Relazione Text Mining - Dr.House

Matteo Ianeri

1 Obiettivo 1: Estrazione dei Sommari

L'obiettivo di questa parte del progetto è scaricare i sommari di tutti gli episodi della prima stagione della serie Dr. House, preferibilmente in lingua inglese.

Descrizione del Processo

Nello sviluppo di questo script, ho utilizzato diverse librerie Python per facilitare l'estrazione e l'analisi dei dati:

- **Requests:** Utilizzata per inviare richieste HTTP. Specificamente, ho effettuato una richiesta GET all'URL della pagina Wikipedia per ottenere il contenuto HTML.
- **BeautifulSoup:** Impiegata per analizzare il contenuto HTML ricevuto, facilitando l'estrazione dei dettagli specifici sugli episodi.
- **Pandas:** Usata per organizzare i dettagli di ciascun episodio in una lista di dizionari, successivamente convertiti in un DataFrame per agevolare ulteriori analisi e visualizzazioni.

Durante la funzione `scrape_house_episodes`, ho iterato su ogni riga della tabella degli episodi, estraendo dettagli come il numero dell'episodio, il titolo, il regista, lo sceneggiatore, la data di messa in onda e gli ascolti negli USA. Per gli episodi con informazioni aggiuntive, come la diagnosi finale, ho gestito l'estrazione in modo condizionale.
Se la richiesta HTTP falliva (indicato da un codice di stato diverso da 200), il codice era programmato per segnalare un errore e restituire una lista vuota.

Visualizzazione dei Dati

Alla fine dello script, ho visualizzato un'anteprima del DataFrame creato per verificare la correttezza e la completezza dei dati estratti.

```
import requests  # Importa la libreria requests per fare richieste
    HTTP
from bs4 import BeautifulSoup  # Importa BeautifulSoup da bs4 per
    analizzare documenti HTML
import pandas as pd  # Importa pandas per la manipolazione e l'
    analisi dei dati
import numpy as np

def scrape_house_episodes(url):  # Definisce una funzione per
    raschiare gli episodi di House da Wikipedia
    episodes = []  # Inizializza una lista vuota per conservare i
        dettagli degli episodi
    response = requests.get(url)  # Effettua una richiesta HTTP GET
         all'URL specificato
    if response.status_code == 200:  # Verifica se la richiesta ha
        avuto successo (codice di stato 200)
        soup = BeautifulSoup(response.content, 'html.parser')  #
            Analizza il contenuto della risposta con BeautifulSoup
        # Trova tutte le righe nella tabella degli episodi usando
            il loro attributo class 'vevent'
        episode_rows = soup.find_all('tr', 'vevent')

        for episode_row in episode_rows:  # Itera su ogni riga
            trovata
            # Estrae i dettagli dell'episodio dalla riga corrente e
                 li memorizza in un dizionario
            episode_details = {
                'episode_number': episode_row.find('th', id=True).
                    get_text(strip=True),  # Numero dell'episodio
                'title': episode_row.find('td', 'summary').get_text
                    (strip=True),  # Titolo dell'episodio
                'directed_by': episode_row.find_all('td')[2].
                    get_text(strip=True),  # Regista
                'written_by': episode_row.find_all('td')[3].
                    get_text(strip=True),  # Sceneggiatore
                'original_air_date': episode_row.find_all('td')[4].
                    get_text(strip=True),  # Data di messa in onda
                'us_viewers': episode_row.find_all('td')[5].
                    get_text(strip=True).split('[')[0]  # Ascolti
                    in USA, rimuove riferimenti
            }

            # Trova una riga successiva che contiene la descrizione
                 estesa dell'episodio
            expand_row = episode_row.find_next_sibling('tr', '
                expand-child')
            if expand_row:  # Se tale riga esiste
                # Estrae il testo della trama e lo assegna al
                    dizionario
                plot_paragraphs = expand_row.find('td', '
                    description').find_all('p')
                plot_text = ' '.join(paragraph.get_text(strip=True)
                     for paragraph in plot_paragraphs)
                episode_details['plot'] = plot_text
```

```
                # Estrae la diagnosi finale e la assegna, se non
                    presente assegna 'N/A'
                final_diagnosis = expand_row.find('i').
                    find_next_sibling(text=True).strip()
                episode_details['final_diagnosis'] =
                    final_diagnosis if final_diagnosis else 'N/A'

            episodes.append(episode_details)  # Aggiunge i dettagli
                 dell'episodio alla lista degli episodi

        return episodes  # Restituisce la lista degli episodi
    else:  # Se la richiesta non    andata a buon fine
        print(f'Errore: impossibile recuperare la pagina ({response
            .status_code}).')  # Stampa un messaggio di errore
        return []  # Restituisce una lista vuota

# URL della pagina Wikipedia della stagione 1 di House
url = 'https://en.wikipedia.org/wiki/House_(season_1)'
house_episodes = scrape_house_episodes(url)  # Chiama la funzione
    scrape_house_episodes con l'URL specificato

# Converti l'elenco di dizionari in un DataFrame per facilitare l'
    analisi e la visualizzazione
house_episodes_df = pd.DataFrame(house_episodes)

# Mostro l'anteprima del dataset
house_episodes_df.head()
```

1.1 Obiettivo 2: Analisi della Similarità

L'obiettivo per questa sezione è analizzare la similarità di argomenti tra tutti gli episodi, presi a coppie, considerando i vettori di embedding delle parole del sommario di ogni episodio, dopo aver eliminato le stopwords. Per il calcolo della similarità, utilizzeremo la similarità coseno.
Prima di procedere con i vettori di embedding, eseguo una pulizia del testo utilizzando diverse tecniche.

