File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python [conda env:base] *

[2]:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from sklearn.metrics.pairwise import cosine_similarity
import ast
import numpy as np
import re
from gensim.models import Word2Vec
df_final = pd.read_csv("oneHot.csv")
df = pd.read_csv("movieData.csv")
merged_combined_df = pd.read_csv("News.csv")
```

[3]:

```
!pip install sentence-transformers
```

```
Requirement already satisfied: sentence-transformers in /opt/anaconda3/lib/python3.12/site-packages (3.4.0)
Requirement already satisfied: transformers<5.0.0,>=4.41.0 in /opt/anaconda3/lib/python3.12/site-packages (from sentence-transformers) (4.45.2)
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.12/site-packages (from sentence-transformers) (4.66.5)
Requirement already satisfied: torch>=1.11.0 in /opt/anaconda3/lib/python3.12/site-packages (from sentence-transformers) (2.4.1)
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/python3.12/site-packages (from sentence-transformers) (1.3.2)
Requirement already satisfied: scipy in /opt/anaconda3/lib/python3.12/site-packages (from sentence-transformers) (1.13.1)
Requirement already satisfied: huggingface-hub>=0.20.0 in /opt/anaconda3/lib/python3.12/site-packages (from sentence-transformers) (0.25.2)
Requirement already satisfied: Pillow in /opt/anaconda3/lib/python3.12/site-packages (from sentence-transformers) (11.0.0)
Requirement already satisfied: filelock in /opt/anaconda3/lib/python3.12/site-packages (from huggingface-hub>=0.20.0->sentence-transformers) (3.13.1)
Requirement already satisfied: fsspec>=2023.5.0 in /opt/anaconda3/lib/python3.12/site-packages (from huggingface-hub>=0.20.0->sentence-transformers) (2024.6.1)
Requirement already satisfied: packaging>=20.9 in /opt/anaconda3/lib/python3.12/site-packages (from huggingface-hub>=0.20.0->sentence-transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /opt/anaconda3/lib/python3.12/site-packages (from huggingface-hub>=0.20.0->sentence-transformers) (6.0.2)
Requirement already satisfied: requests in /opt/anaconda3/lib/python3.12/site-packages (from huggingface-hub>=0.20.0->sentence-transformers) (2.32.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/anaconda3/lib/python3.12/site-packages (from huggingface-hub>=0.20.0->sentence-transformers) (4.12.2)
Requirement already satisfied: sympy in /opt/anaconda3/lib/python3.12/site-packages (from torch>=1.11.0->sentence-transformers) (1.13.3)
```

```
Requirement already satisfied: networkx in /opt/anaconda3/lib/python3.12/site-packages
(from torch>=1.11.0->sentence-transformers) (3.3)
Requirement already satisfied: jinja2 in /opt/anaconda3/lib/python3.12/site-packages (f
rom torch>=1.11.0->sentence-transformers) (3.1.4)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/python3.12/site-package
s (from torch>=1.11.0->sentence-transformers) (75.1.0)
Requirement already satisfied: numpy>=1.17 in /opt/anaconda3/lib/python3.12/site-packag
es (from transformers<5.0.0,>=4.41.0->sentence-transformers) (1.26.4)
Requirement already satisfied: regex!=2019.12.17 in /opt/anaconda3/lib/python3.12/site-
packages (from transformers<5.0.0,>=4.41.0->sentence-transformers) (2024.9.11)
Requirement already satisfied: safetensors>=0.4.1 in /opt/anaconda3/lib/python3.12/site
-packages (from transformers<5.0.0,>=4.41.0->sentence-transformers) (0.4.5)
Requirement already satisfied: tokenizers<0.21,>=0.20 in /opt/anaconda3/lib/python3.12/
site-packages (from transformers<5.0.0,>=4.41.0->sentence-transformers) (0.20.1)
Requirement already satisfied: joblib>=1.1.1 in /opt/anaconda3/lib/python3.12/site-pack
ages (from scikit-learn->sentence-transformers) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.12/si
te-packages (from scikit-learn->sentence-transformers) (3.5.0)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/anaconda3/lib/python3.12/site-pa
ckages (from jinja2->torch>=1.11.0->sentence-transformers) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/lib/python3.1
2/site-packages (from requests->huggingface-hub>=0.20.0->sentence-transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.12/site-packa
ges (from requests->huggingface-hub>=0.20.0->sentence-transformers) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/python3.12/site
-packages (from requests->huggingface-hub>=0.20.0->sentence-transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.12/site
-packages (from requests->huggingface-hub>=0.20.0->sentence-transformers) (2024.12.14)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /opt/anaconda3/lib/python3.12/site
-packages (from sympy->torch>=1.11.0->sentence-transformers) (1.3.0)
```

[11]:

```python
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import SentenceTransformer

# Initialize model once (outside the function for efficiency).
bert_model = SentenceTransformer('all-MiniLM-L6-v2')

def recommend_movies(user_inputs, df, df_final):

    # 1) Identify the "input movies" by partial matching on title
    input_descriptions = []
    input_genres_set = set()  # Will store all genres from matched input movies

    for segment in user_inputs:
        # Find rows where 'title' contains the segment (case-insensitive)
        matched_rows = df[df['title'].str.contains(segment, case=False, na=False)]

        # Collect their descriptions (original text, for user-friendly display)
        for desc in matched_rows['description'].fillna(''):
            input_descriptions.append(desc)

        # Collect their mapped genres
```

```
    # Collect their mapped genres
    for mg_list in matched_rows['genres']:
        if isinstance(mg_list, list):
            input_genres_set.update(mg_list)

# If no movies were matched, return early
if not input_descriptions:
    return {
        "Recommended Movies": [],
        "Recommended Movies Descriptions": [],
        "Recommended Movies Genres": [],
        "Similarity Scores": [],
        "Input Genres": [],
        "All Descriptions": [],
        "All Unique Genres": [],
        "Message": "No movies matched the provided title segments."
    }

# 2) Create a combined embedding from all matched input movies
#    We'll use the stemmed descriptions for embedding
matched_stemmed = []
for segment in user_inputs:
    # This time gather the 'description_stemmed' for matched rows
    matched_rows = df[df['title'].str.contains(segment, case=False, na=False)]
    for d_stemmed in matched_rows['description_stemmed'].fillna(''):
        matched_stemmed.append(d_stemmed)

# Encode and average
user_desc_embs = bert_model.encode(matched_stemmed)
user_input_emb = np.mean(user_desc_embs, axis=0)  # shape: (embedding_dim,)

# 3) Compute similarity for all movies in df
#    (A) BERT embeddings
all_stemmed = df['description_stemmed'].fillna('').tolist()
movie_desc_embs = bert_model.encode(all_stemmed)  # shape: (num_movies, embedding_d

semantic_similarity = cosine_similarity(
    [user_input_emb],       # shape: (1, embedding_dim)
    movie_desc_embs         # shape: (num_movies, embedding_dim)
).flatten()                  # shape: (num_movies,)

#    (B) Categorical similarity
movie_genres_sets = df['genres'].apply(
    lambda glist: set(glist) if isinstance(glist, list) else set()
)
categorical_sim_list = []
for mg_set in movie_genres_sets:
    union_size = len(input_genres_set | mg_set)
    if union_size == 0:
        categorical_sim_list.append(0.0)
    else:
        intersection_size = len(input_genres_set & mg_set)
```

```
            categorical_sim_list.append(intersection_size / union_size)

    categorical_similarity = np.array(categorical_sim_list)

    # Combine them
    combined_similarity = 0.7 * semantic_similarity + 0.3 * categorical_similarity

    # 4) Find top 5 recommended movies
    df['similarity'] = combined_similarity
    recommendations = df.nlargest(5, 'similarity')

    # 5) Build the output data
    recommended_movies = recommendations['title'].tolist()
    recommended_descriptions = recommendations['description'].fillna('').tolist()
    recommended_genres_lists = recommendations['genres'].tolist()
    similarity_scores = recommendations['similarity'].tolist()

    # 6) Build the combined lists requested:
    #    (A) All descriptions: input movies + recommended movies
    all_descriptions = input_descriptions + recommended_descriptions

    #    (B) All unique genres: union of input genres + recommended genres
    recommended_genres_set = set()
    for mg_list in recommended_genres_lists:
        if isinstance(mg_list, list):
            recommended_genres_set.update(mg_list)
    all_unique_genres = list(input_genres_set.union(recommended_genres_set))

    # 7) Return everything in a dictionary
    return {
        "Recommended Movies": recommended_movies,
        "Recommended Movies Descriptions": recommended_descriptions,
        "Recommended Movies Genres": recommended_genres_lists,
        "Similarity Scores": similarity_scores,
        "Input Genres": list(input_genres_set),
        "All Descriptions": all_descriptions,
        "All Unique Genres": all_unique_genres
    }


# Example usage
user_input = input()
user_inputs = [title.strip() for title in user_input.split(",") if title.strip()]
result = recommend_movies(user_inputs, df, df_final)

# Display results
print("Recommended Movies:", result["Recommended Movies"])
print("Recommended Movies Descriptions:", result["Recommended Movies Descriptions"])
print("Recommended Movies Genres:", result["Recommended Movies Genres"])
print("Similarity Scores:", result["Similarity Scores"])
print("Input Genres:", result["Input Genres"])
print("All Descriptions:", result["All Descriptions"])
```

```
print("All Unique Genres:", result["All Unique Genres"])
```

Loading widget...

