AYDIN ADNAN MENDERES UNIVERSITY

ENGINEERING FACULTY
COMPUTER SCIENCE ENGINEERING DEPARTMENT

Spotify Song Recommender

CSE419 – Artificial Intelligence 2023/2024

Burak TÜZEL
191805057
Talha Alper ASAV
201805072

Lecturer:

Asst. Prof. Dr. Fatih SOYGAZİ

Spotify Song Recommender

Scraping Songs from Playlists:

Here We used “Spotipy” Python library to scrape our songs from the playlists we found using the keywords such as “Türkçe Pop” or “Türkçe Rock”. In the picture below you can see all the keywords and the scraping algorithm we used. There is limitation for 50 playlist per keyword in the Spotify Web API, so we used a while loop. Finally, we saved all the scraped data to a .csv file. You can check all the code from the given spotifyPSC.py file.

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import csv
import math
import time
from requests.exceptions import ReadTimeout

# Set up credentials
client_id = '3851cad810c64531b5a0afae3d22b3f9'
client_secret = 'e881bd74c38f41c78ab16e957d29f3ce'

# Authenticate with Spotify API
client_credentials_manager = SpotifyClientCredentials(client_id=client_id, client_secret=client_secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

```
# Define the base keyword and additional terms
base_keyword = 'Türkçe'
additional_terms = ['Pop', 'Hareketli', 'Rock', 'Slow', 'Akustik', 'Klasik', 'Rap', 'Jazz', 'Efsane', 'En İyi']

# Generate similar keywords by combining the base keyword with additional terms
similar_keywords = [f"{base_keyword} {term}" for term in additional_terms]
```

```
# Write all playlist data to CSV
csv_file = "scrapedSongs.csv"
with open(csv_file, mode='w', newline='', encoding='utf-8-sig') as file:
    writer = csv.writer(file)

    # Write playlist data
    for playlist in all_playlist_data:
        writer.writerow(playlist['tracks'])
```

Finding the Frequent Itemset:

From the data we scraped we use each playlist as a transaction and their song names as item names.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
import csv

# Read transactions from the playlist data
file_path = "scrapedSongs.csv"
transactions = []
with open(file_path, "r", newline="", encoding="utf-8") as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        transactions.append(row)
```

```
# Find frequent itemsets using the Apriori algorithm from mlxtend
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)
min_support = 0.01  # Minimum support threshold
frequent_itemsets = apriori(df, min_support=min_support, use_colnames=True)

# Save frequent itemsets to a CSV file
output_file = "frequent_itemsets.csv"
frequent_itemsets.to_csv(output_file, index=False, encoding="utf-8-sig")
```

Then used the found frequent item sets to create recommendations.

```
target_item = 'Poşet'

# Generate recommendations for the target item
all_recommendations = []
for itemset in frequent_itemsets:
    if target_item in itemset:
        recommendations = itemset - {target_item}
        all_recommendations.extend(recommendations)

# Count occurrences of each recommendation
recommendation_counts = {recommendation: all_recommendations.count(recommendation) for recommendation in set(all_recommendations)}

# Sort recommendations by their frequency in descending order
sorted_recommendations = sorted(recommendation_counts.items(), key=lambda x: x[1], reverse=True)

# Extract top 10 unique recommendations from the sorted list
top_recommendations = [recommendation[0] for recommendation in sorted_recommendations[:10]]

# Print top 10 recommendations
print("Top {} Unique Recommendations Ordered by Frequency:".format(len(top_recommendations)))
for recommendation in top_recommendations:
    print(recommendation)
```

Here are the results for Frequent Itemset Based Recommendations for ‘Poşet’ as Input Song

```
Top 10 Unique Recommendations Ordered by Frequency:
Yakar Geçerim
Salla
Havaalanı
Gıybet
İki Deli
Bodrum
Dansöz
Naber?
Şans Meleğim
Öp

In [6]:
```

Then We Applied TF-IDF Vectorizer to Scraped Data:

We applied TF-IDF Vectorizer to find the relationships in the scraped data. Then using those relationships we created recommendations.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

```
# Read transactions from the playlist data
file_path = "scrapedSongs.csv"
transactions = []
with open(file_path, "r", newline="", encoding="utf-8") as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        transactions.append(row)

# Preprocess data
playlist_songs = [" ".join(playlist) for playlist in transactions]
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(playlist_songs)
```

Here in this function we are calculating the TF-IDF scores and based on the scores we are creating recommendations.

```
# Function to recommend songs
def recommend_songs(input_song, tfidf_matrix, tfidf_vectorizer, n=10):
    input_song_tfidf = tfidf_vectorizer.transform([input_song])

    # Calculate cosine similarity
    cosine_similarities = linear_kernel(input_song_tfidf, tfidf_matrix).flatten()

    # Get indices and scores of top n similar playlists
    top_indices_with_scores = [(i, score) for i, score in enumerate(cosine_similarities)]
    top_indices_with_scores.sort(key=lambda x: x[1], reverse=True)
    top_indices_with_scores = top_indices_with_scores[:n]

    # Get songs from the recommended playlists
    recommended_songs = []
    for i, _ in top_indices_with_scores:
        playlist_songs = transactions[i]
        recommended_songs.extend(playlist_songs)

    # Remove duplicates from recommended songs
    recommended_songs = list(set(recommended_songs))

    # Get TF-IDF scores of recommended songs
    tfidf_scores = []
    for song in recommended_songs:
        song_tfidf = tfidf_vectorizer.transform([song])
        tfidf_score = song_tfidf.max()  # Get the maximum TF-IDF score for the song
        tfidf_scores.append((song, tfidf_score))

    # Sort TF-IDF scores in descending order
    tfidf_scores.sort(key=lambda x: x[1], reverse=True)

    return recommended_songs[:n], top_indices_with_scores, tfidf_scores[:n]
```

Here again we are creating recommendations based on the input song ‘Poşet’

```
# Example usage
input_song = "Poşet"
recommendations, top_indices_with_scores, tfidf_scores = recommend_songs(input_song, tfidf_matrix, tfidf_vectorizer)
print("Top 10 recommended songs for:", input_song)
for song in recommendations:
    print(song)

# Printing top indices with scores
print("\nTop indices with scores:")
for index, score in top_indices_with_scores:
    print("Playlist Index:", index, "Score:", score)

# Printing top 10 TF-IDF scores of recommended songs
print("\nTop 10 TF-IDF scores of recommended songs:")
for song, tfidf_score in tfidf_scores:
    print("Song:", song, "TF-IDF Score:", tfidf_score)
```

Here are the results for TF-IDF functions.

```
Top indices with scores:
Playlist Index: 148 Score: 0.15879438680119182
Playlist Index: 882 Score: 0.14519826366357838
Playlist Index: 522 Score: 0.14229012215067238
Playlist Index: 40 Score: 0.12035849819436087
Playlist Index: 967 Score: 0.12025908605323822
Playlist Index: 72 Score: 0.09879560449112143
Playlist Index: 861 Score: 0.08492716539776586
Playlist Index: 127 Score: 0.07786972866579421
Playlist Index: 880 Score: 0.07567845740853392
Playlist Index: 24 Score: 0.06977953863592115
```