1.1.1 Rimozione StopWords

Le stop word sono parole comuni che vengono escluse dalle attività di elaborazione del testo, come nel caso dell'elaborazione del linguaggio naturale e degli algoritmi dei motori di ricerca. Termini come "the", "and" e "is" vengono considerati insignificanti perché ricorrono frequentemente e hanno un valore semantico limitato.
Spesso queste parole vengono rimosse per migliorare l'efficienza e la precisione di varie operazioni di analisi testuale. L'eliminazione delle stop word permette agli algoritmi di focalizzarsi su termini più rilevanti, facilitando una comprensione più approfondita del contesto e del significato del testo.
L'obiettivo principale di questa rimozione è ridurre il carico computazionale e i requisiti di archiviazione durante l'analisi dei dati testuali. Eliminando parole comuni e ricorrenti, i dati diventano più snelli e significativi per l'elaborazione.
Le stop word derivano solitamente da un elenco predefinito di termini considerati irrilevanti per l'analisi, e questo elenco può variare in base all'attività o al dominio specifico. Alcune stop word comuni in inglese includono "a", "an", "the", "in", "and" e "is".
Le stop word cambiano da una lingua all'altra, poiché ciascuna ha il proprio insieme di parole di uso frequente che possono essere considerate insignificanti. Ad esempio, "the" è una stop word comune in inglese, ma potrebbe non avere un corrispettivo diretto in altre lingue.
La rimozione delle stop word può avere effetti sia positivi che negativi sull'analisi del testo. Da un lato, aiuta a ridurre il rumore e a migliorare l'accuratezza dei modelli di apprendimento automatico e dei motori di ricerca. Dall'altro, può comportare la perdita di informazioni contestuali importanti, specialmente in applicazioni come l'analisi del sentiment.

1.1.2 Tokenizzazione

I token possono essere parole, numeri o segni di punteggiatura. La tokenizzazione è il processo attraverso il quale un testo viene suddiviso in unità più piccole, individuando i confini tra le parole. Questi confini corrispondono alla fine di una parola e all'inizio della successiva. La tokenizzazione rappresenta un passaggio preliminare fondamentale per altre operazioni di elaborazione del linguaggio naturale, come lo stemming e la lemmatizzazione.
Esistono tre tipi principali di tokenizzazione in Python:

- **Tokenizzazione delle parole:** Consiste nel suddividere una frase nelle sue singole parole.
- **Tokenizzazione delle frasi:** Suddivide un paragrafo in frasi separate.
- **Tokenizzazione tramite espressioni regolari:** Utilizza pattern specifici per dividere il testo in base a regole o condizioni prestabilite.

Il motivo principale per cui questo processo è importante è che aiuta le macchine a comprendere il linguaggio umano scomponendolo in pezzi di dimensioni ridotte, che sono più facili da analizzare. Utilizzando il tokenizer Punkt di NLTK, ho suddiviso i sommari in token. La tokenizzazione è il processo di suddivisione del testo in parole, frasi, simboli o altri elementi significativi chiamati token. Questo facilita l'analisi successiva del testo.

1.1.3 Rimozione Punteggiatura

Ho implementato una funzione per eliminare la punteggiatura dai token, poiché caratteri come virgole, punti e virgolette possono alterare gli outcome dell'analisi.

1.1.4 Normalizzazione

Oltre alla rimozione della punteggiatura, ho convertito tutti i token in minuscolo per garantire che le parole siano comparate uniformemente, indipendentemente dal loro uso nel testo.

1.1.5 Lemmatizzazione

Prima di parlare della lemmatizzazione, spiego il perchè non ho usato lo stemming.
Lo stemming è un processo che consiste nel rimuovere i suffissi dalle parole per ottenere una forma abbreviata, anche se quest'ultima potrebbe non avere un significato preciso. Ad esempio, le parole "storia" e "storico" vengono ridotte a "histori". Questo metodo è particolarmente utilizzato nell'apprendimento automatico per ridurre le parole alla loro radice, semplificando l'elaborazione, anche se la forma risultante non è sempre una parola valida.
La lemmatizzazione, invece, ha lo stesso obiettivo dello stemming, ossia ridurre le parole alla loro forma di base, ma lo fa in modo più accurato. A differenza dello stemming, che può produrre risultati non significativi come "histori", la lemmatizzazione restituisce una parola con senso compiuto, come "storia". Questo approccio considera il contesto grammaticale delle parole per ottenere una rappresentazione semantica corretta.
La lemmatizzazione richiede più tempo rispetto allo stemming perché, oltre a rimuovere i suffissi, individua la forma significativa della parola. Lo stemming, invece, si limita a ridurre la parola alla sua radice, richiedendo meno tempo di elaborazione.
Entrambi i processi hanno applicazioni specifiche: lo stemming è comunemente usato nell'analisi del sentiment, dove è sufficiente identificare la radice delle parole, mentre la lemmatizzazione è più utile in applicazioni come i chatbot, che devono comprendere e rispondere in modo più preciso.
Lo stemming è un processo di normalizzazione linguistica nell'elaborazione del linguaggio naturale e nel recupero delle informazioni, il cui scopo è ridurre le parole alle loro forme di base o radici. Rimuovendo affissi come prefissi o suffissi, si ottiene una forma semplificata della parola, utile per attività come il recupero delle informazioni. Tuttavia, lo stemming è un processo euristico, quindi non sempre produce parole valide, ma è comunque efficace quando l'obiettivo è trovare corrispondenze di significato piuttosto che correttezza grammaticale.
Gli algoritmi di stemming applicano regole ed euristiche per identificare e rimuovere gli affissi, semplificando il testo e migliorando l'efficienza nell'analisi delle informazioni.
La lemmatizzazione, d'altra parte, è un processo linguistico che riduce le parole alla loro forma base o lemma, normalizzando le varie forme flesse di una parola per renderle più facili da analizzare o confrontare. Questo processo è particolarmente utile nell'elaborazione del linguaggio naturale (NLP) e nell'analisi del testo.

