In-Class Assignment: Text Similarity & Recommendation Systems

DATA 5420/6420

Name: Dallin Moore

In this in-class assignment we will utilize two different implementations of similarity to create a content-based and query-based movie recommender. We will first utilize content-based filtering in order to find movies with similar plot descriptions so that we can then recommend like movies. Then we will create a second recommeder that uses a search query to find the most relevant movie recommendations!

The dataset we will be working with can be pulled from Kaggle [here](), which contains metadata of 5,000 movies from The Movie Database (TMDB), an open-source, editable database for movies and TV shows.

Let's bring in the `tmdb_5000_movies.csv` file and take a look at its contents, as well as load our necessary dependencies.

```
import pandas as pd
import nltk
import re
import numpy as np

nltk.download('punkt')
nltk.download('stopwords')

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

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
df = pd.read_csv('/content/cleaned_tmdb_5000_movies.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4802 entries, 0 to 4801
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
```

```
 ---  ------                --------------  -----
  0   budget                4802 non-null   int64
  1   genres                4774 non-null   object
  2   homepage              1712 non-null   object
  3   id                    4802 non-null   int64
  4   keywords              4802 non-null   object
  5   original_language     4802 non-null   object
  6   original_title        4802 non-null   object
  7   overview              4799 non-null   object
  8   popularity            4802 non-null   float64
  9   production_companies  4802 non-null   object
  10  production_countries  4802 non-null   object
  11  release_date          4801 non-null   object
  12  revenue               4802 non-null   int64
  13  runtime               4800 non-null   float64
  14  spoken_languages      4802 non-null   object
  15  status                4802 non-null   object
  16  tagline               3958 non-null   object
  17  title                 4802 non-null   object
  18  vote_average          4802 non-null   float64
  19  vote_count            4802 non-null   int64
dtypes: float64(3), int64(4), object(13)
memory usage: 750.4+ KB
```

Amongst our columns of interest `genres`, `tagline`, `overview` and `title` do we have missing data?

Yes, there is missing data for overview and tagline. There is also missing data for genres, but not as many instances of missing data.

Building a Movie Recommender System

We will take the following steps to build the recommender system:

1) Prep the dataframe

2) Preprocess the text

3) Feature Engineering

4) Document Similairty Computation

5) Find top similar movies

6) Create a movie recommender function

1) Prep the Dataframe

Let's take a few data preparation steps, including dropping out unneccesary columns (we just care about text columns in this case), getting rid of null values, and merging the `tagline` column with the `overview` column as a plot `description` column.

```
df = df[['title', 'tagline', 'overview', 'genres', 'popularity']]
df['description'] = df['tagline'].map(str) + ' ' + df['overview'].map(str) + ' ' + df['genre
df.head()
```

```
<ipython-input-19-82b7ec14d6ce>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
  df['description'] = df['tagline'].map(str) + ' ' + df['overview'].map(str) + ' ' + df[
```

	title	tagline	overview	genres	popularity	description
0	Avatar	Enter the World of Pandora.	In the 22nd century, a paraplegic Marine is di...	Action, Adventure, Fantasy, Science Fiction	150.437577	Enter the World of Pandora. In the 22nd centur...
1	Pirates of the Caribbean: At World's End	At the end of the world, the adventure begins.	Captain Barbossa, long believed to be dead, ha...	Adventure, Fantasy, Action	139.082615	At the end of the world, the adventure begins....
		A Plan No	A cryptic message	Action		A Plan No One

Next steps: View recommended plots

```
df.tagline.fillna('',inplace=True) # add empty strings
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4802 entries, 0 to 4801
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   title        4802 non-null   object 
 1   tagline      4802 non-null   object 
 2   overview     4799 non-null   object 
 3   genres       4774 non-null   object 
 4   popularity   4802 non-null   float64
 5   description  4802 non-null   object 
dtypes: float64(1), object(5)
memory usage: 225.2+ KB
```

```
df.dropna(inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4771 entries, 0 to 4801
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   title        4771 non-null   object 
 1   tagline      4771 non-null   object 
 2   overview     4771 non-null   object 
 3   genres       4771 non-null   object 
 4   popularity   4771 non-null   float64
 5   description  4771 non-null   object 
dtypes: float64(1), object(5)
memory usage: 260.9+ KB
```

```
df.head()
```

