In [1]:
```
import sys
import pandas as pd
import requests
import csv
import os
import random
import time
import urllib.request
import re
import json
```

In [4]:
```
from bs4 import BeautifulSoup as bs
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from datetime import datetime
from rake_nltk import Rake
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

In [5]:
```
with open('arxivData.json', 'r') as f:
    data = json.load(f)
```

In [6]:
```
con_month = { 1:'Jan',2:'Feb', 3:'Mar', 4:'Apr',5:'May', 6:"Jun",7:'Jul',8:'Aug',9:'Sep',10:'Oct',11:'Nov',12:'Dec'}
con_day = { 1:'01',2:'02',3:'03',4:'04',5:'05',6:'06',7:'07',8:'08',9:'09'}
```

In [7]:
```
for i in data:
    if i['month'] in con_month:
        i['month'] = con_month[i['month']]
    if i['day'] in con_day:
        i['day'] = con_day[i['day']]
    else:
        i['day'] = str(i['day'])
    i['year'] = str(i['year'])

for i in data:
    date = i['day']+'-'+i['month']+'-'+i['year']
    #print("DATE=",date)
    k = datetime.strptime(date,'%d-%b-%Y')
    i['date'] = datetime.timestamp(k)
    del i['day'], i['month'],i['year']

for i in data:
    i['summary'] = i['summary'].replace('\n',' ')
    i['title'] = i['title'].replace('\n',' ')
    del i['id'],i['link'],i['tag'] #irrelevant for us
```

In [8]:
```
term = input()
```

```
schema
```

In [9]:
```
papers = []
for i in data:
    if term in i['title'] or term in i['summary']:
        papers.append(i)

#sort according to date
papers = sorted(papers, key = lambda i:i['date'])
len(papers)
```

Out[9]:
```
135
```

In [10]:
```
#take top 5 and bottom 5
final_summary = []
if len(papers)>10:
    final_list = papers[:5] + papers[-5:]
else:
    final_list = papers
for i in final_list:
    final_summary.append(i['summary'])
```

In [11]:
```
# Removes punctuation
#re.sub is used for replacing strings
def pre_process(text):

    text = text.lower()
    text = re.sub("&lt;/?.*?&gt;","&lt;&gt",text)
    text = re.sub("(\\d|\\W)+"," ",text)
    return text
```

In [12]:
```
# Create vocabulary and word counts

def get_stop(fpath):

    with open(fpath,"r",encoding = "utf-8") as f:
        stopw = f.readlines()
        stop_set = set(m.strip() for m in stopw)
        return frozenset(stop_set)
```

In [13]:
```
def sort_coo(coo_matrix):
    tuples = zip(coo_matrix.col, coo_matrix.data)
    return sorted(tuples,key=lambda x:(x[1],x[0]),reverse=True)
```

In [14]:
```
def extraction(fnames,sitems,topn=0):
    sitems = sitems[:topn]
    scoreval = []
    featureval = []

    for idx, score in sitems:
        scoreval.append(round(score,3))
        featureval.append(fnames[idx])

    results = {}

    for idx in range(len(featureval)):
        results[featureval[idx]] = scoreval[idx]

    return results
```

In [15]:
```
for i in final_list:
    i['summary'] = pre_process(i['summary'])
```

In [18]:
```
stopwords = get_stop('stopwords.txt')
```

In [19]:
```
docs = final_summary
```

In [20]:
```
try:
    cv = CountVectorizer(max_df = 0.85,stop_words = stopwords,max_features=100
00)
    word_cv = cv.fit_transform(docs)
    #print(word_cv)
except:
    pass
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\feature_extraction\text.p
y:300: UserWarning: Your stop_words may be inconsistent with your preprocessi
ng. Tokenizing the stop words generated tokens ['herse', 'himse', 'itse', 'my
se'] not in stop_words.
  'stop_words.' % sorted(inconsistent))
```

In [21]:
```
tt = TfidfTransformer(smooth_idf=True,use_idf=True)
tt.fit(word_cv)
```

Out[21]:
```
TfidfTransformer(norm='l2', smooth_idf=True, sublinear_tf=False, use_idf=Tru
e)
```

In [22]:
```
feature_names = cv.get_feature_names()
#feature_names
```

In [23]:
```
final_text_key = []
for doc in docs:
    tvector = tt.transform(cv.transform([doc]))
    sitems = sort_coo(tvector.tocoo())
    #print(sitems)
    keywords = extraction(feature_names,sitems,10)
    #print(keywords)
    text_key = {
        'summary':pre_process(doc),
        'keywords':list(keywords.keys())
    }
    final_text_key.append(text_key)
```

In [24]:
```
#final_text_key[:2]
```

In [25]:
```
for i in final_list:
    for j in final_text_key:
        if j['summary']==i['summary']:
            i['keywords']=j['keywords']
            del i['summary']
            break
```

In [26]:
```
final_list[-1]
```

Out[26]:
```
{'author': "[{'name': 'Neil Dhir'}, {'name': 'Houman Dallali'}, {'name': 'Mo
Rastgaar'}]",
 'title': 'Coregionalised Locomotion Envelopes - A Qualitative Approach',
 'date': 1520879400.0,
 'keywords': ['new',
  'signals',
  'sensors',
  'locomotion',
  'control',
  'exploit',
  'method',
  'learning',
  'walking',
  'variates']}
```

In [32]:
```
f = term+'.csv'
```

In [33]:
```
def get_cit(title):
        dat = requests.get('https://scholar.google.com/scholar?q='+title, head
ers = {'User-agent': 'your bot 0.1'})
        #requests.get(link, headers = {'User-agent': 'your bot 0.1'})
        dat.raise_for_status()
        soup = bs(dat.text,"html.parser")
        res_link = soup.select(".gs_rt a")
        for i in res_link:
            print(i.text)
            print(i['href'])
            break
        cit_link = soup.select(".gs_fl a")
        for i in cit_link:
                if "Cited by" in i.text:
                    print(int(i.text[9:]))
                    return int(i.text[9:])
                    break
```

In [34]:
```
with open(f,"w") as file:
    fwriter = csv.writer(file)
    fwriter.writerow(["author","title","date","keywords","citation"])
    for i in final_list:
        fwriter.writerow([i['author'],i['title'],i['date'],i['keywords'],get_c
it(i['title'])])
        time.sleep(3)
```