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string

# Carica una volta le stopwords
stop_words = set(stopwords.words('english'))

def tokenize_normalize(text):
    # Rimuove la punteggiatura, numeri e caratteri speciali non
        desiderati dal testo
    translator = str.maketrans('', '', string.punctuation + '
        0123456789')
    text_no_punctuation = text.translate(translator)

    # Tokenizza il testo e rimuove le stopwords, quindi lemmatizza
    tokens = word_tokenize(text_no_punctuation)
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [
        lemmatizer.lemmatize(token.lower()) for token in tokens
        if token.lower() not in stop_words
    ]
    return lemmatized_tokens

# Applicazione della funzione al DataFrame
# Assumendo che house_episodes_df sia il DataFrame e contenga una
    colonna 'plot'
house_episodes_df['normalized_plot'] = house_episodes_df['plot'].
    apply(tokenize_normalize)
house_episodes_df['normalized_plot_text'] = house_episodes_df['
    normalized_plot'].apply(' '.join)
```

In questo frammento di codice, importo le librerie necessarie per il trattamento del testo. La funzione `word_tokenize` serve per suddividere il testo in singole parole o token. Utilizzo `stopwords` per filtrare parole come "the", "is", e "in", che sono frequenti in inglese ma portano poco valore semantico. `WordNetLemmatizer` è utilizzato per ridurre le parole alla loro forma base o lemma, una pratica comune nel preprocessing del testo per ridurre la complessità e migliorare le performance degli algoritmi di analisi. Infine, importo `string` per accedere a una serie di caratteri predefiniti, utili per la rimozione di punteggiatura.
Carico l'elenco delle stopwords in inglese e lo converto in un set, che è una struttura dati ottimizzata per rapide operazioni di verifica dell'appartenenza, utile per filtrare rapidamente queste parole durante l'analisi del testo.
Questa funzione `tokenize_normalize` è progettata per processare il testo in più fasi. Inizialmente, rimuovo la punteggiatura e i numeri per purificare il testo da elementi non necessari che potrebbero influenzare negativamente l'analisi. Successivamente, tokenizzo il testo pulito in parole singole. Dopo la tokenizzazione, applico la lemmatizzazione, che trasforma ogni parola nella sua forma base tenendo conto del contesto, e allo stesso tempo filtro le stopwords. Questo processo produce un elenco di lemmi significativi che sono pronti per ulteriori analisi.

Qui applico la funzione `tokenize_normalize` a una colonna del DataFrame, presumibilmente contenente i sommari degli episodi di una serie TV. Ogni sommario viene processato e il risultato viene salvato in una nuova colonna `normalized_plot`. Successivamente, converto gli elenchi di lemmi in stringhe continue per facilitare eventuali analisi testuali future e le memorizzo in un'altra colonna `normalized_plot_text`.
(A causa di un errore che non sono riuscito a risolvere, non ho utilizzato il POS Tagging, nonostante migliorasse la precisione della lemmatizzazione).

1.2 Creazione Modelli Embedding e Similarità Coseno

1.2.1 WORD2VEC

Word2vec è una semplice rete neurale artificiale a due strati progettata per elaborare il linguaggio naturale, l'algoritmo richiede in ingresso un corpus e restituisce un insieme di vettori che rappresentano la distribuzione semantica delle parole nel testo. Per ogni parola contenuta nel corpus, in modo univoco, viene costruito un vettore in modo da rappresentarla come un punto nello spazio multidimensionale creato. In questo spazio le parole saranno più vicine se riconosciute come semanticamente più simili. Per capire come Word2vec possa produrre word embedding è necessario comprendere le architetture CBOW e Skip-Gram. Modello CBOW: Questo metodo prende il contesto di ogni parola come input e cerca di prevedere la parola corrispondente a quel contesto. Modello Skip-Gram: è un'architettura di rete neurale progettata per prevedere le parole di contesto circostanti (vicine) di una parola obiettivo (centrale) in un corpus di testo. In altre parole, dato un termine centrale, Skip-Gram cerca di predire le parole che appaiono intorno a quel termine entro una certa finestra di contesto.
Entrambi hanno i loro vantaggi e svantaggi. Secondo Mikolov, Skip Gram funziona bene con piccole quantità di dati e rappresenta bene le parole rare.
D'altra parte, CBOW è più veloce e ha rappresentazioni migliori per le parole più frequenti.
Utilizzo il modello Word2Vec basato su Skip-Gram per generare rappresentazioni vettoriali delle parole nei sommari degli episodi della serie "Dr. House". Successivamente, calcolo la similarità coseno tra gli episodi per identificare quelli più simili tra loro. La colonna 'normalized-plot-text' del DataFrame house-episodes-df contiene i sommari degli episodi pre-elaborati, che vengono tokenizzati in parole utilizzando la funzione word-tokenize della libreria nltk. Successivamente, creo un modello Word2Vec utilizzando l'architettura Skip-Gram, addestrato sui dati tokenizzati per ottenere rappresentazioni vettoriali dense per ogni parola. I parametri del modello includono una dimensione del vettore (vector-size) di 100, una finestra di contesto (window) di 5, un numero minimo di occorrenze della parola (min-count) pari a 1, e l'uso dell'algoritmo Skip-Gram indicato da sg=1.
Per ogni episodio, calcolo l'embedding medio delle parole presenti utilizzando il modello Word2Vec addestrato; la funzione average-vector calcola la media dei vettori delle parole presenti nel vocabolario del modello. Una volta ottenuti gli embeddings medi per ciascun episodio, calcolo la matrice di similarità coseno tra tutti gli episodi utilizzando la funzione cosine-similarity della libreria sklearn. Questa matrice indica quanto ogni episodio è simile agli altri basandosi sulla rappresentazione vettoriale delle parole. La matrice di similarità coseno viene poi convertita in un DataFrame pandas per una migliore visualizzazione. Utilizzo quindi la funzione heatmap della libreria seaborn per creare una mappa di calore (heatmap) che rappresenta graficamente le similarità tra gli episodi.
La mappa di calore risultante fornisce una chiara visualizzazione della simi-

larità tra gli episodi di "Dr. House". Gli episodi con alta similarità coseno sono rappresentati da colori più caldi (ad esempio, rosso), mentre quelli con bassa similarità sono rappresentati da colori più freddi (ad esempio, blu). Questo tipo di analisi aiuta a identificare episodi con trame simili o tematiche ricorrenti. L'uso di Word2Vec con il modello Skip-Gram ha permesso di creare rappresentazioni semantiche efficaci dei sommari degli episodi. La similarità coseno tra questi embeddings ha offerto una panoramica delle relazioni semantiche tra episodi diversi, evidenziando quelli con contenuti tematici simili. Questo approccio può essere ulteriormente raffinato con tecniche di pre-elaborazione più avanzate e modelli di embedding di parole più sofisticati per migliorare la precisione e l'accuratezza dei risultati.

