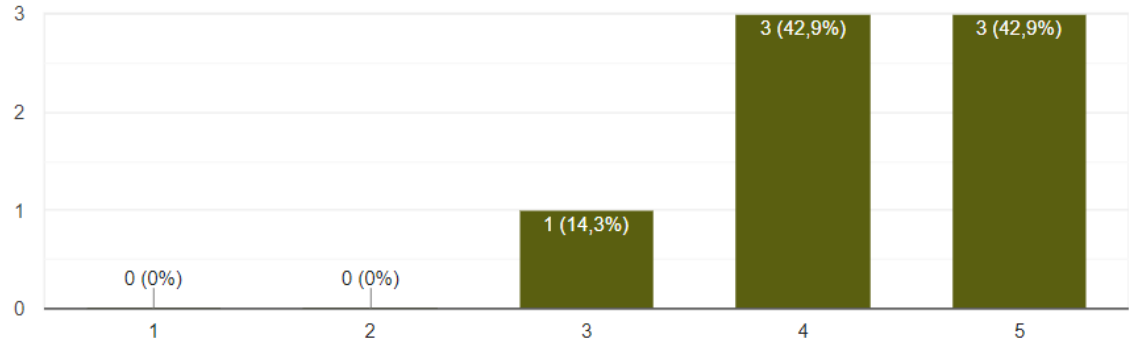
Feedback sulla playlist

Vorrei conoscere la tua opinione in modo da poter comprendere l'efficacia dell'algoritmo implementato come sistema di raccomandazione.
Compila questo breve sondaggio e comunicaci le tue idee (le risposte saranno anonime).

La playlist contiene alcune delle tue canzoni preferite o che in genere ascolti?

