Lab 4: Building a movie recommendation system using similarity

Background:

- We learned about how W2V and movie recommender work, let's try to put them in practice

Objectives:

```
* Item Recommender based on movie descriptions
* User Recommender based on movie reviews
```

Note:

```
* This notebook is mostly additional knowledge
* not needed for any homework assignment
```

In [2]:
```
from IPython.display import Image
Image(url= "lab4-1.png", width=400, height=400)
```

Out[2]:

Data:

```
* We obtained two datasets:
    * Meta data for a small set of popular movies
    * IMDB reviews (Here I only used a single movie, you can try adding more movies if you like
 to)
```

In [3]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.metrics.pairwise import linear_kernel, cosine_similarity
```

First, let's look at recommendations based on movie similarities:

In [4]:
```
# the description and tagline columns are the ones we're interested in
movies = pd.read_csv('./small_movie_metadata.csv')
movies.head()
```

Out[4]:

	Unnamed: 0	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	origina
0	0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	['Animation', 'Comedy', 'Family']	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy
1	1	False	NaN	65000000	['Adventure', 'Fantasy', 'Family']	NaN	8844	tt0113497	en	Ju
2	2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	['Romance', 'Comedy']	NaN	15602	tt0113228	en	Gru Ol
3	3	False	NaN	16000000	['Comedy', 'Drama', 'Romance']	NaN	31357	tt0114885	en	Wai I
4	4	False	{'id': 96871, 'name': 'Father of the Bride Col...	0	['Comedy']	NaN	11862	tt0113041	en	Father Bride

5 rows × 27 columns

In [5]:
```
movies.columns
```

Out[5]:
```
Index(['Unnamed: 0', 'adult', 'belongs_to_collection', 'budget', 'genres',
       'homepage', 'id', 'imdb_id', 'original_language', 'original_title',
       'overview', 'popularity', 'poster_path', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'video',
       'vote_average', 'vote_count', 'year', 'description'],
      dtype='object')
```

In [6]:
```
# we have a small dataset of 9000+ movies containing their information
len(movies)
```

Out[6]:
```
9099
```

Here I use TFIDF vectorizer to form the document-term matrix

In [6]:
```
# filling in the null values to ease vectorization process
movies['tagline'] = movies['tagline'].fillna('')
movies['description'] = movies['overview'] + movies['tagline']
movies['description'] = movies['description'].fillna('')
```

In [7]:
```
print(movies['title'][0])
print(movies['description'][0])
```

```
Toy Story
Led by Woody, Andy's toys live happily in his room until Andy's birthday brings Buzz Lightyear onto th
e scene. Afraid of losing his place in Andy's heart, Woody plots against Buzz. But when circumstances
separate Buzz and Woody from their owner, the duo eventually learns to put aside their differences.
```

In [8]:
```
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(movies['description'])
```

In [9]:
```
# again, we have 9099 movies, and 268,124 words in our movie lexicon
tfidf_matrix.shape
```

Out[9]:
```
(9099, 268124)
```

In [10]:
```
tfidf_matrix[1:1,1:2]
```

Out[10]:
```
<0x1 sparse matrix of type '<class 'numpy.float64'>'
	with 0 stored elements in Compressed Sparse Row format>
```

In [11]:
```
#Since we have used the TF-IDF Vectorizer, calculating the Dot Product will directly give us the Cosine
# Therefore, we will use sklearn's linear_kernel instead of cosine_similarities since it is much faster.
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [12]:
```
cosine_sim[0]
```

Out[12]:
```
array([1.        , 0.00680476, 0.        , ..., 0.        , 0.00344913,
       0.        ])
```

In [13]:
```
cosine_sim
```

Out[13]:
```
array([[1.        , 0.00680476, 0.        , ..., 0.        , 0.00344913,
        0.        ],
       [0.00680476, 1.        , 0.01531062, ..., 0.00357057, 0.00762326,
        0.        ],
       [0.        , 0.01531062, 1.        , ..., 0.        , 0.00286535,
        0.00472155],
       ...,
       [0.        , 0.00357057, 0.        , ..., 1.        , 0.07811616,
        0.        ],
       [0.00344913, 0.00762326, 0.00286535, ..., 0.07811616, 1.        ,
        0.        ],
       [0.        , 0.        , 0.00472155, ..., 0.        , 0.        ,
        1.        ]])
```

In [14]:
```
movies = movies.reset_index()
titles = movies['title']
indices = pd.Series(movies.index, index=movies['title'])
```