```
# Assumiamo che 'normalized_plot_text' sia la colonna dei sommari
    puliti
data = [word_tokenize(text) for text in house_episodes_df['
    normalized_plot_text']]
# Crea il modello Word2Vec usando Skip-gram
model_sg = gensim.models.Word2Vec(data, vector_size=100, window=5,
    min_count=1, sg=1)

def average_vector(words, model):
    valid_words = [word for word in words if word in model.wv.
        key_to_index]
    if valid_words:
        return np.mean([model.wv[word] for word in valid_words],
            axis=0)
    else:
        return np.zeros(model.vector_size)

# Prepariamo la matrice degli embeddings medi per ogni episodio
    utilizzando il modello Skip-gram
episode_embeddings_sg = np.array([average_vector(episode, model_sg)
     for episode in data])
# Calcola la matrice di similarit  coseno tra gli embeddings
similarity_matrix_sg = cosine_similarity(episode_embeddings_sg)
# Converti la matrice di similarit  in un DataFrame per una
    migliore visualizzazione
similarity_df_sg = pd.DataFrame(similarity_matrix_sg, index=
    house_episodes_df.index, columns=house_episodes_df.index)

# Visualizzazione con heatmap per Skip-gram
plt.figure(figsize=(12, 10))
sns.set(font_scale=0.8)  # Ajusta la dimensione del font

heatmap_sg = sns.heatmap(similarity_df_sg, annot=True, cmap='
    coolwarm', fmt=".2f", cbar_kws={'label': 'Cosine Similarity'})
heatmap_sg.set_xticklabels(heatmap_sg.get_xticklabels(), rotation
    =90)
heatmap_sg.set_yticklabels(heatmap_sg.get_yticklabels(), rotation
    =0)
plt.title('Similarit  Cosine Word2Vec (Skip-gram) Tra Episodi di
    Dr. House')
plt.tight_layout()
plt.show()
```

1.2.2 TF-IDF

In questa analisi, utilizzo il metodo TF-IDF (Term Frequency-Inverse Document Frequency) per valutare la similarità tra i sommari degli episodi della serie "Dr. House". Il TF-IDF è una tecnica di vettorizzazione che quantifica l'importanza di una parola in un documento rispetto a un corpus di documenti. Successivamente, calcolo la similarità coseno tra i vettori TF-IDF degli episodi per identificare quelli con contenuti più simili.
Partendo dai dati, assumiamo che la colonna 'normalized-plot-text' del DataFrame house-episodes-df contenga i sommari degli episodi già pre-elaborati e normalizzati. Creiamo un'istanza di TfidfVectorizer, una classe della libreria sklearn che trasforma il testo in una matrice di feature TF-IDF. Questa matrice rappresenta il peso delle parole nei diversi episodi in base alla loro frequenza e importanza nel corpus.
Applicando il vectorizer ai sommari puliti, otteniamo la matrice TF-IDF, dove ogni riga rappresenta un episodio e ogni colonna una parola del vocabolario, con valori corrispondenti al peso TF-IDF di quella parola in quel particolare episodio. Una volta ottenuta la matrice TF-IDF, calcolo la similarità coseno tra tutti gli episodi utilizzando la funzione cosine-similarity di sklearn. La similarità coseno misura la distanza angolare tra i vettori nello spazio delle caratteristiche, fornendo un indicatore di quanto due episodi siano simili in termini di contenuto testuale.
La matrice di similarità coseno risultante viene convertita in un DataFrame pandas per facilitare la visualizzazione e l'interpretazione. Utilizzo poi la funzione heatmap della libreria seaborn per creare una mappa di calore (heatmap) che rappresenta graficamente la similarità tra gli episodi. La heatmap visualizza la similarità tra episodi usando una scala di colori, dove i colori più caldi (ad esempio, rosso) indicano una similarità maggiore e i colori più freddi (ad esempio, blu) indicano una minor similarità.
I risultati della mappa di calore forniscono una chiara visualizzazione delle relazioni di similarità tra i diversi episodi di "Dr. House". Gli episodi che mostrano una maggiore similarità coseno tendono a condividere termini e frasi comuni nei loro sommari, suggerendo tematiche o sviluppi narrativi simili. Questo tipo di analisi è utile per identificare episodi con contenuti tematici ricorrenti o per esplorare la coesione narrativa tra episodi diversi.
In conclusione, l'uso del TF-IDF per rappresentare il testo degli episodi e del calcolo della similarità coseno offre un modo efficace per esplorare la similarità semantica tra gli episodi della serie. Questo approccio può essere ulteriormente migliorato ottimizzando i parametri del TF-IDF o applicando tecniche di riduzione della dimensionalità per migliorare la qualità delle rappresentazioni e la precisione delle similarità calcolate.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Assumiamo che 'normalized_plot_text' sia la colonna con i sommari
    puliti
# Crea un'istanza di TfidfVectorizer
# Personalizza i parametri se necessario (es. max_df, min_df)
vectorizer = TfidfVectorizer()

# Applica il vectorizer ai dati puliti per creare la matrice TF-IDF
tfidf_matrix = vectorizer.fit_transform(house_episodes_df['
    normalized_plot_text'])

# Calcola la matrice di similarit  a coseno utilizzando la matrice
    TF-IDF
similarity_matrix_tfidf = cosine_similarity(tfidf_matrix)

# Converti la matrice di similarit  in un DataFrame per una
    migliore visualizzazione
similarity_df_tfidf = pd.DataFrame(similarity_matrix_tfidf, index=
    house_episodes_df.index, columns=house_episodes_df.index)

# Configurazione e visualizzazione della heatmap della similarit
plt.figure(figsize=(12, 10))
sns.set(font_scale=0.8)  # Ajusta la dimensione del font

# Visualizza la heatmap
heatmap = sns.heatmap(similarity_df_tfidf, annot=True, cmap='
    coolwarm', fmt=".2f", cbar_kws={'label': 'Cosine Similarity'})
heatmap.set_xticklabels(heatmap.get_xticklabels(), rotation=90)
heatmap.set_yticklabels(heatmap.get_yticklabels(), rotation=0)

plt.title('Similarit  TF-IDF Cosine Tra Episodi di Dr. House')
plt.tight_layout()
plt.show()
```

