UE20CS312 - Data Analytics

Worksheet 4a - Collaborative & Content based filtering

PES University

SUNDEEP A , Dept of CSE - PES1UG20CS445

Collaborative & Content based filtering

The **Collaborative filtering method** for recommender systems is a method that is solely based on the past interactions that have been recorded between users and items, in order to produce new recommendations.

The **Content-based** approach uses additional information about users and/or items. The Content-based approach requires a good amount of information about items' features, rather than using the user's interactions and feedback.

Prerequisites

- Revise the following concepts
  - TF-IDF
  - Content-based filtering
  - Cosine Similarity
- Install the following software
  - pandas
  - numpy
  - sklearn

Task

After the disastrous pitfall of Game of Thrones season 8), George R. R. Martin set out to fix mindless mistakes caused by the producers David and Daniel.

A few years down the line, we now are witnessing George R. R. Martin's latest work: House of the Dragon. This series is a story of the Targaryen civil war that took place about 200 years before events portrayed in Game of Thrones.

In this notebook you will be exploring and analying tweets related to The House of Dragon TV series. First we shall tokenize the textual data using TF-IDF. Then we will proceed to find the top-k most similar tweets using cosine similarity between the transformed vectors.

The dataset has been extracted using the Twitter API by utilizing a specific search query. The data has been extensively preprocessed and a small subset has been stored within the `twitter_HOTD_DA_WORKSHEET4A.csv`

Note: This notebook may contain spoilers to the show.

Data Dictionary

author_id: A unique identifier assigned to the twitter user.

tweet_id: A unique identifier assigned to the tweet.

text: The text associated with the tweet.

retweet_count: The number of retweets for this particular tweet.

reply_count: The number of replies for this particular tweet.

like_count: The number of likes for this particular tweet.

quote_count: The number of quotes for this particular tweet.

tokens: List of word tokens extracted from `text`.

hashtags: List of hashtags extracted from `text`.

Points

The problems in this worksheet are for a total of 10 points with each problem having a different weightage.

- Problem 1: 4 points
- Problem 2: 4 points
- Problem 3: 2 points

Loading the dataset

In [1]:
```
# Import pandas
import pandas as pd
# Use pandas read_csv function to load csv as DataFrame
df = pd.read_csv('./twitter_HOTD_DA_WORKSHEET4A.csv')
df.head(5)
```

Out[1]:

	author_id	tweet_id	text	retweet_count	reply_count	like_count	quote_cou
0	1576486223442583554	1577813753902555136	rt i would perform my duty if mother had only ...	327	0	0	
1	732912167846973441	1577813747846254592	rt viserys look at me!! aemond #houseofthedragon	2164	0	0	
2	891426095928672257	1577813743794257920	rt house of the dragon is a show about a king ...	905	0	0	
3	352012951	1577813724366249986	man just thinking of when viserys finally croa...	0	0	0	
4	1090278774925611008	1577813708818059264	rt jajaja. #houseofthedragon	29	0	0	

Problem 1 (4 points)

Tokenize the string representations provided in the **tokens** column of the DataFrame using TF-IDF from sklearn. Then print out the TF-IDF of the first row of the DataFrame.

Solution Steps:

1. Initialize the `TfidfVectorizer()`
2. Use the `.fit_transform()` method on the entire text
3. `.transform()` the Text
4. Print number of samples and features using `.shape`
5. Print the TF-IDF of the first row

For futher reference: https://www.analyticsvidhya.com/blog/2021/09/creating-a-movie-reviews-classifier-using-tf-idf-in-python/

In [2]:
```
# Imports
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

In [3]:
```
# Convert string representation of a list into a list of strings
import ast
text = []
for r in df['tokens']:
    res = ast.literal_eval(r)
    if(' '.join(res).lower() == ''):
        print(r)
    text.append(' '.join(res).lower())
# Print the end result
#text
```

In [4]:
```
#part 1:
tf_idf_vect = TfidfVectorizer()
#part 2:
fit_text = tf_idf_vect.fit_transform(text)
fit_text
#part 3:
fit_text = tf_idf_vect.transform(text)
fit_text
#part 4:
print("Number of Samples: %d \n  Number of features: %d" % fit_text.shape)
```

```
Number of Samples: 8061
  Number of features: 10950
```

In [5]:
```
#Part 5:
print("TF-IDF of First row are : \n",fit_text[0])
```

```
TF-IDF of First row are :
   (0, 7080)    0.5632689245894086
  (0, 6277)     0.3774828944719565
  (0, 3021)     0.47218947633853087
  (0, 1054)     0.5632689245894086
```

Problem 2 (4 points)

Find the top-5 most similar tweets to the tweet with index `7558` using cosine similarity between the TF-IDF vectors.

Solution Steps:

1. Import `cosine_similarity` from sklearn.metrics.pairwise
2. Compute `cosine_similarity` using text_tf with index `7558` and all other rows
3. Use `argsort` to sort the cosine_similarity results
4. Print indices of top-5 most similar results from sorted array (hint: argsort sorts in ascending order)
5. Display text of top-5 most similar results using `df.iloc[index]`

In [6]:
```
# Print out the tokens from index `7654`
print(text[7558])
# Print out the text from index `7654`
print(df.iloc[7558][2])
```

```
viserys wanna build lego set mind business let man live peace
rt viserys just wanna build his lego set and mind his business . let that man live in peac
e #houseofthedragon
```

In [7]:
```
#part 1:
from sklearn.metrics.pairwise import cosine_similarity
```

In [8]:
```
#part 2:
similarity_measure_cosine = cosine_similarity(fit_text[7558], fit_text)
print(similarity_measure_cosine)
```

```
[[0.         0.10725007 0.         ... 0.         0.         0.        ]]
```

In [9]:
```
#part 3:
similarity_measure_cosine_sorted = similarity_measure_cosine.argsort()
print(similarity_measure_cosine_sorted)
```

```
[[   0 5294 5293 ... 3656 7558 7595]]
```

In [10]:
```
#part 4:
top_5_similar = similarity_measure_cosine_sorted[0][-6:]
top_5_similar = top_5_similar[::-1]
#Since we want to find the top 5 similar to index 7558 , we exclude 7558
top_5_similar = [i for i in top_5_similar if i!=7558]
top_5_similar
```

Out[10]:
```
[7595, 3656, 6548, 7705, 3534]
```

In [11]:
```
#Part 5
df.iloc[top_5_similar[:]].text
```

Out[11]:
```
7595    viserys just wanna build his lego set and mind...
3656    rt my man's literally just trying to build a l...
6548    rt daemon and rhaenyra are never letting this ...
7705    mom said it's my turn on the valyrian lego set...
3534    rt don't play with them. they're here for busi...
Name: text, dtype: object
```

Problem 3 (2 point)

A great disadvantage in using TF-IDF is that it can not capture semantics. If you had classify tweets into positive/negative, what technique would you use to map words to vectors? In short words, provide the sequence of solution steps to solve this task. Note: Assume sentiment labels have been provided.

(Hint: take a look at how I've provided solution steps in previous problems)

Word2Vec can be used to map words to vectors.
Solution Steps :
Step 1 : Import Word2Vec from gemsim.models
Step 2 : Tokenize the sentence into words
Step 3 : Creating the word embeddings[Building/Training the Word2Vec model]
Step 4 : Evaluate the model [This can be done by Cosine Similarity of the two words you enter.]