	title	tagline	overview	genres	popularity	description
0	Avatar	Enter the World of Pandora.	In the 22nd century, a paraplegic Marine is di...	Action, Adventure, Fantasy, Science Fiction	150.437577	Enter the World of Pandora. In the 22nd centur...
1	Pirates of the Caribbean: At World's End	At the end of the world, the adventure begins.	Captain Barbossa, long believed to be dead, ha...	Adventure, Fantasy, Action	139.082615	At the end of the world, the adventure begins....
2	Spectre	A Plan No One	A cryptic message from Bond's	Action, Adventure	107.376788	A Plan No One Escapes A

Next steps: View recommended plots

```
df['description'].loc[0]
```

```
'Enter the World of Pandora. In the 22nd century, a paraplegic Marine is dispatched to
the moon Pandora on a unique mission, but becomes torn between following orders and pro
tecting an alien civilization. Action, Adventure, Fantasy, Science Fiction'
```

2) Preprocess the Text

In this next step we will define a normalize document function since this isn't particulary complicated text. Let's think about what steps we need to take in cleaning and normalization, and which we don't.

```
stop_words = nltk.corpus.stopwords.words('english')

# add comments
def normalize_document(doc):
    doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I|re.A)
    doc = doc.lower()
    doc = doc.strip()
    tokens = nltk.word_tokenize(doc)
    filtered_tokens = [token for token in tokens if token not in stop_words]
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)

norm_corpus = normalize_corpus(list(df['description'])) # add input
len(norm_corpus)
```

```
4771
```

```
norm_corpus[0:2]
```

```
array(['enter world pandora 22nd century paraplegic marine dispatched moon pandora
unique mission becomes torn following orders protecting alien civilization action
adventure fantasy science fiction',
       'end world adventure begins captain barbossa long believed dead come back life
headed edge earth turner elizabeth swann nothing quite seems adventure fantasy
action'],
      dtype='<U823')
```

3) Feature Engineering - Extract TF-IDF Features

For this task we will use TFIDF features, and will utilize the `TfidfVectorizer` from `sklearn`.

```
# set parameters for tf-idf for unigrams and bigrams
tf = TfidfVectorizer(ngram_range=(1,2),min_df=2, max_df=0.8)
# extract tfidf features from norm_corpus
tfidf_matrix = tf.fit_transform(norm_corpus)
tfidf_matrix.shape
```

```
(4771, 21385)
```

4a) Compute Pairwise Document Similarity - Cosine Similarity

We'll run through calculating document similarity first with cosine similarity, then we'll repeat the process with our second distance metric.

```
doc_sim = cosine_similarity(tfidf_matrix)
doc_sim_df = pd.DataFrame(doc_sim)                                                # take c
doc_sim_df.head()
```

	0	1	2	3	4	5	6	7	8
0	1.000000	0.064992	0.023083	0.020625	0.069052	0.053474	0.004955	0.069593	0.037559
1	0.064992	1.000000	0.028223	0.003318	0.052748	0.049763	0.010109	0.038441	0.075137
2	0.023083	0.028223	1.000000	0.006774	0.017930	0.018678	0.004266	0.043630	0.019999
3	0.020625	0.003318	0.006774	1.000000	0.010695	0.002631	0.015687	0.026388	0.025063
4	0.069052	0.052748	0.017930	0.010695	1.000000	0.016853	0.025817	0.077786	0.005009

5 rows × 4771 columns

5a) Find Top Similar Movies for a Sample Movie

Our next step will be getting the list of movies from our `df['title']` column, and then examining the top most similar movies for a sample movie choice. Let's see what comes up for the movie *Interstellar* (or you can choose a different one).