1.2.3 Bag of Words

In questa analisi, utilizzo la tecnica Bag-of-Words (BoW) per rappresentare i sommari degli episodi della serie "Dr. House" e calcolare la similarità tra di essi. La tecnica Bag-of-Words è un metodo di rappresentazione testuale in cui un documento è rappresentato come un vettore di frequenze di parole, ignorando l'ordine delle parole ma tenendo conto della loro presenza o assenza.
Per iniziare, assumo che la colonna 'normalized-plot-text' del DataFrame house-episodes-df contenga i sommari pre-elaborati degli episodi. Utilizzo il CountVectorizer di sklearn, una classe che trasforma il testo in una matrice di conteggi delle parole. Questo processo trasforma ogni sommario in un vettore che rappresenta la frequenza di ogni parola all'interno del sommario stesso. Ogni riga della matrice BoW rappresenta un episodio, mentre ogni colonna rappresenta una parola del vocabolario. I valori della matrice corrispondono al numero di volte in cui ciascuna parola appare in un episodio specifico.
Una volta creata la matrice BoW, calcolo la similarità coseno tra i vettori BoW degli episodi utilizzando la funzione cosine-similarity di sklearn. La similarità coseno misura la distanza angolare tra i vettori nello spazio delle caratteristiche, fornendo una misura di quanto due episodi siano simili in termini di contenuto testuale, basato sulle frequenze delle parole.
La matrice di similarità coseno risultante viene poi convertita in un DataFrame pandas per facilitare la visualizzazione e l'interpretazione dei dati. Successivamente, utilizzo la funzione heatmap della libreria seaborn per creare una mappa di calore (heatmap) che rappresenta graficamente la similarità tra gli episodi. La mappa di calore mostra visivamente la similarità tra episodi usando una scala di colori, dove i colori più caldi (ad esempio, rosso) indicano una similarità maggiore e i colori più freddi (ad esempio, blu) indicano una minor similarità.
I risultati della mappa di calore forniscono una chiara visualizzazione delle relazioni di similarità tra i diversi episodi di "Dr. House". Gli episodi con una similarità coseno elevata tendono a condividere parole comuni nei loro sommari, suggerendo che trattano argomenti simili o sviluppi narrativi ricorrenti. Questo tipo di analisi può essere particolarmente utile per identificare episodi con trame simili o per esplorare come temi specifici vengano trattati in diversi episodi.
In conclusione, l'uso del modello Bag-of-Words per rappresentare il testo degli episodi e del calcolo della similarità coseno offre un modo semplice ed efficace per esplorare la similarità testuale tra gli episodi della serie. Tuttavia, poiché il modello BoW non cattura le relazioni semantiche tra le parole e ignora l'ordine delle parole, può essere meno accurato rispetto a tecniche più avanzate come TF-IDF o Word2Vec nel catturare il significato semantico completo dei sommari degli episodi.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Crea un'istanza di CountVectorizer, puoi personalizzare i
    parametri qui
vectorizer = CountVectorizer()

# Supponiamo che 'normalized_plot_text' sia la colonna con i
    sommari puliti
# Applica il CountVectorizer ai dati per creare la matrice BoW
bow_matrix = vectorizer.fit_transform(house_episodes_df['
    normalized_plot_text'])

# Calcola la matrice di similarit  a coseno utilizzando la matrice
     BoW
similarity_matrix_bow = cosine_similarity(bow_matrix)

# Converti la matrice di similarit  in un DataFrame per una
    migliore visualizzazione
similarity_df_bow = pd.DataFrame(similarity_matrix_bow, index=
    house_episodes_df.index, columns=house_episodes_df.index)

