databricks Search Engine for Movie Plot Summaries (A1Q2)

(https://databricks.com)

References

TF-IDF wiki: https://en.wikipedia.org/wiki/Tf%E2%80%93idf (https://en.wikipedia.org/wiki/Tf%E2%80%93idf)

NLTK: https://www.nltk.org/ (https://www.nltk.org/)

Cosine Similarity with TF-IDF: https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/ (https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/)

Imports

```
!pip install nltk
```

```
Collecting nltk
  Downloading nltk-3.8.1-py3-none-any.whl (1.5 MB)
     |████████████████████████████████| 1.5 MB 4.3 MB/s eta 0:00:01
Collecting tqdm
  Downloading tqdm-4.66.1-py3-none-any.whl (78 kB)
     |████████████████████████████████| 78 kB 7.9 MB/s  eta 0:00:01
Requirement already satisfied: click in /databricks/python3/lib/python3.9/site-packages (from nltk) (8.0.4)
Collecting regex>=2021.8.3
  Downloading regex-2023.8.8-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (771 kB)
     |████████████████████████████████| 771 kB 22.4 MB/s eta 0:00:01
Requirement already satisfied: joblib in /databricks/python3/lib/python3.9/site-packages (from nltk) (1.1.1)
Installing collected packages: tqdm, regex, nltk
Successfully installed nltk-3.8.1 regex-2023.8.8 tqdm-4.66.1
WARNING: You are using pip version 21.2.4; however, version 23.2.1 is available.
You should consider upgrading via the '/local_disk0/.ephemeral_nfs/envs/pythonEnv-e7405311-592a-48a9-98c9-7f13e083a407/bin/python -m
pip install --upgrade pip' command.
```

```
import nltk
from nltk.corpus import stopwords
from math import log, sqrt

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
plot_summary_location = '/FileStore/tables/plot_summaries.txt'
search_terms_location = '/FileStore/tables/search_terms-2.txt'
movies_metadata_location = '/FileStore/tables/movie_metadata.tsv'
```

```
# ps = plot summary and total_documents = total number of movies
ps = sc.textFile(plot_summary_location)
total_documents = ps.count()
```

```
total_documents
```

```
Out[5]: 42306
```

```
# spliting each summary to list of strings
ps = ps.map(lambda x: x.split())

# converting all the strings to lower case
ps = ps.map(lambda x: [word.lower() for word in x])
```

```
# removing punctuations
def lower_clean_str(x):
  punc='!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
  lowercased_str = x.lower()
  for ch in punc:
    lowercased_str = lowercased_str.replace(ch, '')
  return lowercased_str

def lower_clean_str_list(l):
    return [lower_clean_str(i) for i in l]

ps = ps.map(lower_clean_str_list)
```

```
ps.take(10)
```

```
Out[8]: [['23890098',
  'shlykov',
  'a',
  'hardworking',
  'taxi',
  'driver',
  'and',
  'lyosha',
  'a',
  'saxophonist',
  'develop',
  'a',
  'bizarre',
  'lovehate',
  'relationship',
  'and',
  'despite',
  'their',
  'prejudices',
  'realize',
  'they',
```

```
# remove strings with length <= 2

def remove_small_strings(x):
    return [word for word in x if len(word)>2]

ps = ps.map(remove_small_strings)
```

```
ps.take(10)
```

```
Out[10]: [['23890098',
  'shlykov',
  'hardworking',
  'taxi',
  'driver',
  'and',
  'lyosha',
  'saxophonist',
  'develop',
  'bizarre',
  'lovehate',
  'relationship',
  'and',
  'despite',
  'their',
  'prejudices',
  'realize',
  'they',
  'arent',
  'different',
```

```
# Once we convert to lower case and remove punctuations, we remove the stop words
ps = ps.map(lambda x: [word for word in x if word not in stop_words])
```

```
# Mapping to get the word frequency of each word in a document

def per_doc_term_count_init(l):
    doc_id = l[0]
    return [((doc_id, word), 1) for word in l[1:]]

ps = ps.flatMap(per_doc_term_count_init)
```

```
ps.take(10)
```

```
Out[13]: [(('23890098', 'shlykov'), 1),
 (('23890098', 'hardworking'), 1),
 (('23890098', 'taxi'), 1),
 (('23890098', 'driver'), 1),
 (('23890098', 'lyosha'), 1),
 (('23890098', 'saxophonist'), 1),
 (('23890098', 'develop'), 1),
 (('23890098', 'bizarre'), 1),
 (('23890098', 'lovehate'), 1),
 (('23890098', 'relationship'), 1)]
```

```
per_doc_term_count = ps.reduceByKey(lambda x, y: x+y)
```

```
per_doc_term_count.take(10)
```

```
Out[15]: [(('23890098', 'hardworking'), 1),
 (('23890098', 'lyosha'), 1),
 (('23890098', 'saxophonist'), 1),
 (('23890098', 'develop'), 1),
 (('23890098', 'lovehate'), 1),
 (('23890098', 'relationship'), 1),
 (('23890098', 'realize'), 1),
 (('23890098', 'arent'), 1),
 (('31186339', 'nation'), 1),
 (('31186339', 'panem'), 1)]
```

```
# get total count of all the terms in a document

def per_doc_all_terms_init(x):
    (doc_id, term), cnt = x
    return (doc_id, cnt)

per_doc_all_terms_count = ps.map(per_doc_all_terms_init).reduceByKey(lambda x, y: x+y)
per_doc_all_terms_count.take(10)
```

```
Out[16]: [('31186339', 432),
 ('595909', 210),
 ('1952976', 406),
 ('24225279', 321),
 ('20532852', 60),
 ('15401493', 184),
 ('18188932', 55),
 ('2940516', 35),
 ('24448645', 43),
 ('15072401', 60)]
```

```
def change_kv_structure(x):
    (doc_id, term), count = x
    return doc_id, (term, count)

per_doc_term_count_doc_key = per_doc_term_count.map(change_kv_structure)
per_doc_term_count_doc_key.take(10)
```

```
Out[17]: [('23890098', ('hardworking', 1)),
 ('23890098', ('lyosha', 1)),
 ('23890098', ('saxophonist', 1)),
 ('23890098', ('develop', 1)),
 ('23890098', ('lovehate', 1)),
 ('23890098', ('relationship', 1)),
 ('23890098', ('realize', 1)),
 ('23890098', ('arent', 1)),
 ('31186339', ('nation', 1)),
 ('31186339', ('panem', 1))]
```

