Fundaments and Application of AI
Final project

Lorenzo D'Amico

June 2024

0.1 Titolo

Assistente personale per la predizione di canzoni.

0.2 Sommario

Il progetto è incentrato sulla realizzazione di un software che permette, dato in ingresso un titolo di una canzone, oppure un insieme di titoli, di ottenere in output un insieme finito di titoli di canzoni simili alla canzone passata in input. Il software è realizzato in python utilizzando delle librerie specifiche per la gestione, l'analisi e la visualizzazione dei dati e la gestione dell'interfaccia utente (librerie citate nella sezione Materials and Methods 0.4). Il progetto comprende in parte del codice realizzato da Vatsal Mavani, nel progetto Music Recommendation System using Spotify Dataset (https://www.kaggle.com/code/vatsalmavani/music-recommendation-system-using-spotify-dataset/notebook).

0.3 Introduzione

Il software è composto da diverse componenti che svolgono in successione diversi compiti al fine di ottenere un risultato il più possibile coerente rispetto agli obiettivi. I componenti principali sono due: un Consistency Satisfaction Problem (problema di soddisfazione dei vincoli) e un componente di Machine Learning. In sintesi sfruttando un dataset di canzoni si vuole ottenere un insieme di raccomandazioni partendo da una o più canzoni in input. Dall'insieme appena descritto si ricava l'insieme delle soluzioni tramite il passaggio attraverso un CSP che scarta in base ai vincoli le canzoni che non rispettano questi ultimi.

0.4 Materiali e metodi

Il progetto è realizzato in codice python così suddiviso:

- **cartella classes/CSP**: in particolare la cartella CSP contiene file per la creazione del un CSP e la risoluzione del problema tramite differenti algoritmi.
- **cartella dataset**: la cartella dataset contiene dei dataset utilizzati per l'analisi dei dati.

- **cartella templates**: la cartella templates contiene dei file html che vengono utiizzati da Flask (libreria python) che permette la realizzazione di un interfaccia grafica.
- **classe csp (classes/csp.py)**: la classe csp si occupa della creazione del problema utilizzando le classi nella cartella CSP e implementa delle funzioni specifiche per questo progetto.
- **classe datacluster (classes/datacluster.py)**: la classe datacluster si occupa della creazione di oggetti che permettono l'allenamento di alcuni modelli utilizzati per l'analisi e la visualizzazione dei dati contenuti nel dataset.
- **classe ML (classes/ml.py)**: la classe ML si occupa di raccogliere gli oggetti di input e, tramite delle funzioni, di restituire la raccomandazione.
- **classe Recommender (classes/recommender.py)**: la classe Recommender si occupa di mettere in comunicazione l'input e il dataset e di riuscire ad effettuare ed analizzare la raccomandazione.
- **classe Configuration (classes/configuration.py)**: la classe Configuration si occupa di creare un collegamento con le API di spotify per python, in modo tale da avere un riferimento alle API e di analizzare il file di configurazione config.txt che contiene le chiavi API e un parametro di configurazione.
- **file index.py**: il file index.py è il riferimento all'intera applicazione. Tramite il comando

```
1       flask --app index run --debugger
2
```

  si può eseguire l'applicazione.

0.5 Esperimenti e risultati

L'esperimento per iniziare ha bisogno di alcuni prerequisiti. Il codice sopradescritto deve essere scaricato in un'unica cartella che può avere un nome arbitrario. Deve essere creato un file di configurazione con all'interno un oggetto tra parentesi graffe che contiene client_id e client_secret come nell'esempio. Esempio:

```
1 {"client_id" : "xxx", "client_secret" : "yyy"}
```

0.5.1 Creazione file di configurazione

Questo oggetto può essere creato seguendo i passi qui descritti:

1. Accedi a Spotify for Developers
   Visita il sito web di Spotify for Developers e accedi con il tuo account Spotify esistente o crea un nuovo account se non ne hai già uno.

2. Crea una nuova applicazione
   Una volta effettuato l'accesso, fai clic sul pulsante "Create an App" o "Create an Application" per creare una nuova applicazione.

3. Compila i dettagli dell'applicazione
   Compila i dettagli richiesti per la tua nuova applicazione, inclusi il nome dell'applicazione, la descrizione e il sito web (puoi anche utilizzare un sito web fittizio se non ne hai uno).