# Configurazione e visualizzazione della heatmap della similarit
plt.figure(figsize=(12, 10))
sns.set(font_scale=0.8)  # Ajusta la dimensione del font
heatmap = sns.heatmap(similarity_df_bow, annot=True, cmap='coolwarm
    ', fmt=".2f", cbar_kws={'label': 'Cosine Similarity'})
heatmap.set_xticklabels(heatmap.get_xticklabels(), rotation=90)
heatmap.set_yticklabels(heatmap.get_yticklabels(), rotation=0)
plt.title('Similarit  Bag-of-Words Cosine Tra Episodi di Dr. House
    ')
plt.tight_layout()
plt.show()
```

1.3 Conclusioni

Dai grafici di similarità coseno forniti per i tre diversi modelli (Bag-of-Words, TF-IDF e Word2Vec con Skip-Gram), posso trarre delle conclusioni dettagliate sulle differenze nell'analisi della similarità tra gli episodi di "Dr. House". Ogni grafico rappresenta una mappa di calore (heatmap) che visualizza la similarità coseno tra gli episodi, con colori più caldi (ad esempio, rosso) che indicano una similarità maggiore e colori più freddi (ad esempio, blu) che indicano una similarità minore.
1. Bag-of-Words (BoW) Similarità Coseno Osservando il grafico del modello Bag-of-Words, noto che la mappa di calore mostra una gamma di valori di similarità coseno che vanno da circa 0.05 a 0.38 tra diversi episodi. Ci sono poche aree con similarità elevata (colori più caldi), il che suggerisce che pochi episodi condividono un alto numero di parole comuni basato sulla semplice frequenza. La maggior parte dei valori di similarità sono relativamente bassi, indicando che il modello BoW non cattura molto bene la semantica o il contesto dei contenuti degli episodi, poiché considera solo la presenza o l'assenza di parole, senza pesare la loro importanza.