```
...:     print( Song: , song,  TF-IDF Score: ,
Top 10 recommended songs for: Poşet
Gel Bana
Gamzelim
Zalim
Raf
Nerdesin?
İki Deli
Pişman Değilim
Gurbet
Yatıya
Güzelim
```

Finally, We Created an App to Increase User Experience:

We are using Tkinter to create the UI. Again, check the recommendationApp.py to see code easily.

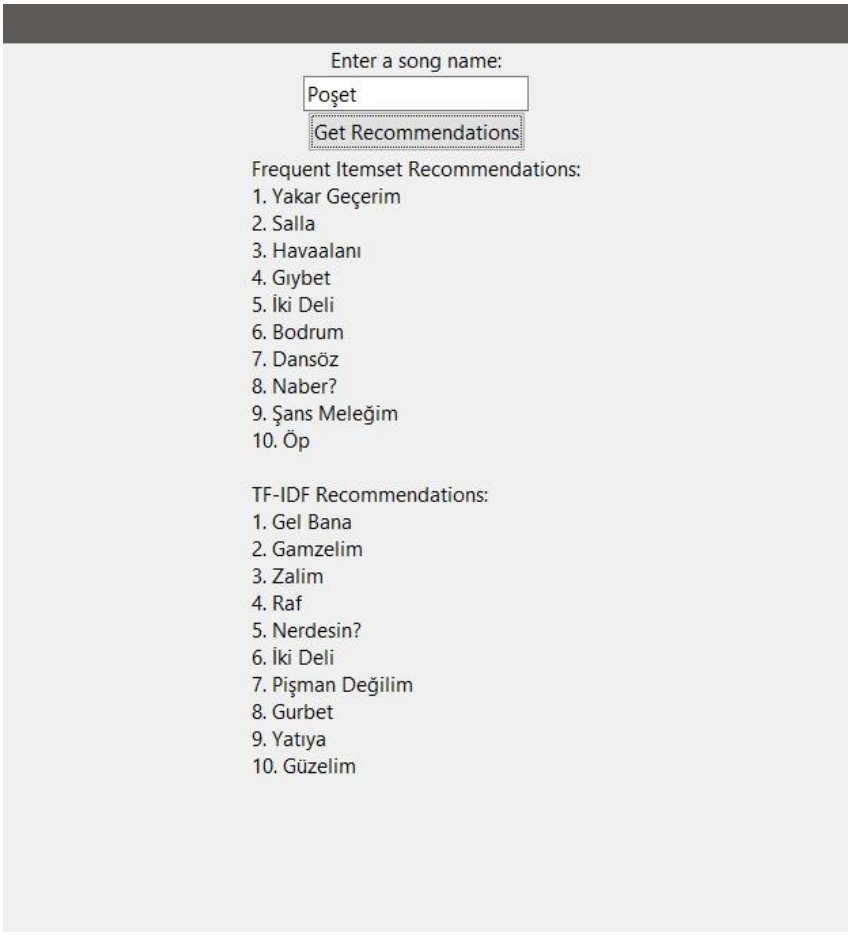
```
import tkinter as tk
from tkinter import ttk
import pandas as pd
import csv
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

```
class RecommendationApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Recommendation App")

        self.create_widgets()

    def create_widgets(self):
        # Song name input
        song_label = ttk.Label(self, text="Enter a song name:")
        song_label.pack()
        self.song_entry = ttk.Entry(self)
        self.song_entry.pack()

        # Button to trigger recommendations
        recommend_button = ttk.Button(self, text="Get Recommendations", command=self.get_recommendations)
        recommend_button.pack()

        # Result label
        self.result_label = ttk.Label(self, text="")
        self.result_label.pack()
```

```
def get_recommendations(self):
    input_song = self.song_entry.get()

    fi_recommendations = self.get_frequent_itemset_recommendations(input_song)
    tfidf_recommendations = self.get_tfidf_recommendations(input_song)

    result_text = "Frequent Itemset Recommendations:\n"
    for i, recommendation in enumerate(fi_recommendations, 1):
        result_text += f"{i}. {recommendation}\n"

    result_text += "\nTF-IDF Recommendations:\n"
    for i, recommendation in enumerate(tfidf_recommendations, 1):
        result_text += f"{i}. {recommendation}\n"

    self.result_label.config(text=result_text)

def get_frequent_itemset_recommendations(self, target_item):
    frequent_itemsets_df = pd.read_csv("frequent_itemsets.csv")
    frequent_itemsets = [set(eval(itemset)) for itemset in frequent_itemsets_df['itemsets']]

    all_recommendations = []
    for itemset in frequent_itemsets:
        if target_item in itemset:
            recommendations = itemset - {target_item}
            all_recommendations.extend(recommendations)

    recommendation_counts = {recommendation: all_recommendations.count(recommendation) for recommendation in set(all_recommenda
    sorted_recommendations = sorted(recommendation_counts.items(), key=lambda x: x[1], reverse=True)
    top_recommendations = [recommendation[0] for recommendation in sorted_recommendations[:10]]

    return top_recommendations
```

```
    def get_tfidf_recommendations(self, input_song):
        transactions = []
        with open("scrapedSongs.csv", "r", newline="", encoding="utf-8") as file:
            csv_reader = csv.reader(file)
            for row in csv_reader:
                transactions.append(row)

        playlist_songs = [" ".join(playlist) for playlist in transactions]
        tfidf_vectorizer = TfidfVectorizer()
        tfidf_matrix = tfidf_vectorizer.fit_transform(playlist_songs)

        input_song_tfidf = tfidf_vectorizer.transform([input_song])

        cosine_similarities = linear_kernel(input_song_tfidf, tfidf_matrix).flatten()
        top_indices = cosine_similarities.argsort()[:-10-1:-1]

        recommended_songs = []
        for i in top_indices:
            playlist_songs = transactions[i]
            recommended_songs.extend(playlist_songs)

        recommended_songs = list(set(recommended_songs))

        return recommended_songs[:10]

if __name__ == "__main__":
    app = RecommendationApp()
    app.mainloop()
```

Here is the final look and results.



REFERENCES

1. Lecture Notes.
2. https://www.geeksforgeeks.org/apriori-algorithm/
3. https://www.semrush.com/blog/tf-idf/
4. https://spotipy.readthedocs.io/en/2.22.1/