4. Ottieni le credenziali del client
   Dopo aver creato l'applicazione, verrai reindirizzato alla pagina delle impostazioni dell'applicazione. Qui troverai le credenziali del client, inclusi il Client ID e il Client Secret. Assicurati di copiare accuratamente queste informazioni in un luogo sicuro.

5. Configura le autorizzazioni
   Nella stessa pagina delle impostazioni dell'applicazione, potrai anche configurare le autorizzazioni richieste per la tua applicazione. Questo determinerà quali operazioni l'applicazione può eseguire utilizzando le API di Spotify.

6. Utilizza le credenziali per creare il file

Una volta fatto ciò si può procedere all'esecuzione dell'applicazione tramite la riga di comando descritta nella sezione precedente. Dopo alcuni secondi si otterà a linea di comando un link che aprirà una pagina sul browser con una barra di ricerca che permette di ricercare una canzone per titolo. Una volta effettuata la ricerca il sistema provvederà a cercare delle canzoni con quel titolo e verrà chiesto di selezionare la canzone corretta. Una volta selezionata la canzone si può scegliere se aggiungere altre canzoni in input oppure cercare delle canzoni simili all'input dato fino a quel momento. Qui entrano in gioco i componenti di machine learning e di csp che permettono di selezionare un insieme di canzoni simili e di scartare quelle che non rispettano i vincoli.

0.5.2 Specifica su i componenti di analisi dei dati

In particolare il modello di machine learning si basa su una Pipeline composta da uno StandardScaler e un KMeans che permettono di scalare i valori numerici prelevati dal dataset (riguardo le caratteristiche della canzone sia informative che audio) e creare una serie di sottogruppi con caratteristiche simili. Alla fine del processo di analisi del dataset si ottiene una matrice di valori numerici che sono rappresentativi delle canzoni. Se si vuole ottenere una canzone simile basta cercare valori simili ad una canzone data in ingresso nella matrice. Per quanto riguarda i vincoli del CSP in realtà c'è un unico vincolo che letteralmente si riassume nella ricerca della presenza della canzone generata dal componente di machine learning nelle playlist dell'utente per cercare di trovare una corrispondenza e quindi evitare di raccomandare una canzone che l'utente già conosce ed ha addirittura inserito in una playlist. (Ad ogni modo la canzone "scartata" sarà comunque presente nella lista di canzoni simili ma sarà a livello grafico caratterizzata da un bordo rosso).

0.5.3 Analisi dei dati

L'analisi dei dati viene effettuata solamente se si imposta nel file di configurazione la chiave "view": "False".

```
    def __init__(self, view=False):
        if view:
            X = self.method1()
            self.method2(view, X)
            Y, song_cluster_pipeline = self.method3()
            self.method4(view, Y)
            self.song_cluster_pipeline =
song_cluster_pipeline
        else:
            Y , song_cluster_pipeline = self.method3()
            self.song_cluster_pipeline =
song_cluster_pipeline

    def method1(self):
        cluster_pipeline = Pipeline([('scaler',
StandardScaler()), ('kmeans', KMeans(n_clusters=10))])
        X = self.genre_data.select_dtypes(np.number)
        cluster_pipeline.fit(X)
        self.genre_data['cluster'] = cluster_pipeline.predict
(X)
        return X

    def method2(self, view, X):
```

```
        tsne_pipeline = Pipeline([('scaler', StandardScaler()
), ('tsne', TSNE(n_components=2, verbose=1))])
        genre_embedding = tsne_pipeline.fit_transform(X)
        projection = pd.DataFrame(columns=['x', 'y'], data=
genre_embedding)
        projection['genres'] = self.genre_data['genres']
        projection['cluster'] = self.genre_data['cluster']

        fig = px.scatter(
            projection, x='x', y='y', color='cluster',
hover_data=['x', 'y', 'genres'])
        if view:
            fig.show()

    def method3(self):
        song_cluster_pipeline = Pipeline([('scaler',
StandardScaler()),
                                          ('kmeans', KMeans
(n_clusters=20,
                                          verbose=False)) #
n_clusters = 20 numero di cluster creati, verbose = False:
 non stampa informazioni extra
                                          ], verbose=False)

        Y = self.data.select_dtypes(np.number) #This selects
only the columns in the DataFrame data that have numerical
 data types
        song_cluster_pipeline.fit(Y)
        song_cluster_labels = song_cluster_pipeline.predict(Y
)
        self.data['cluster_label'] = song_cluster_labels
        return Y, song_cluster_pipeline

    def method4(self, view, Y):
        pca_pipeline = Pipeline([('scaler', StandardScaler())
, ('PCA', PCA(n_components=2))])
        song_embedding = pca_pipeline.fit_transform(Y)
        projection = pd.DataFrame(columns=['x', 'y'], data=
song_embedding)
        projection['title'] = self.data['name']
        projection['cluster'] = self.data['cluster_label']

        fig = px.scatter(
            projection, x='x', y='y', color='cluster',
hover_data=['x', 'y', 'title'])
        if view:
            fig.show()
```

In questa sezione in sostanza vengono effettuate delle analisi sui dataset.