Conclusione sul Bag-of-Words: Il modello Bag-of-Words è semplice e basato sulla frequenza delle parole senza considerare il loro significato o contesto. Questo limita la sua capacità di catturare la semantica e porta a valori di similarità piuttosto bassi tra gli episodi. Episodi che condividono parole comuni, indipendentemente dalla loro rilevanza, appaiono più simili, ma il modello è poco efficace nel rilevare similarità semantiche più profonde.
2. TF-IDF (Term Frequency-Inverse Document Frequency) Similarità Coseno Osservando il grafico del modello TF-IDF, noto che la mappa di calore presenta valori di similarità generalmente più bassi rispetto a BoW, con la maggior parte dei valori che si aggirano tra 0.01 e 0.13. Rispetto a BoW, il modello TF-IDF tende a ridurre l'impatto delle parole comuni (che sono meno informative) aumentando il peso delle parole più rare e distintive, che possono essere più indicative del contenuto semantico. Anche in questo caso, la similarità tra la maggior parte degli episodi è bassa, indicando che gli episodi diversi non condividono molti termini distintivi e importanti con peso alto.
Conclusione sul TF-IDF: Il modello TF-IDF migliora rispetto al modello BoW pesando le parole in base alla loro importanza globale nel corpus, ma i valori di similarità restano bassi, suggerendo che gli episodi hanno contenuti distinti. TF-IDF cattura meglio le parole significative e riduce l'influenza delle parole comuni, ma non riesce a cogliere relazioni semantiche complesse tra parole diverse.
3. Word2Vec (Skip-Gram) Similarità Coseno Osservando il grafico del modello Word2Vec, noto che la mappa di calore con Skip-Gram mostra una gamma di valori di similarità coseno molto più ampia, con molti episodi che mostrano valori di similarità elevata (tra 0.60 e 0.86). Rispetto agli altri due modelli, Word2Vec sembra catturare meglio le similarità semantiche tra gli episodi, riflettendo una comprensione più profonda del significato contestuale delle parole. I valori di similarità più elevati indicano che il modello Word2Vec è in grado