Calculate the term frequency

```
tf = per_doc_term_count_doc_key.join(per_doc_all_terms_count)

tf.take(5)
```

```
Out[18]: [('5720418', (('group', 1), 188)),
 ('5720418', (('led', 2), 188)),
 ('5720418', (('ward', 2), 188)),
 ('5720418', (('frank', 1), 188)),
 ('5720418', (('hatch', 1), 188))]
```

Get the names of the movies

```
# mmd = movie meta data
mmd = sc.textFile(movies_metadata_location).map(lambda x: x.split('\t'))

# Get the movie/document id and the movie name which are the indices below
mmd = mmd.map(lambda x: (x[0], x[2]))

mmd.take(10)
```

```
Out[19]: [('975900', 'Ghosts of Mars'),
 ('3196793', 'Getting Away with Murder: The JonBenét Ramsey Mystery'),
 ('28463795', 'Brun bitter'),
 ('9363483', 'White Of The Eye'),
 ('261236', 'A Woman in Flames'),
 ('13696889', 'The Gangsters'),
 ('18998739', "The Sorcerer's Apprentice"),
 ('10408933', "Alexander's Ragtime Band"),
 ('9997961', 'Contigo y aquí'),
 ('2345652', 'City of the Dead')]
```

```
# Combine mmd and ps to replace movie/document id with the movie name.

mmd_ps_join = mmd.join(tf)
mmd_ps_join.take(10)
```

```
Out[20]: [('261236', ('A Woman in Flames', (('class', 1), 206))),
 ('261236', ('A Woman in Flames', (('housewife', 1), 206))),
 ('261236', ('A Woman in Flames', (('leaves', 1), 206))),
 ('261236', ('A Woman in Flames', (('arrogant', 1), 206))),
 ('261236', ('A Woman in Flames', (('husband', 1), 206))),
 ('261236', ('A Woman in Flames', (('idea', 1), 206))),
 ('261236', ('A Woman in Flames', (('becoming', 1), 206))),
 ('261236', ('A Woman in Flames', (('call', 1), 206))),
 ('261236', ('A Woman in Flames', (('named', 1), 206))),
 ('261236', ('A Woman in Flames', (('yvonne', 1), 206)))]
```

```
def transform_and_get_term_frequency(x):
    doc_id, (doc_name, ((term, term_count_per_doc), doc_all_terms_count)) = x
    return term, (doc_id, doc_name, term_count_per_doc/doc_all_terms_count)

tf_joined = mmd_ps_join.map(transform_and_get_term_frequency)
```

```
tf_joined.take(10)
```

```
Out[22]: [('class', ('261236', 'A Woman in Flames', 0.0048543689320388345)),
 ('housewife', ('261236', 'A Woman in Flames', 0.0048543689320388345)),
 ('leaves', ('261236', 'A Woman in Flames', 0.0048543689320388345)),
 ('arrogant', ('261236', 'A Woman in Flames', 0.0048543689320388345)),
 ('husband', ('261236', 'A Woman in Flames', 0.0048543689320388345)),
 ('idea', ('261236', 'A Woman in Flames', 0.0048543689320388345)),
 ('becoming', ('261236', 'A Woman in Flames', 0.0048543689320388345)),
 ('call', ('261236', 'A Woman in Flames', 0.0048543689320388345)),
 ('named', ('261236', 'A Woman in Flames', 0.0048543689320388345)),
 ('yvonne', ('261236', 'A Woman in Flames', 0.0048543689320388345))]
```