In particolare vengono utilizzati due algoritmi chiamati: KMeans (clustering) e StandardScaler (standardization). Mentre per la visualizzazione dei dati sono stati utilizzati due algoritmi: TSNE (algoritmo di riduzione di dimensionalità) e PCA (riduzione di dimensionalità lineare) che permettono di analizzare i dati dei dataset delle canzoni e dei generi al fine di comprendere al meglio su quali dati si sta lavorando.

0.5.4 Spiegazione funzione di raccomandazione

```
def recommend_songs(self, song_list, n_songs=10):
        metadata_cols = ['name', 'year', 'artists']
        song_dict = self.flatten_dict_list(song_list)
        song_center = self.get_mean_vector(song_list, self.
    spotify_data)

        scaler = self.song_cluster_pipeline.steps[0][1]
        scaled_data = scaler.transform(self.spotify_data[self
    .number_cols])
        scaled_song_center = scaler.transform(song_center.
    reshape(1, -1))

        distances = cdist(scaled_song_center, scaled_data, '
    cosine')
        index = list(np.argsort(distances)[:, :n_songs][0])

        rec_songs = self.spotify_data.iloc[index]
        rec_songs = rec_songs[~rec_songs['name'].isin(
    song_dict['name'])]
        return rec_songs[metadata_cols].to_dict(orient='
    records')
```

Si assume di avere una matrice M dove le righe sono osservazioni e le colonne sono le variabili/features.
L'idea di base è di normalizzare/standardizzare i valori sulle righe ovvero avere una media = 0 e deviazione standard = 1 per ogni feature/variabile/-colonna di M, individualmente, prima di applicare qualsiasi modello di machine learning.
Piccolo esempio:

```
data = [[0, 0],
        [1, 0],
        [0, 1],
        [1, 1]]
scaledData = [[-1, -1],
              [1, -1],
```

```
                [-1, 1],
                [1, 1]]
```

Media delle colonne = 0 e devizione standard delle colonne = 1.

Standardization:

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

with mean:

$$\mu = 1/N \sum_{i=1}^{N} (x_i) \tag{2}$$

and standard deviation:

$$\sigma = \sqrt{1/N \sum_{i=1}^{N} (x_i - \mu)^2} \tag{3}$$

Queste operazioni vengono effettuate su un vettore di dati numerici caratteristici di ogni canzone. Per creare questo vettore viene richiamata la funzione descritta seguentemente

```
    def get_mean_vector(self, song_list, data):
        song_vectors = []

        for song in song_list:
            song_data = self.get_song_data(song, spotify_data
    )
            if song_data is None:
                print('Warning: {} does not exist in Spotify
    or in database'.format(song['name']))
                continue
            song_vector = song_data[self.number_cols].values
            song_vectors.append(song_vector)

        song_matrix = np.array(list(song_vectors))
        return np.mean(song_matrix, axis=0)
```

Nella riga riguardante il calcolo della distanza tra i valori prelevati da ogni canzone si è scelto di utilizzare la metrica 'cosine' che segue la seguente formula.

```
    Y = cdist(XA, XB, 'cosine')
```

Computes the cosine distance between vectors u and v

$$1 - \frac{u \cdot v}{\|\mathbf{u}\|_2 \|\mathbf{v}\|_2} \tag{4}$$

Una volta calcolato il vettore delle distanze viene creato un indice che verrà utilizzato per ricercare nel dataset le canzoni trovate. Ovviamente dal vettore distanze vengono prelevate soltato un numero finito di canzoni che in questo caso è 10.

0.6 Conclusione

In conclusione il progetto realizzato fornisce un insieme di canzoni simili ad un input dato. Si nota che rispetto un insieme di esperimenti uguali l'output rimarrà invariato poiché dietro la predizione ci sono calcoli puramente matematici effettuati sempre a partire dallo stesso dataset di canzoni. Dunque la predizione rimane limitata poiché il dataset ha un numero di canzoni limitato.