```
Syntactic-head-driven generation
https://dl.acm.org/citation.cfm?id=991969
11
An implemented model of punning riddles
https://www.aaai.org/Papers/AAAI/1994/AAAI94-096.pdf
107
Operations for learning with graphical models
http://www.jair.org/papers/paper62.html
769
Genetic algorithms in time-dependent environments
https://link.springer.com/chapter/10.1007/978-3-662-04448-3_13
51
Reasoning with individuals for the description logic
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.7645&rep=rep1&typ
e=pdf
3
MRI tumor segmentation with densely connected 3D CNN
https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10574/10574
1F/MRI-tumor-segmentation-with-densely-connected-3D-CNN/10.1117/12.2293394.sh
ort
26
```

In [35]:
```
f="schema.csv"
```

In [37]:
```
df=pd.read_csv(f,encoding='latin-1')
df.head()
```

Out[37]:

	author	title	date	keywords	citation
0	[{'name': 'Esther Koenig'}]	Syntactic-Head-Driven Generation	767903400.0	['head', 'semantic', 'syntactic', 'driven', 'g...	11.0
1	[{'name': 'Kim Binsted'}, {'name': 'Graeme Rit...	An implemented model of punning riddles	771445800.0	['jokes', 'joke', 'jape', 'model', 'words', 'u...	107.0
2	[{'name': 'W. L. Buntine'}]	Operations for Learning with Graphical Models	786220200.0	['graphical', 'learning', 'models', 'framework...	769.0
3	[{'name': 'Christopher Ronnewinkel'}, {'name':...	Genetic Algorithms in Time-Dependent Environments	941653800.0	['rate', 'population', 'phase', 'mutation', 'g...	51.0
4	[{'name': 'Ian Horrock'}, {'name': 'Ulrike Sat...	Reasoning with Individuals for the Description...	957983400.0	['tbox', 'reasoning', 'implementation', 'algor...	3.0

In [38]:
```
df=df.dropna()
```

In [39]:
```
df=df.sort_values(['citation','date'],ascending=[0,1])
```

In [40]:
```
highest = df.iloc[0]['title']
highest
```

Out[40]:
```
'Operations for Learning with Graphical Models'
```

In [41]:
```
paper_mat = df.pivot_table(columns='title',values='citation')
paper_mat
```

Out[41]:

title	An implemented model of punning riddles	Genetic Algorithms in Time-Dependent Environments	MRI Tumor Segmentation with Densely Connected 3D CNN	Operations for Learning with Graphical Models	Reasoning with Individuals for the Description Logic SHIQ	Syntactic-Head-Driven Generation
citation	107.0	51.0	26.0	769.0	3.0	11.0

In [42]:
```
high_cit=paper_mat[highest]
high_cit
```

Out[42]:
```
citation    769.0
Name: Operations for Learning with Graphical Models, dtype: float64
```

In [43]:
```
sim2high=paper_mat.corrwith(high_cit,axis=0)
sim2high
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\function_base.py:2522: R
untimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar)
C:\ProgramData\Anaconda3\lib\site-packages\numpy\lib\function_base.py:2451: R
untimeWarning: divide by zero encountered in true_divide
  c *= np.true_divide(1, fact)
```

Out[43]:
```
title
An implemented model of punning riddles                      NaN
Genetic Algorithms in Time-Dependent Environments            NaN
MRI Tumor Segmentation with Densely Connected 3D CNN         NaN
Operations for Learning with Graphical Models                NaN
Reasoning with Individuals for the Description Logic SHIQ    NaN
Syntactic-Head-Driven Generation                             NaN
dtype: float64
```

In [44]:
```
df=pd.read_csv(f, encoding='latin-1')
df=df.dropna()
#for i in df['keywords']:
#    i=((" ".join(((i[1:len(i)-2]).split(","))))))
df['keywords']
```

Out[44]:
```
0    ['head', 'semantic', 'syntactic', 'driven', 'g...
1    ['jokes', 'joke', 'jape', 'model', 'words', 'u...
2    ['graphical', 'learning', 'models', 'framework...
3    ['rate', 'population', 'phase', 'mutation', 'g...
4    ['tbox', 'reasoning', 'implementation', 'algor...
5    ['tumor', 'gliomas', 'enhancing', '3d', 'model...
Name: keywords, dtype: object
```

In [45]:
```
# instantiating and generating the count matrix
count = CountVectorizer()
count_matrix = count.fit_transform(df['keywords'])

# generating the cosine similarity matrix
cosine_sim = cosine_similarity(count_matrix, count_matrix)
print(cosine_sim)
```

```
[[1.  0.  0.  0.  0.  0. ]
 [0.  1.  0.  0.  0.  0.1]
 [0.  0.  1.  0.  0.  0. ]
 [0.  0.  0.  1.  0.  0. ]
 [0.  0.  0.  0.  1.  0. ]
 [0.  0.1 0.  0.  0.  1. ]]
```

In [ ]:
```
```