Copia

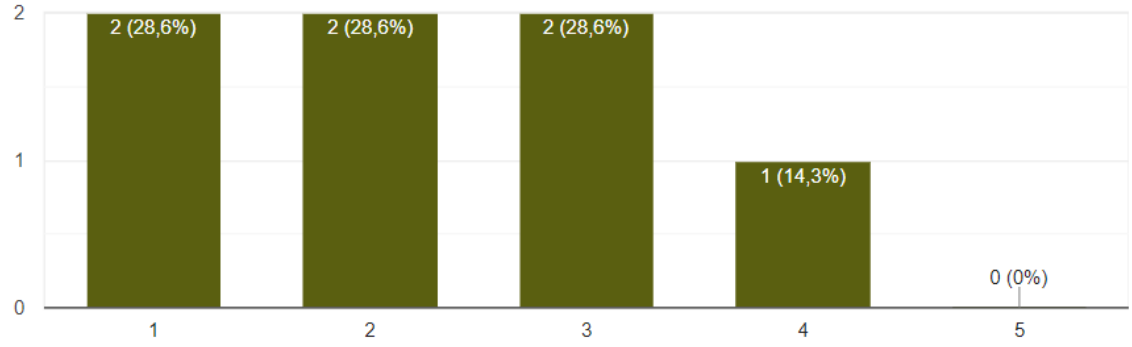
7 risposte

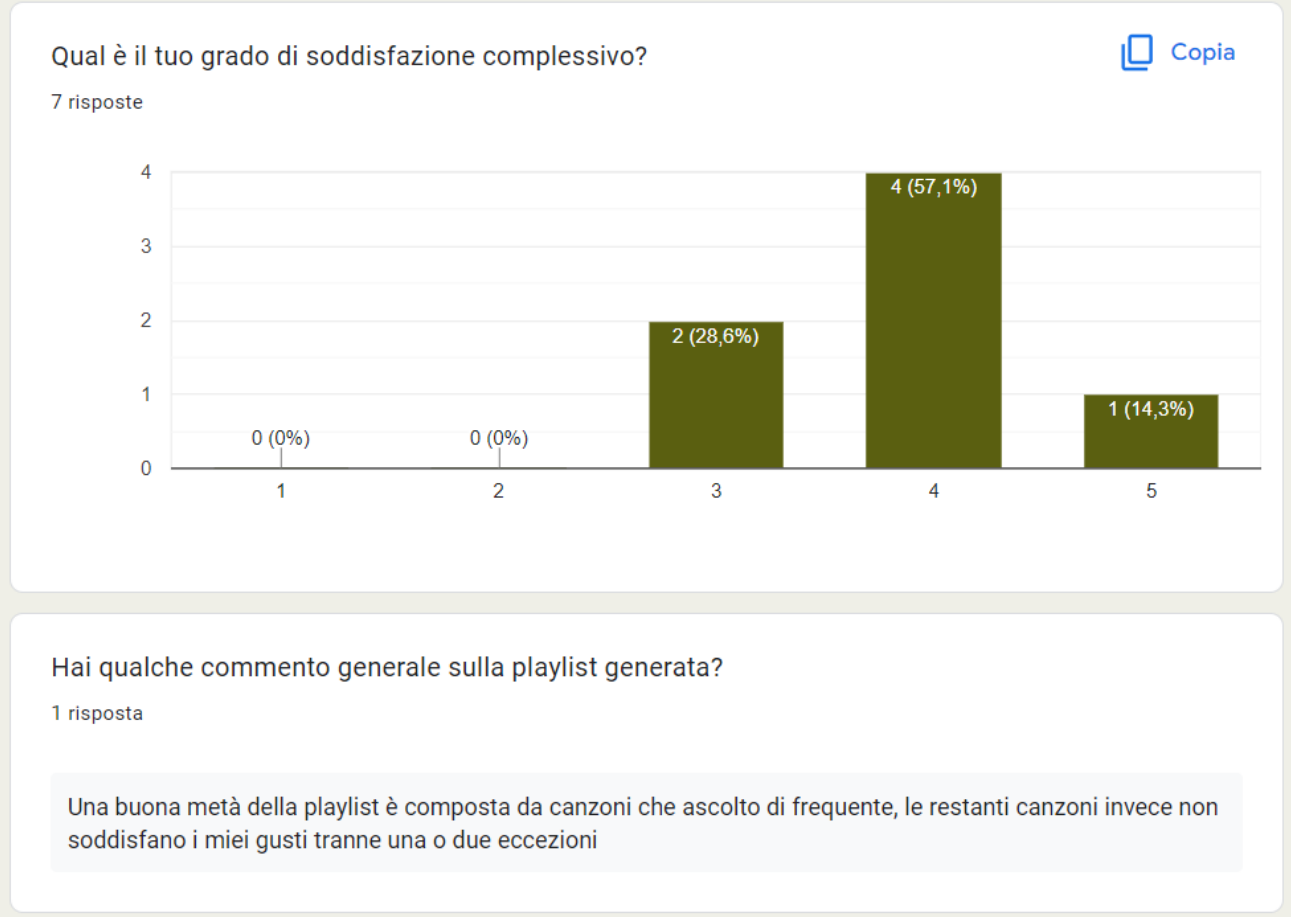


La playlist contiene alcune delle canzoni che in genere NON ascolti? Se sì, le ascolteresti volentieri?

Copia

7 risposte





Dal sondaggio possiamo evincere che le tracks inserite nella playlist suggerita dall'algoritmo, include sia tracks che generalmente gli stessi utenti ascoltano di frequente e quindi vengono apprezzate, invece non hanno ricevuto lo stesso punteggio quelle tracks che sono state inserite come suggerimento. Alla fine, il grado di soddisfazione complessivo degli utenti risulta essere sopra la media.

Conclusione

È stato scelto di implementare un algoritmo basato sul metodo di **Content-based filtering** perché si basa sull'analisi di un set di informazioni, dove è possibile che non tutti gli utenti hanno una "storia passata", ovvero che non hanno espresso opinione, alimentando così il set di dati. Mentre per il metodo **Collaborative filtering** per ottenere un risultato accettabile, è importante che tutti gli utenti partecipano nella raccolta dei dati.
In questo caso, dunque, si presuppone che gli elementi scelti dall'algoritmo, possano piacere pure agli utenti che non hanno espresso alcuna opinione, in quanto, piacciono ad altri utenti della lista.

È possibile notare come tra i risultati ottenuti dalle verifiche, le features come *energy*, *tempo* e *valence* abbiano ottenuto un valore di similitudine superiore a 0.8, mentre solo l'*acousticness* ha ottenuto un valore prossimo allo 0. Il sondaggio riporta i feedback ricevuti dagli utenti che hanno preso parte al progetto come "attori" per la simulazione di un caso di utilizzo reale, denota come i risultati ottenuti e le verifiche analizzate, siano pressoché positive.