We'll take the following steps:

- Create a list of our movie titles
- Locate our sample movie's index number
- Save movie similarity values
- Extract the top 5 most similar to our sample movie
- Print out their names

```
# saving all the unique movie titles to a list
movie_list = df['title'].values
movie_list
```

```
array(['Avatar', "Pirates of the Caribbean: At World's End", 'Spectre',
       ..., 'Newlyweds', 'Signed, Sealed, Delivered', 'My Date with Drew'],
      dtype=object)
```

```
# extracted the index number for the sample movie
movie_idx = np.where(movie_list == "The Martian")[0][0]
movie_idx
```

```
270
```

```
# extracting the similarity scores associated with the sample movie
movie_similarities = doc_sim_df.iloc[movie_idx].values
movie_similarities
```

```
array([0.0417861 , 0.04092668, 0.03534661, ..., 0.        , 0.01132623,
       0.        ])
```

```
similar_movie_idxs = np.argsort(-movie_similarities)[1:6]
similar_movies = movie_list[similar_movie_idxs]
similar_movies
```

```
array(['The Last Days on Mars', 'Swept Away', 'Red Planet', 'Alive',
       'Mission to Mars'], dtype=object)
```

```
df['overview'].loc[270]
```

```
'During a manned mission to Mars, Astronaut Mark Watney is presumed dead after a fierce
storm and left behind by his crew. But Watney has survived and finds himself stranded a
nd alone on the hostile planet. With only meager supplies, he must draw upon his ingenu
ity, wit and spirit to subsist and find a way to signal to Earth that he is alive.'
```

```
np.where(movie_list == 'The Last Days on Mars')[0][0]
```

```
2963
```

```
df['overview'].loc[2963]
```

```
'Sir Robert Chiltern is a successful Government minister, well-off and with a loving wi
fe. All this is threatened when Mrs Cheveley appears in London with damning evidence of
a past misdeed. Sir Robert turns for help to his friend Lord Goring, an apparently idle
philanderer and the despair of his father. Goring knows the lady of old, and, for him,
```

How did the movie recommender perform? Are the suggestions logical, are some of them more illogical? Of the recommendations that do make sense, what overlap might be being captured between these movies?

All of the movies were obviously about mars, which shouldn't be surprising and shows that the model worked.

6a) Build a movie recommender function

Now let's define a function to recommend the top 5 similar movies for any movie in our dataset:

```
# combine everything done above
def movie_recommender(movie_title, movies=movie_list, doc_sims=doc_sim_df):

    movie_idx = np.where(movies == movie_title)[0][0]

    movie_similarities = doc_sims.iloc[movie_idx].values

    similar_movie_idxs = np.argsort(-movie_similarities)[1:6]

    similar_movies = movies[similar_movie_idxs]

    return print("Based on your interest in", movie_title,"I'd Recommend checking out:", sim
```

```
movie_recommender('Bambi') # input movie
```

```
Based on your interest in Bambi I'd Recommend checking out: ['The Notebook' 'American Dr
 'Love in the Time of Cholera' 'Veer-Zaara']
```

Have a friend/family member/partner try out your movie recommender and report how it performs on their selected movie! Did they find a new movie they'd consider watching?

Maybe. Although I'm not familiar with all of the recomended movies, I was expecting more animated movies and that was not the case.

7) Query-Based Recommendation

We can now apply many of the same steps we just took to create a query-based recommendation system instead of a content-based recommendation system. Here, instead of comparing an input movie against all other movies, we'll compare an input query against all the movies -- all still using cosine similarity!

```
def query_movie_recommender(search_query, movies=movie_list, tfidf_matrix=tfidf_matrix):
    # Transform the search query into its vector form
    query_vector = tf.transform([search_query])

    cosine_similarities = cosine_similarity(query_vector, tfidf_matrix)

    similar_movie_idxs = cosine_similarities[0].argsort()[-5:][::-1]

    similar_movies = movies[similar_movie_idxs]

    return print("Based on your search query, I'd recommend checking out:", similar_movies)
```

```
query_movie_recommender('love story with dogs')
```

```
Based on your search query, I'd recommend checking out: ['All Good Things' 'Never Again'
```

Pratically speaking, how do these approaches to recommendation differ? Would you say one appears to perform better than the other?

Not necessarily. I think that they both have their individual use cases. It might be easier to use a movie to create more recomendations because you already know what that movie is about so there could be less guessing. The query is nice if you want something completely new however. If you don't know any movies like what you want.

Try this out with a friend/partner/family member and report back on the success of your recommendations!

They were very impressed! There are lots of recomendations that I had never heard of and it was fun to use.