Calculate the IDF of each term

```
# Calculate the idf of each term
# per_doc_term_count = (doc, term), term_count_in_doc

def per_doc_term_bool_init(x):
    (doc_id, term), count = x
    return (doc_id, term), 1

per_doc_term_bool = per_doc_term_count.map(per_doc_term_bool_init)
```

```
per_doc_term_bool.take(10)
```

```
Out[24]: [(('23890098', 'hardworking'), 1),
 (('23890098', 'lyosha'), 1),
 (('23890098', 'saxophonist'), 1),
 (('23890098', 'develop'), 1),
 (('23890098', 'lovehate'), 1),
 (('23890098', 'relationship'), 1),
 (('23890098', 'realize'), 1),
 (('23890098', 'arent'), 1),
 (('31186339', 'nation'), 1),
 (('31186339', 'panem'), 1)]
```

```
def transform_for_idf(x):
    (doc_id, term), bool_cnt = x
    return term, bool_cnt

def calculate_idf(x):
    term, term_docs_cnt = x
    return term,  log((1+total_documents)/(1+term_docs_cnt))

idf = per_doc_term_bool.map(transform_for_idf).reduceByKey(lambda x, y: x+y).map(calculate_idf)
```

```
idf.take(10)
```

```
Out[26]: [('nation', 5.223362207022427),
 ('panem', 9.554095547308759),
 ('twelve', 5.334587842132651),
 ('poorer', 7.285412005990393),
 ('past', 2.609686950678585),
 ('rebellion', 5.437772078367882),
 ('must', 2.294744870131187),
 ('provide', 4.100199948942278),
 ('boy', 2.4921895884993623),
 ('selected', 5.15144962543214)]
```

```
idf_dict = {k: v for k, v in idf.collect()}
```

```
# Joining idf and tf_joined to get the final tf-idf value

def get_tf_idf_value(x):
    term, ((doc_id, doc_name, tf), idf) = x
    return term, (doc_id, doc_name, tf*idf)

tf_idf = tf_joined.join(idf)

tf_idf = tf_idf.map(get_tf_idf_value).cache()
```

```
tf_idf.take(10)
```

```
Out[29]: [('housewife', ('261236', 'A Woman in Flames', 0.02591991192686932)),
 ('housewife', ('14856464', 'A Question of Silence', 0.11607612732467566)),
 ('housewife', ('8499528', 'She-Devil', 0.011458158491276996)),
 ('housewife', ('9009812', 'Montenegro', 0.04045077164344758)),
 ('housewife',
  ('23057797',
   "Adultery Diary: One More Time While I'm Still Wet",
   0.4449584880779234)),
 ('housewife', ('23266073', 'Secretariat', 0.012534041917687982)),
 ('housewife', ('24389074', 'The Beloved', 0.266975092846754)),
 ('housewife', ('217824', 'Diary of a Mad Housewife', 0.08090154328689517)),
 ('housewife', ('9590609', 'While She Was Out', 0.024605999340714658)),
 ('housewife', ('26099108', 'The Kids Are All Right', 0.01927617998893531))]
```

```
# Return the movies with highest tf-idf score w.r.t the search terms

st_rdd = sc.textFile(search_terms_location).cache()
st_rdd.collect()
```

```
Out[30]: ['stealing', 'dragon', 'crime', 'movie with plot twists', 'happy ending movie']
```

```
# Helper functions

def calculate_search_term_tf_idf(x):
    # Multiply the term frequency w.r.t the search term string and
    # use the idf value of the term calculated from the plot summaries.
    term, tf_val = x
    return term, tf_val * idf_dict.get(term, 0)

def doc_mag_wrt_term_init(x):
    term, (doc_id, movie_name, tf_idf_val) = x
    return movie_name, tf_idf_val ** 2

def st_mag_init(x):
    term, tf_idf_val = x
    return term, tf_idf_val ** 2

def get_dot_product_init(x):
    # ('movies', (('1850269', 'Kekexili: Mountain Patrol', 0.021384103580413562), 1.0852432567059882))
    term, ((doc_id, movie_name, tf_idf_doc), tf_idf_term) = x
    return movie_name, tf_idf_doc * tf_idf_term

def get_cosine_sim_init(x):
    movie_name, (dot_prod_val, doc_mag_term_val) = x
    return movie_name, dot_prod_val/doc_mag_term_val
```

Calculate the top recommendations based on the search terms

```
Top movie recommendations for "stealing" are:
Union City
Fallen Arches
Young Guns of Texas
The Ace of Scotland Yard
The Brave Ones
Betrayal
The Crouching Beast
The Marshal of Windy Hollow
The Curse of the Doll People
Graveyard Disturbance
---------------------------------------------------------
Top movie recommendations for "dragon" are:
Lung Fung Restaurant
The Young Master
One Droopy Knight
Dragonquest
Eragon
Dragon Tiger Gate
Cleopatra Jones and the Casino of Gold
Project A
```