PES University, Bangalore

Established under Karnataka Act No. 16 of 2013

UE20CS312 - Data Analytics - Worksheet 4a

Course instructor: Gowri Srinivasa, Professor Dept. of CSE, PES University

Designed by Harshith Mohan Kumar, Dept. of CSE - harshithmohankumar@pesu.pes.edu

Ishita Bharadwaj, PES1UG20CS648
Collaberated with Hita, PES1UG20CS645 and Jeffrey, PES1UG20CS651

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

```
# 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
```

	author_id	tweet_id	text	retweet_count	rep
0	1576486223442583554	1577813753902555136	rt i would perform my duty if mother had only ...	327	
1	732912167846973441	1577813747846254592	rt viserys look at me!! aemond #houseofthedragon	2164	
2	891426095928672257	1577813743794257920	rt house of the dragon is a show about a king ...	905	
3	352012951	1577813724366249986	man just thinking of when viserys finally croa...	0	
4	1090278774925611008	1577813708818059264	rt jajaja. #houseofthedragon	29	
...	...	...	...	...	
8056	2916627870	1570663468055040000	rt may i go on record that if matt	166	

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/

```
# Imports
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

```
# 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[:5]
```

```
['perform duty mother betrothed',
 'viserys look aemond',
 'house dragon king bunch questionable decisions based idea dead',
 'man thinking viserys finally croaks fucking races gon na fucking hate damn
cliffhanger making wait year war start lol',
 'jajaja']
```

```
len(text)
```

```
8061
```

```
#using the count vectorizer
count = CountVectorizer()
word_count=count.fit_transform(text)
# print(word_count)
word_count.shape
```

```
(8061, 10950)
```

8061 sentences (list elements in `text`) and 10950 number of unique words in all sentences.

```
print(word_count.toarray())
```

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

```
tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(word_count)
df_idf = pd.DataFrame(tfidf_transformer.idf_, index=count.get_feature_names_out(),columns=
```

```
#inverse document frequency
df_idf.sort_values(by=['idf_weights'])
```

	idf_weights
dragon	2.158942
house	2.161317
episode	3.180374
rhaenyra	3.340764
daemon	3.511810
...	...
hoodie	9.301770
hood	9.301770
honma	9.301770
hoteeel	9.301770
lastnights	9.301770

TF-IDF of the first row of the DataFrame-

```
#tfidf
tf_idf_vector=tfidf_transformer.transform(word_count)
feature_names = count.get_feature_names_out()
first_document_vector=tf_idf_vector[0]
df_tfifd= pd.DataFrame(first_document_vector.T.todense(), index=feature_names, columns=["t
df_tfifd.sort_values(by=["tfidf"],ascending=False)
```

	tfidf
perform	0.563269
betrothed	0.563269
duty	0.472189
mother	0.377483
populiariame	0.000000
...	...
feeding	0.000000
feel	0.000000
feeling	0.000000
feelings	0.000000
cθlaTeral	0.000000

10950 rows × 1 columns

```
tf_idf_vector[0].shape
```

```
(1, 10950)
```

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]`

```
# 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 ir
```

```
# Print out the tokens from index `7595`
print(text[7595])
# Print out the text from index `7595`
print(df.iloc[7595][2])
```

```
viserys wanna build lego set mind business let man live peace
viserys just wanna build his lego set and mind his business . let that man live in pe
```

```
# Print out the tokens from index `7595`
print(text[3656])
# Print out the text from index `7595`
print(df.iloc[3656][2])
```

```
man literally trying build lego set valyria deal shit
rt my man's literally just trying to build a lego set of valyria but has to deal with
```

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
cs= {}
```

```
for i in range(len(text)):
  cs[i]=float(cosine_similarity(tf_idf_vector[i],tf_idf_vector[7558]))
```

```
d=dict(sorted(cs.items(), key=lambda item: item[1],reverse=True))
```

```
import itertools
out = dict(itertools.islice(d.items(), 6))
```

```
out
```

```
{7558: 1.0,
 7595: 1.0,
 3656: 0.42562320434123724,
 6548: 0.3798704794558604,
 7705: 0.2825189055056654,
 3534: 0.2694649041162415}
```

```
for k,v in out.items():
  print(k,": ",df.iloc[k][2])
```

```
7558 :  rt viserys just wanna build his lego set and mind his business . let that mar
7595 :  viserys just wanna build his lego set and mind his business . let that man li
3656 :  rt my man's literally just trying to build a lego set of valyria but has to c
6548 :  rt daemon and rhaenyra are never letting this man live in peace 😭 #houseoft
7705 :  mom said it's my turn on the valyrian lego set ( #houseofthedragon
3534 :  rt don't play with them. they're here for business only. #houseofthedragon
```

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)

We can use Bag Of Words (BoW) or TF-IDF or Word2Vec to extract features of the tweets numerically before we train a model with them. TF-IDF and BoW cannot capture high-level semantic meanings behind text data. However Word2Vec preserves the relationship between words in a sentence, keeping it's semantics. Word2Vec maps all the words into vector of a given dimension - called word embeddings. These are used as features to train a model(a postive-negative classifier in our case). Words in a similar context have similar embeddings.

1. Read file
2. Cleaning - Remove numbers, stopwords, and special characters.(like punctuation marks)

3. Tokenise the corpus. (`token` field in df/dataset)

```
tokenized_tweet = df.tidy_tweet.apply(lambda x: x.split())
```

4. Preprocessing - Apply stemming, lemmatisation on the stitched-back-to-tweet tokens.

```
for i in range(len(tokenized_tweet)):
  tokenized_tweet[i] = ' '.join(tokenized_tweet[i])
  df['tidy_tweet'] = tokenized_tweet
```

5. Apply `Word2Vec` on the preprocessed tweets.

```
  model_w2v = gensim.models.Word2Vec(
  tokenized_tweet,
  size=200, # desired no. of features/independent variables
  window=5, # context window size
  min_count=2, # Ignores all words with total frequency lower than 2.
  sg = 1, # 1 for skip-gram model
  hs = 0,
  negative = 10, # for negative sampling
  workers= 32, # no.of cores
  seed = 34
  )

  model_w2v.train(tokenized_tweet, total_examples= len(combi['tidy_tweet']),
```

Colab paid products - Cancel contracts here