```
 Titan, Avatar, Harry Potter


Recommended Movies: ['Shingeki no Kyojin Picture Drama/Attack on Titan: Chibi Theate
r - Fly, Cadets, Fly!/「進撃の巨人」ちみキャラ劇場"とんでけ! 訓練兵団"', 'Shin Koihime†Muso
u/真恋姫†無双 ～乙女繚乱☆三国志演義～', 'Hokuto no Ken/Fist of the North Star/北斗の拳',
'X/1999/X - The Movie/エックス, X/1999', 'Duel Masters VSRF/Duel Masters Versus Revolu
tion Final/デュエル・マスターズVSRF']
```

[12]:

```
#all unique categories
unique_categories = merged_combined_df['category'].dropna().unique()
print(unique_categories)
```

```
['Americas' 'Reality Check' 'Asia' 'Europe' 'Middle East' 'World' 'Africa'
 'UK/Local' 'Politics' 'Other' 'Sports' 'Business' 'Technology'
 'Family & Education' 'Science & Environment' 'Oceania']
```

[13]:

```
df['mapped_categories'].head()
```

[13]:

```
0    ['Americas', 'Adventure', 'World', 'Science & ...
1    ['Americas', 'Adventure', 'Science & Environme...
2    ['Americas', 'Adventure', 'Science & Environme...
3    ['Americas', 'Adventure', 'World', 'Culture', ...
4                              ['Adventure', 'World']
Name: mapped_categories, dtype: object
```

[14]:

```
import ast  # Import Abstract Syntax Tree for safe string-to-list conversion

recommended_categories = set()  # Use a set to avoid duplication
recommended_movies = result["Recommended Movies"]

for title in recommended_movies:
    # Find the index of the movie in the DataFrame
    matches = df[df['title'] == title]
    if not matches.empty:
```

```
        index = matches.index[0]

    # Convert string representation of a list into an actual list if needed
    raw_categories = df.loc[index, 'mapped_categories']
    if isinstance(raw_categories, str):
        try:
            raw_categories = ast.literal_eval(raw_categories)  # Safely convert string
        except (ValueError, SyntaxError):
            raw_categories = []  # If conversion fails, use an empty list

    # Filter out 'Other' and add the rest to the set
    recommended_categories.update(
        category for category in raw_categories if category != "Other"
    )

# Optionally, convert the set back to a list if needed
recommended_categories = list(recommended_categories)

# Print the results
print("Recommended Categories:", recommended_categories)
```

```
Recommended Categories: ['Science & Environment', 'Americas', 'World', 'Asia', 'Adventu
re', 'Culture', 'Entertainment', 'Thriller']
```

[15]:

```
filtered_news = merged_combined_df[merged_combined_df['category'].isin(recommended_cate

# Print and store each news title and link
news_list = []
for _, row in filtered_news.iterrows():
    # print(f"Title: {row['title']}, Link: {row['link']}")
    news_list.append({'title': row['title'], 'category':row['category'], 'link': row['l

# Optionally store the filtered news in a DataFrame or save it
filtered_news_df = pd.DataFrame(news_list)
filtered_news_df.head()
```

[15]:

	title	category	link
0	How US election fraud claims changed as Trump ...	Americas	https://www.bbc.com/news/articles/cy9j8r8gg0do
1	Would Donald Trump's tariffs hurt US consumers...	Americas	https://www.bbc.com/news/articles/c20myx1erl6o
2	BBC Verify on claims of Pennsylvania voter fra...	Americas	https://www.bbc.com/news/videos/c9wrp5gnqwvo
3	Ros Atkins on... Harris's struggle to distance...	Americas	https://www.bbc.com/news/videos/cpqdl1r3p0yo

4	Biden condemns misinformation about hurricane ...	Americas	https://www.bbc.com/news/articles/czd1q9q9gl9o

[32]:

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# 1. ユーザーが入力した映画タイトルに部分一致する映画を取得（例: user_inputs = ['Titan', 'Avatar'
matched_rows = df[df['title'].str.contains('|'.join(user_inputs), case=False, na=False)

# 2. description_stemmed を使って連結
input_descriptions = matched_rows['description_stemmed'].dropna().tolist()
if not input_descriptions:
    input_descriptions=matched_rows['title'].dropna().tolist()

combined_description = " ".join(input_descriptions)
```