In [18]:
```python
# get the index of single movie
idx = indices['The Godfather']
idx
```

Out[18]:
```
692
```

In [21]:
```python
# get similarity of all movies compared to The Godfather
cosine_sim[idx]
```

Out[21]:
```
array([0.        , 0.00282713, 0.01327441, ..., 0.        , 0.        ,
       0.00983762])
```

In [33]:
```python
# sort the similarities
sim_scores = list(enumerate(cosine_sim[idx]))
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
# get top 5 most similar
# returns [index, similarity score]
sim_scores[:10]
```

Out[33]:
```
[(692, 1.0),
 (973, 0.2200595178936745),
 (8387, 0.10029390541992986),
 (3509, 0.06761848151322326),
 (4196, 0.06562175180478302),
 (29, 0.05614183229965806),
 (5667, 0.05602843927906767),
 (2412, 0.05502278169436636),
 (1582, 0.05023453567639341),
 (4221, 0.04750779835449127)]
```

In [34]:
```python
sim_scores = sim_scores[:10]
# get the names of the movies according to the indices
movie_indices = [i[0] for i in sim_scores]
titles.iloc[movie_indices]
```

Out[34]:
```
692                The Godfather
973       The Godfather: Part II
8387                  The Family
3509                        Made
4196         Johnny Dangerously
29               Shanghai Triad
5667                        Fury
2412              American Movie
1582     The Godfather: Part III
4221                     8 Women
Name: title, dtype: object
```

In [35]:
```python
# for each movie, we recommend its most similar movies
def get_recommendations(title):
    idx = indices[title]

    # list and sort the cosine similarity values for this movie given the index
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    # the top one is obviously itself, which we should ignore
    # here we just keep the top 30 for convenience
    sim_scores = sim_scores[1:31]
    movie_indices = [i[0] for i in sim_scores]

    #now we locate the indices of the most similar movies, we want to return their names
    return titles.iloc[movie_indices]
```

In [36]:
```
get_recommendations('The Godfather').head(20)
```

Out[36]:
```
973      The Godfather: Part II
8387                 The Family
3509                       Made
4196        Johnny Dangerously
29              Shanghai Triad
5667                       Fury
2412              American Movie
1582    The Godfather: Part III
4221                    8 Women
2159              Summer of Sam
618                     Thinner
3609              Harlem Nights
8816              Run All Night
3288          Jaws: The Revenge
2192           The Color Purple
5406            The Kid Brother
3715                  3 Ninjas
7657           The Tillman Story
3607            Family Business
6398                Renaissance
Name: title, dtype: object
```

In [37]:
```
get_recommendations('Toy Story').head(10)
```

Out[37]:
```
2502                 Toy Story 2
7535                 Toy Story 3
6193    The 40 Year Old Virgin
2547            Man on the Moon
6627               Factory Girl
4702    What's Up, Tiger Lily?
889      Rebel Without a Cause
6554    For Your Consideration
4988           Rivers and Tides
1599                  Condorman
Name: title, dtype: object
```

In [40]:
```
get_recommendations('The Dark Knight').head(10)
```

Out[40]:
```
7931                    The Dark Knight Rises
132                            Batman Forever
1113                           Batman Returns
8227    Batman: The Dark Knight Returns, Part 2
7565                Batman: Under the Red Hood
524                                    Batman
7901                         Batman: Year One
2579              Batman: Mask of the Phantasm
2696                                      JFK
8165    Batman: The Dark Knight Returns, Part 1
Name: title, dtype: object
```

In [41]:
```
get_recommendations('Harry Potter and the Chamber of Secrets').head(10)
```

Out[41]:
```
5390         Harry Potter and the Prisoner of Azkaban
6280              Harry Potter and the Goblet of Fire
7649    Harry Potter and the Deathly Hallows: Part 1
7821    Harry Potter and the Deathly Hallows: Part 2
7257           Harry Potter and the Half-Blood Prince
6720        Harry Potter and the Order of the Phoenix
3806         Harry Potter and the Philosopher's Stone
3840              Porn Star: The Legend of Ron Jeremy
3542                                    The Dead Pool
2903                            The Fighting Seabees
Name: title, dtype: object
```

In [42]:
```
get_recommendations('Iron Man 2').head(10)
```

Out[42]:
```
6928                    Iron Man
8285                  Iron Man 3
8758     Avengers: Age of Ultron
2320               The Dark Half
889          Rebel Without a Cause
8201              The Guilt Trip
998                Touch of Evil
8490                     RoboCop
5791               Starting Over
8762    Captain America: Civil War
Name: title, dtype: object
```