di identificare meglio episodi che, pur non condividendo le stesse parole esatte, condividono parole semanticamente correlate o contesti simili.
Conclusione su Word2Vec: Il modello Word2Vec con Skip-Gram offre una rappresentazione più ricca e semantica del testo, catturando relazioni più profonde tra le parole rispetto ai modelli BoW e TF-IDF. Di conseguenza, fornisce una misura di similarità più significativa tra gli episodi. Questo modello è molto più efficace nel rilevare similitudini basate sul contesto e il significato delle parole, il che spiega i valori di similarità più elevati rispetto ai modelli basati solo sulla frequenza delle parole.
Conclusioni Generali Dall'analisi delle mappe di calore, è evidente che il modello Word2Vec (Skip-Gram) è il più potente tra i tre per catturare la similarità semantica tra gli episodi di "Dr. House". Mentre Bag-of-Words e TF-IDF sono utili per rappresentazioni semplici basate sulla frequenza e importanza delle parole, Word2Vec offre una comprensione più profonda, sfruttando relazioni semantiche e contesti più complessi. Questo rende Word2Vec una scelta preferita per analisi in cui il significato e il contesto delle parole sono critici. Tuttavia, l'efficacia di ciascun modello dipende dall'obiettivo specifico dell'analisi e dalla natura dei dati.

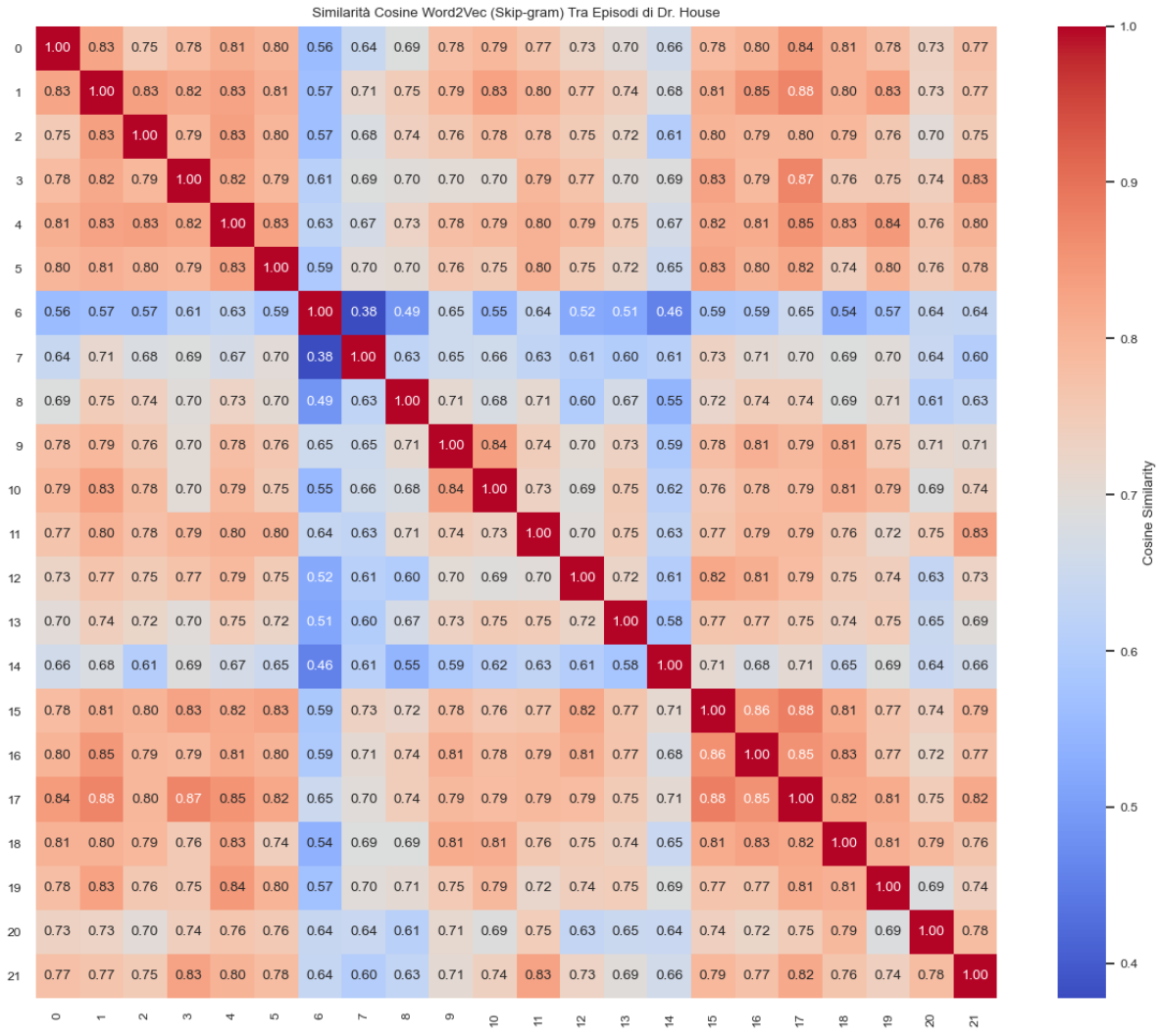

Figure 1: Similarità WORD2VEC Cosine Tra Episodi di Dr. House

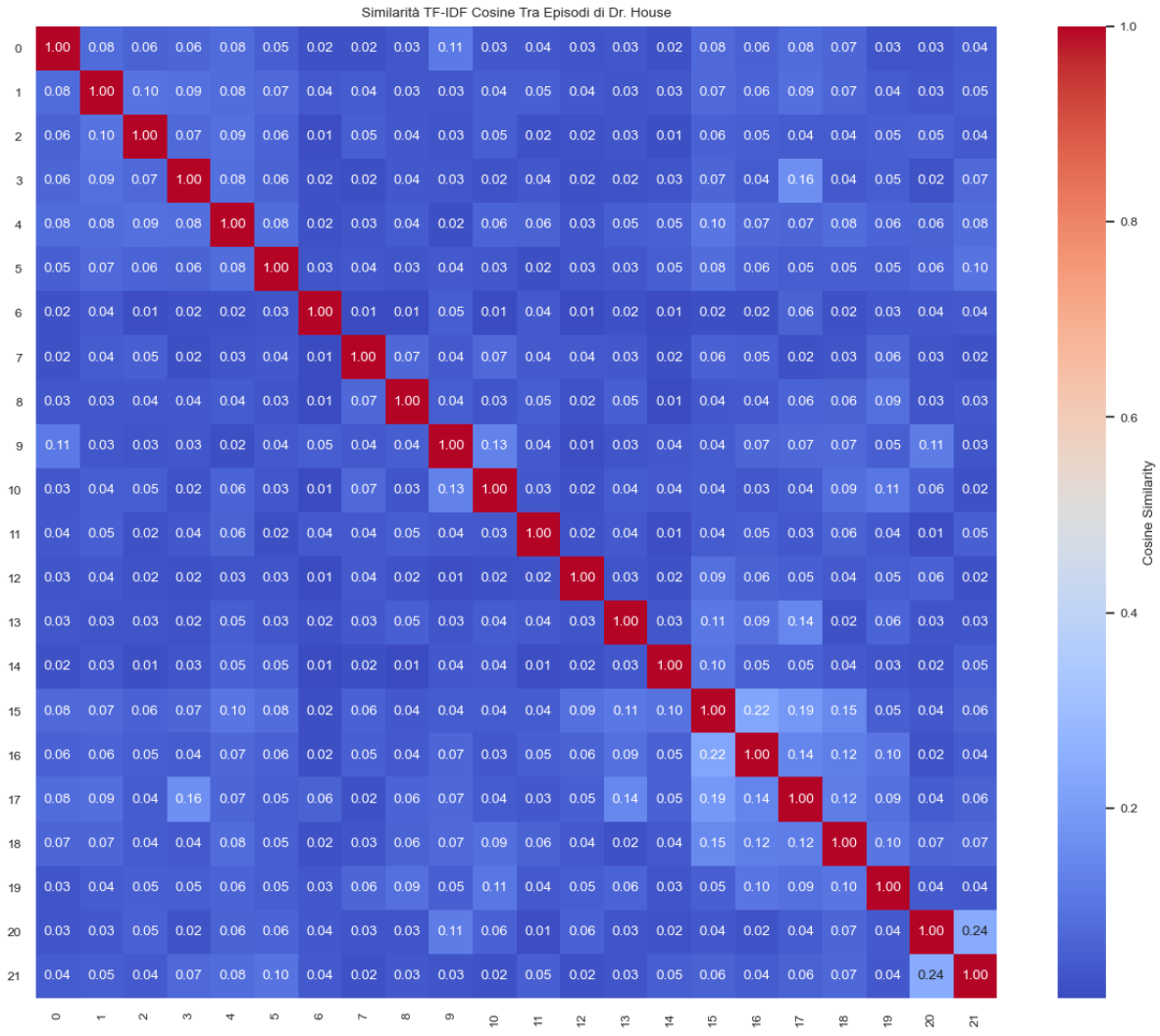

Figure 2: Similarità TF-IDF Cosine Tra Episodi di Dr. House

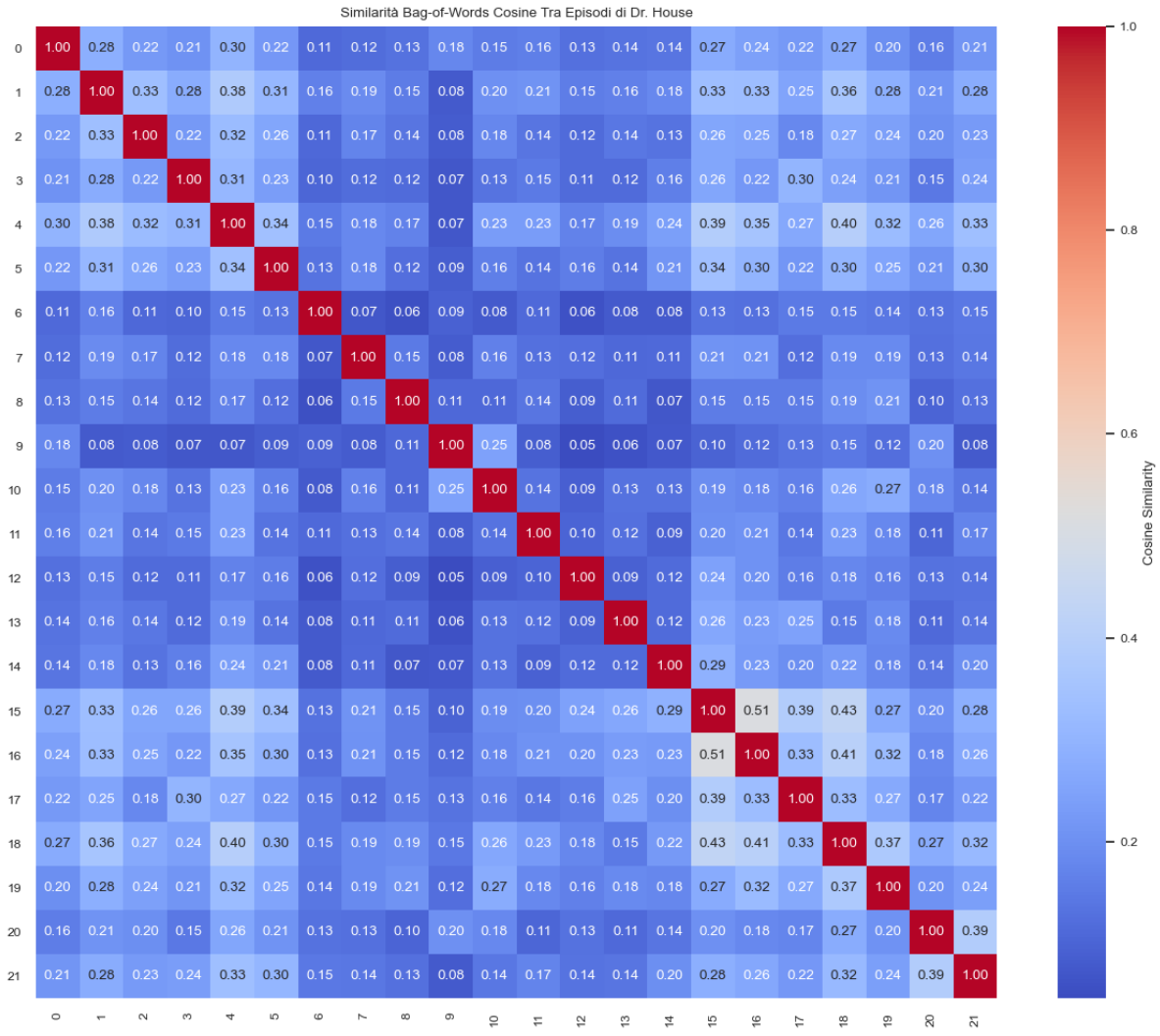

Figure 3: Similarità Bag-of-Words Cosine Tra Episodi di Dr. House