Offline Webpage Search Tool

Although the author and publisher have made every effort to ensure that the information in this writing was correct at press time, the author and publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause.

This small program performs a similarity query against web pages that are stored in the local hard drive. It uses Latent Semantic Indexing (LSI) to rank the documents. Unlike a conventional searching tool, it calculates semantic relatedness of the texts. A document may appear in the search result even if the search query is not in the text. To make it easier, *Gensim*, an open source python library for NLP, is used for this app.

The first step to do is to convert the html pages to plain text. We utilize *BeautifulSoap* for this purpose. For simplicity, noises in the HTML pages (i.e. page navigation, table of content, etc) are not removed.

```
import os
from bs4 import BeautifulSoup, Comment


def get_file_paths(dir_name):
    for dir_path, dir_names, file_names in os.walk(dir_name):
        for file_name in file_names:
            yield os.path.join(dir_path,file_name)

def get_content(file):
    soup = BeautifulSoup(file, 'html.parser')
    soup = soup.body
    if soup == None:
        return
    for code in soup(['script']):
        code.extract()
```

```
    comments = soup.findAll(text=lambda text:isinstance(text, Comment))
    [comment.extract() for comment in comments]
    content = soup.get_text()
    return content

file_paths = get_file_paths('Pages')
for file_path in file_paths:
    if file_path.endswith('htm') or file_path.endswith('html'):
        file = open(file_path,'r',encoding='utf8')
        content = get_content(file)
        file.close()
        if content:
            file_name = os.path.basename(file_path)
            if file_name.endswith('htm'):
                file_name = file_name.replace('htm','txt')
            elif file_name.endswith('html'):
                file_name = file_name.replace('html','txt')
            file = open('Corpus/'+file_name,'w+',encoding='utf8')
            file.write(content)
            file.close()
```

Next, a dictionary is created based on the plain text files (corpus). In the dictionary, each word is assigned to a unique number.

```
import os
from gensim import corpora

def get_file_paths(dir_name):
    for dir_path, dir_names, file_names in os.walk(dir_name):
        for file_name in file_names:
            yield os.path.join(dir_path,file_name)

def make_dictionary(file_paths):
    texts = []
    for file_path in file_paths:
        file = open(file_path,'r',encoding='utf8',errors='ignore')
        text = file.read()
        text = text.lower().split()
        texts.append(text)
    dictionary = corpora.Dictionary(texts)
    dictionary.save('dictionary.dict')
    return dictionary

file_paths = get_file_paths('Corpus')
dictionary = make_dictionary(file_paths)

for dict in dictionary.items():
    print(dict)
```

The content of the dictionary file will be like this:

```
(18331, 'ticket')
(18332, 'tiers,')
(18333, 'tower')
(18334, 'town')
(18335, 'tray,')
(18336, 'trellis')
(18337, 'trick.')
(18338, 'ucla')
(18339, 'unassembled')
(18340, 'usc')
(18341, 'veggies')
(18342, 'vertical')
(18343, 'vida')
(18344, 'visuals')
(18345, 'wallet,')
(18346, 'washington')
(18347, 'watering')
(18348, 'wellness')
(18349, 'wide."')
(18350, 'williams')
```

The plain text files are now represented as a stream of vectors. Each document is mapped to the dictionary that contains all of the corpus's unique words. A simple term count modelling (bag-of-words) is implemented in the process. A file list (*file_list.csv*) is also created for displaying search results.

```
import os
import csv
from gensim import corpora

def get_file_paths():
    for dir_path, dir_names, file_names in os.walk('Corpus'):
        for file_name in file_names:
            yield os.path.join(dir_path,file_name)   #,'r',encoding='utf-8',errors='ignore'

def make_bow(file_paths,dictionary):
    file_write = open('file_list.csv','w+',newline='')
    writer = csv.writer(file_write)
    idx = 0
    for file_path in file_paths:
        file_read = open(file_path,'r',encoding='utf8',errors='ignore')
        content = file_read.read()
        file_read.close()
        content = content.lower().split()
        writer.writerow([idx,file_path])
        idx += 1
        yield dictionary.doc2bow(content)
    file_write.close()

file_paths = get_file_paths()
dictionary = corpora.Dictionary.load('dictionary.dict')
vectors = make_bow(file_paths,dictionary)

for v in vectors:
    print(v)

corpora.MmCorpus.serialize('bow.mm', vectors)
```

The BOW model is saved. Here is the content of the file:

```
(1011, 1), (1035, 2), (1038, 1), (1062, 1), (1072, 2), (1087, 1),
(1463, 1), (2188, 1), (2307, 1), (3187, 1), (3341, 2), (3378, 1),
(5860, 1), (5862, 1), (5863, 1), (5864, 1), (5866, 1), (5867, 1),
(5869, 1), (5870, 1), (5871, 1), (5872, 1), (5873, 1), (5874, 1),
(5876, 1), (5877, 1), (5878, 1), (5879, 1), (5880, 1), (5881, 1),
(5882, 1), (5883, 1), (5884, 1), (5885, 1), (5887, 1), (5889, 1),
(5891, 1), (5892, 1), (8327, 1), (9052, 1), (10930, 1), (12832,
1), (20344, 1), (20345, 1), (20346, 1), (20347, 1)]
```

We can now use the dictionary *(dictionary.dict)* and the vectorized document corpus *(bow.mm)* to create LSI model. Essentially, words that are commonly found together within a single topic are grouped together. The number of topics that you want to create is up to you.

```
from gensim import corpora, models

dictionary = corpora.Dictionary.load('dictionary.dict')
corpus = corpora.MmCorpus('bow.mm')

lsi = models.LsiModel(corpus, id2word=dictionary, num_topics=5)
lsi.save('model.lsi')
```