We can also look at one specifc movie and check "review similarities" for each user

- Of course, we can also look at users rating for several movies just like in the example, but those data is often not publically available.
- instead we look at users' similarity in terms of their review text, this might help us capture their taste and writing style

First, just a brief review of the w2v content we learned in class:

In [53]:
```
from gensim.models.word2vec import Word2Vec
```

Here we just use a single movie's review data from IMDB

In [54]:
```
df = pd.read_excel('Star Wars The Last Jedi-tt2527336.xlsx')
df.head()
```

Out[54]:

	Index	User_name	Date	Title	Rating	Spoilers	Content	Helpful
0	1	yupman	2 January 2018	The Last Jedi was just magical	1	yes	SPOILER: This movie was just magical.The Force...	1,137/1,332
1	2	shoresk-37122	14 February 2018	I made an account just to say how disappointed...	3	yes	This didn't feel like Star Wars. Now, I know p...	529/620
2	3	gogoschka-1	21 January 2018	My Issues With The Storytelling in The Last Je...	none	yes	I didn't hate TLJ, but even if I completely ig...	444/523
3	4	rick20033	25 December 2017	Horrible	1	yes	It's hard to imagine a studio being THIS stupi...	664/818
4	5	milox33	18 December 2017	25 reasons why The Last Jedi is a FAILURE	2	yes	I had relatively high expectations of the Epis...	603/748

We're mainly interested in the "Content" section

- Spoilers: next class we'll use a deep learning model to predict user ratings

In [55]:
```
# same preprocessing function used in class lab
import re
from sklearn import feature_extraction
stop_words = feature_extraction.text.ENGLISH_STOP_WORDS
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer

def preprocess(text):
  text = text.lower() #lowercase
  text = re.sub(r'[^\w\s]', '', text) #remove punctuations
  text = re.sub(r'\d+', '', text) #remove numbers
  text = " ".join(text.split()) #stripWhitespace
  text = text.split()
  text = [x for x in text if x not in stop_words] #remove stopwords
  text = " ".join(text)
  return(text)
```

In [56]:
```
df['review_processed']=df['Content'].apply(lambda x:preprocess(x))
df['review_processed']=df['review_processed'].apply(lambda x:x.split())
```

In [57]:
```
# try tuning these parameters
model = Word2Vec(sentences=df['review_processed'].tolist(), size=300, min_count=1, window=9,workers=-1,s
```

In [58]:
```
vocab = model.wv.index2word
```

In [59]:
```
model.wv.most_similar('jedi', topn=10)
```

Out[59]:
```
[('greg', 0.22424671053886414),
 ('scriptluke', 0.22144389152526855),
 ('hyperfan', 0.21331095695495605),
 ('fashioning', 0.2130623161792755),
 ('sleeping', 0.20837238430976868),
 ('joker', 0.20772646367549896),
 ('linesa', 0.2052239030599594),
 ('glimmers', 0.20460020005702972),
 ('wisdomin', 0.20280110836029053),
 ('todayi', 0.19892311096191406)]
```

We can use w2v to get some general sense about the sentiment from the reviewers

In [60]:
```
model.wv.most_similar('director', topn=10)
```

Out[60]:
```
[('smithers', 0.22281339764595032),
 ('partically', 0.21553602814674377),
 ('assisted', 0.2116401642560959),
 ('snokess', 0.20481878519058228),
 ('yang', 0.19858521223068237),
 ('onscreen', 0.1981717348098755),
 ('hopesorry', 0.19800515472888947),
 ('campaigning', 0.1979249119758606),
 ('plated', 0.19484788179397583),
 ('objectivelytherein', 0.19368767738342285)]
```

In [61]:
```
model.wv.most_similar('story', topn=10)
```

Out[61]:
```
[('tearyeyed', 0.23702871799468994),
 ('iow', 0.2256821393966748),
 ('unsubtle', 0.21841345727443695),
 ('hindrance', 0.21751773357391357),
 ('survives', 0.2097315788269043),
 ('steeply', 0.20562198758125305),
 ('itthen', 0.20497597754001617),
 ('communal', 0.20290125906467438),
 ('storystar', 0.19620609283447266),
 ('revitalize', 0.19492854177951813)]
```