Below are the most contributing words for each of the first three topics. The results will be much better if we first clean the corpus by removing stop words.

```
[(0, '0.524*"the" + 0.395*"to" + 0.331*"and" + 0.311*"a" +
0.255*"of" + 0.191*"you" + 0.169*"is" + 0.154*"for" +
0.144*"your" + 0.143*"in"'), (1, '-0.355*"and" + -0.339*"of" +
0.261*"you" + 0.221*"your" + -0.214*"jump" + -0.209*"^" +
-0.198*"up" + 0.189*"to" + 0.175*"the" + -0.156*"to:"'), (2,
'-0.417*"{" + -0.410*"}" + -0.384*"#opm" + -0.328*">" +
-0.264*"#nav" + -0.170*"pos" + -0.150*"a" + -0.142*"li" +
-0.111*"cash" + -0.111*"0;"')]
```

Using the LSI model, we can represent any document as a distribution over topics.

```
from gensim import corpora, models, similarities

lsi = models.LsiModel.load('model.lsi')
corpus = corpora.MmCorpus('bow.mm')

corpus_lsi = lsi[corpus]
index = similarities.MatrixSimilarity(corpus_lsi)
index.save('index.index')
```

Below is the percentage contribution of topics in each document.

```
[(0, 80.23350674892399), (1, 1.3730768141841119), (2, -0.8501687222361713), (3, -6.021960915153211), (4, 9.056738621168893)]
[(0, 34.06191829908811), (1, 0.9749780291150096), (2, -5.181590683571151), (3, -4.463697601642491), (4, 3.578501336814819)]
[(0, 127.1011458641257), (1, 17.1551817404665), (2, -27.717898211382064), (3, -6.7184445276394555), (4, 11.883354863383419)]
[(0, 70.14730371346542), (1, 2.7896656449357664), (2, -10.289500379424233), (3, 1.2933605517375133), (4, -1.662002048854076)]
[(0, 264.1516772921563), (1, -38.020686542395836), (2, 7.701960389146188), (3, 11.238243151637342), (4, 28.039420372801466)]
[(0, 357.1479159864813), (1, -55.80119276377017), (2, 11.98443921474841), (3, 21.568303813361602), (4, 20.236292985176107)]
[(0, 385.91278048556467), (1, -68.72454753159215), (2, 6.478007153982141), (3, -12.569000162863762), (4, 109.90875307035374)]
[(0, 48.68552807272912), (1, -6.999547381018917), (2, -6.134152066124536), (3, 3.459819599207048), (4, 9.2933548043002)]
[(0, 58.51755581503092), (1, 6.004483558159249), (2, -5.765526848185348), (3, -1.0037499864588801), (4, 4.870932001061491)]
[(0, 75.00178527379711), (1, -0.7739747034948583), (2, -1.2136169631571137), (3, 3.68717727405751), (4, 1.3230202493612713)]
[(0, 80.24244288021227), (1, -23.016731291348528), (2, -4.8355759652199), (3, 14.859353243544593), (4, 11.371828767975938)]
```

Now, it is the time to perform the searching process. Input any keyword and the program will show the most relevant content (text files).

```
from gensim import corpora, models, similarities
import csv

lsi = models.LsiModel.load('model.lsi')
dictionary = corpora.Dictionary.load('dictionary.dict')
index = similarities.MatrixSimilarity.load('index.index')

user_input = 'pc cash register'
vec_bow = dictionary.doc2bow(user_input.lower().split())
```

```
vec_lsi = lsi[vec_bow]

sims = index[vec_lsi]
sims = sorted(enumerate(sims), key=lambda item: -item[1])

top = 0
for sim in sims:
    file_read = open('file_list.csv','r')
    reader = csv.reader(file_read)
    for rec in reader:
        if str(sim[0]) == str(rec[0]):
            print(str(rec[1]))
            break
    file_read.close()
    if top == 5:
        break
    top += 1
```

The results are numerical.

```
[(15, 0.8678724), (18, 0.38589263), (17, 0.36120132), (23,
0.3015579), (24, 0.2773644), (19, 0.2740904), (6, 0.26867273),
(22, 0.23732695), (28, 0.23089159), (16, 0.22934666), (13,
0.21374027), (1, 0.20992568), (35, 0.20143361), (8, 0.20019281),
(38, 0.19241111), (26, 0.19096313), (41, 0.18021046), (25,
0.16652937), (20, 0.15526325), (0, 0.1517339), (29, 0.13896051),
(21, 0.13023992), (36, 0.12367727), (9, 0.12203306), (7,
0.11837326), (37, 0.11030092), (12, 0.10811432), (14, 0.1001394),
(40, 0.0955003), (31, 0.079884), (2, 0.078371264), (4,
0.07591311), (39, 0.07450342), (11, 0.062778115), (10,
0.05777069), (30, 0.057719395), (32, 0.036349557), (5,
0.034659944), (3, 0.020521775), (33, 0.018504776), (34,
0.011736311), (27, 0.0)]
```

Here, with the help of the csv file (*file_list.csv*) that we created earlier, we can print the most similar documents in a better way.

```
Corpus\Convert your Mobile_Laptop_Ipad in to your business point
of sale system _ OnlyPOS.txt
Corpus\Essential tools for building, repairing, and upgrading PCs
(and other electronic devices) _ PCWorld.txt
Corpus\Darts Connect is the world's first smart dartboard _
Trusted Reviews.txt
Corpus\How to Turn Your Computer Into a Cash Register _
Bizfluent.txt
Corpus\How to Turn Your Computer Into a Cash Register _ Your
Business.txt
Corpus\Freeware Program To Turn A PC Into A Cash Register -
Howtopossystem.com 2019.txt
```

Reference:
https://radimrehurek.com/gensim/index.html