In [62]:
```
v_hamill = model.wv['hamill']
v_actor = model.wv['actor']
import numpy
numpy.dot(v_actor, v_hamill)/(numpy.linalg.norm(v_actor)* numpy.linalg.norm(v_hamill))
```

Out[62]:
```
-0.04689048
```

Now back to our recommender system

- very similarly, we can calculate the similarities between reviews
- under the assumption that each user only writes on review
    - we can get the similarites between users

These steps are exact the same as we did in the movie item recommender:

In [63]:
```
df.head()
```

Out[63]:

	Index	User_name	Date	Title	Rating	Spoilers	Content	Helpful	review_processed
0	1	yupman	2 January 2018	The Last Jedi was just magical	1	yes	SPOILER: This movie was just magical.The Force...	1,137/1,332	[spoiler, movie, just, magicalthe, force, like...
1	2	shoresk-37122	14 February 2018	I made an account just to say how disappointed...	3	yes	This didn't feel like Star Wars. Now, I know p...	529/620	[didnt, feel, like, star, wars, know, people, ...
2	3	gogoschka-1	21 January 2018	My Issues With The Storytelling in The Last Je...	none	yes	I didn't hate TLJ, but even if I completely ig...	444/523	[didnt, hate, tlj, completely, ignore, fans, c...
3	4	rick20033	25 December 2017	Horrible	1	yes	It's hard to imagine a studio being THIS stupi...	664/818	[hard, imagine, studio, stupid, unwise, disres...
4	5	milox33	18 December 2017	25 reasons why The Last Jedi is a FAILURE	2	yes	I had relatively high expectations of the Epis...	603/748	[relatively, high, expectations, episode, viii...

In [64]:
```
# filling in the null values to ease vectorization process
df['Content'] = df['Content'].fillna('')
```

In [65]:
```
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(df['Content'])
```

In [66]:
```
# here we have 5109 reviews, and 406,783 terms in our lexicon
tfidf_matrix.shape
```

Out[66]: (5109, 406783)

In [67]:
```
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [68]:
```
# just like we did before, now we create a list called users, instead of movie titles
df = df.reset_index()
users = df['User_name']
indices = pd.Series(df.index, index=df['User_name'])
```

In [69]:
```
# for each user, we recommend its most similar users
def get_recommendations(user):
    idx = indices[user]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:31]
    user_indices = [i[0] for i in sim_scores]
    return users.iloc[user_indices]
```

In [70]:
```
df.head(2)
```

Out[70]:

	index	Index	User_name	Date	Title	Rating	Spoilers	Content	Helpful	review_processed
0	0	1	yupman	2 January 2018	The Last Jedi was just magical	1	yes	SPOILER: This movie was just magical.The Force...	1,137/1,332	[spoiler, movie, just, magicalthe, force, like...
1	1	2	shoresk-37122	14 February 2018	I made an account just to say how disappointed...	3	yes	This didn't feel like Star Wars. Now, I know p...	529/620	[didnt, feel, like, star, wars, know, people, ...

In [71]:
```
# most similar users to 'yupman', based on their review content
get_recommendations('yupman').head(10)
```

Out[71]:
```
1394          jasontjordan
1041           Tikonderoga
2000           garth-11341
1621     deviloutofnowhere
183         Marco Bonelli
5           kingjon-08903
1569          shayol-48294
11              mrbenflood
715        elisdanielflores
2425             omer_a_uk
Name: User_name, dtype: object
```

Obviously the user recommendation is only based on a single movie, it could get much better if you have dozens of movies to compare.

- reference:

- https://towardsdatascience.com/using-cosine-similarity-to-build-a-movie-recommendation-system-ae7f20842599 (https://towardsdatascience.com/using-cosine-similarity-to-build-a-movie-recommendation-system-ae7f20842599)
- https://www.kaggle.com/rounakbanik/movie-recommender-systems/notebook (https://www.kaggle.com/rounakbanik/movie-recommender-systems/notebook)
- https://github.com/bdferris642/airbnb_insight/blob/master/Topic%20Analysis.ipynb (https://github.com/bdferris642/airbnb_insight/blob/master/Topic%20Analysis.ipynb)
- https://nlp.stanford.edu/projects/glove/ (https://nlp.stanford.edu/projects/glove/)
- https://github.com/TharinduDR/Simple-Sentence-Similarity/blob/master/Sentence%20Similarity%20-%20Word%20Vectors.ipynb (https://github.com/TharinduDR/Simple-Sentence-Similarity/blob/master/Sentence%20Similarity%20-%20Word%20Vectors.ipynb)

In [ ]: