Title: Build a simple Vector Space Information Retrieval System

Author: Si Chen

Date: Mar 17 2017

Table of Contents

Description:

This system is a simple vector space information retrieval system to search over Wikipedia movie corpus using conjunctive ("AND") operation on user queries and output a ranked result list and present selected documents. This system is implemented from following aspects: construct a term frequency (count) matrix which record each the term frequency of each term in each document

- construct weight matrix with calculating the tf-idf weight of each term in each document and normalizing the weight by the length of documents
- Rank documents according to their proximity to the query using cosine similarity score of query and documents
- provide a Web UI for user to input key searching query and search the corpus, output a ranked result list
- implement a feature that user can access corpus contents by clicking title in the results' list
- implement a feature that user can refer the 10 most likely documents to specific document by clicking "more like this"

Package:

The imported package is the same as PA3.

- **Flask**
  Flask is a micro framework for python web development.
  <Build Instruction>
  Refer to http://flask.pocoo.org
  1> download flask version: 0.12
  https://pypi.python.org/pypi/Flask/0.12#downloads
  2> go to file downloaded folder and execute:
  $ sudo pip install Flask

- **NLTK**
  NLTK is a platform for python program to process human language data. This platform provides various corpus and libraries for processing tokenization, stemming and stop words. In this system, for normalizing and optimizing term I used the libraries:
    - RegexpTokenizer
    - SnowballStemmer
    - stopwords corpus (See APPENDIX stop words list)

  <Build Instruction>
  Refer to http://www.nltk.org/install.html
  1> install NLTK
  $ sudo pip install –U nltk
  2> install Numpy
  $ sudo pip install –U numpy
  3> install NLTK Data
  run python
  >>> import nltk
  >>> nltk.download()
  4> A new window should open, showing the NLTK Downloader. Select all the data and download.
  5> test installation
  >>> from nltk.book import *
  If all the texts until text9 are output successfully which means you installed nltk successfully.

Running Instruction:

1> **Create count matrix, weight matrix and doc-data (no need to execute again)**
python vs_index.py
<created file>
- shelv_index.db: the count matrix that records document term frequency in each document of each token (token unit)
- shelv_normalizedWeight.db: the normalized weight matrix that records document term normalized weight in each document of each token
- shelv_dtw.db: the count matrix that records document term frequency of each token in each document (document unit)
- shelv_dtwn.db: the normalized weight matrix that records document term normalized weight of each token in each document (document unit)

2> **Invoke the flask UI and allow users to query the index**
python vs_query.py

3> **Search films**
open http://127.0.0.1:5000 in browser and use the UI input key words in:
> search films text box
and click search button

4> **Confirm results**
There limits only 10 results can be output in one page, if there appears next and previous button you can click to access next 10 results or previous 10 results.

5> **Access detail contents**
Click the film title, there is a new window shows the detail contents or corresponding film

6> **Refer more like this movie**
Click the “more like this” button under each movie title, there is a new window shows the top 10 more likely movies’ title and several lines of introduction to this movie

Files in Folder

Before running:

- films_corpus.json
  The corpus of 2016 films from wikipedia
- test_corpus.json
  A handmade corpus to test how the system works in generating term frequency matrix, normalized document term weight.
- vs_index.py
  The module that creates index (shelve files) and doc-data
- vs_query.py
  The module that invokes the flask UI and allow users to query the index
- querysearch.py
  The module contains several functions for other module to implement their functions

After running:

- doc_data.json
  created data stores the information needed to present an article to the user
- shelv_index.db
  the count matrix that records document term frequency in each document of each token (token unit)
- shelv_normalizedWeight.db
  the normalized weight matrix that records document term normalized weight in each document of each token
- shelv_dtw.db
  the count matrix that records document term frequency of each token in each document (document unit)
- shelv_dtwn.db
  the normalized weight matrix that records document term normalized weight of each token in each document (document unit)

System Function details

In this system, I calculate the term frequency by normalizing token, calculating document term tf-idf weight, calculating query term tf-idf weight and calculating query-document cosine similarity score.

Normalizing token

1. Tokenization
   For removing punctuation and space in the text, I used tokenization to extract word from sentences and graphs. In this system, I used regular expression to define the style of word and process the tokenization.
2. Case Folding
   For voiding case affect, in this system, I converted all the term into lower case.
3. Stop words control
   Before processing word, check whether the word exists in stop words list. If it is, add the word into stop words list and output as ignored term.
4. Stemming
   In this sytem, I used SnowballStemmer to do the stemming of the word.

Calculating document term tf-idf weight

In this system, to evaluate the weight of term in documents, I used tf-idf weight as following formula.

$$(1 + \log(tf)) * (log\frac{N}{df})$$

where,

- tf is term frequency which is the count of times the normalized term appears in this document.
- N is the number of documents in this collection.
- df is document frequency which is the number of how many documents the normalized term appears.

To implement the calculation of document term tf-idf weight, I implemented following function in my system.

1. collect document term frequency in this document-> tf
2. collect how many document does this term appears in all documents in this collection -> df
3. calculate tf-idf weight

Calculating document length

For calculating the length of document, square all the term weight in this document and take the root value of the sum.

$$document\ length = \sqrt[2]{w_1^2 + w_2^2 + w_3^2 + w_4^2 + \cdots}$$

Normalizing document term tf-idf weight

With the length of document, normalize each document term weight in the document with corresponding document length and store them into shelv_normalizedWeight.db

Abstracting query term weight

Calculate the query term weight for the normalized term in query by the formula as below. For calculating the weight, I also used idf to evaluate the informativeness of this term.

$$(1 + \log(tf)) * (log \frac{N}{df})$$

Calculating the cosine similarity score between query and document

In order to calculate the similarity, summarize all the production of term weight in query and document.

$$\sum_{q,d} w_{t,q} \times w_{t,d}$$

Extract top 30 high score results

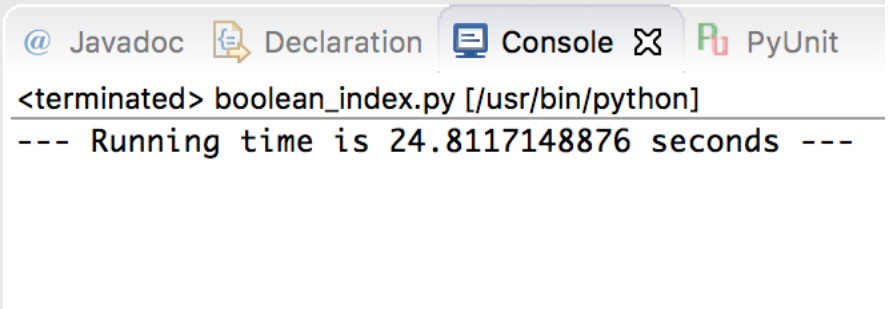
Using heap to store only 30 results with the 30 highest similarity score and output the results in the descending order of similarity score in SERP.html

Optional: more like this

In this part, I used all the terms in documents as query to find other documents with high similarity score. Since we need to find likely documents, I used disjunctive operation to search similar documents. Also, there is no need to output all similar documents, in this part I only output the top 10 document with highest similarity score.

Index constructing time

the time used to build all shelve files on my local machine is shown as below



Test cases

1. test data preparation:

For testing the function can operate correctly I prepared the test cases with 14 documents, and the token in this document collection are “test”, “title”, “huge” and “apple”. The term frequency is as below:

token	document term frequency													
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
test	1	2	3	4	5	1	1	1	1	1	1	1	0	0
title	1	1	1	1	1	6	7	8	9	10	1	1	0	0
apple	0	0	0	0	0	0	0	0	0	0	0	1	2	2
huge	0	0	0	0	0	0	0	0	0	0	0	1	2	1

2. Calculate tf-idf weight and normalized weight manually

In order to examine the correctness in calculating weight of document term with length normalization.

Thus the weight of document term before length normalization should be like this according to the tf-idf formula:

$$(1 + \log(tf)) * (log \frac{N}{df})$$

token	document term weight before normalization				
	1	**11**	**12**	**13**	**14**
test	0.06695	0.06695	0.06695	0	0
title	0.06695	0.06695	0.06695	0	0
apple	0	0	0.6690	0.8704	0.8704
huge	0	0	0.6690	0.8704	0.6690

Therefore, the length of each document is calculated using the following formula:

$$\sqrt[2]{w_{test}^2 + w_{title}^2 + w_{apple}^2 + w_{huge}^2}$$

document	length
1	0.09468
11	0.09468
12	0.95083
13	1.2309
14	1.0978

Next, normalize the weight of document term with length of each document

token	document term weight before normalization				
	1	**11**	**12**	**13**	**14**
test	0.70711	0.70711	0.07041	0	0
title	0.70711	0.70711	0.07041	0	0
apple	0	0	0.70359	0.70711	0.79286
huge	0	0	0.70359	0.70711	0.6094

Finally, with the weight or query term without length normalization, we can calculate the similarity between query and document using following formula:

$$\sum_i w_{qi} * w_{di}$$

test	query					query document similarity score				
	test	title	apple	huge	search	1	11	12	13	14
1	1	0	0	0	0	0.0473	0.0473	0.0047	0	0
2	0	1	0	0	0	0.0473	0.0473	0.0047	0	0
3	0	0	1	0	0	0	0	0.4707	0.4731	0.5304
4	0	0	0	1	0	0	0	0.4707	0.4731	0.4077
5	0	0	0	0	1	0	0	0	0	0
6	1	1	0	0	0	0.0947	0.0947	0.0094	0	0
7	1	1	1	1	0	0.0947	0.0947	0.9508	0.9461	0.9381
8	1	1	1	1	1	0.0947	0.0947	0.9508	0.9461	0.9381
9	5	1	0	0	0	0.1278	0.1278	0.0127	0	0
10	0	0	2	1	0	0	0	1.0831	1.0885	1.0978

The red score which the score can be calculated, but in conjunction search the document which does not contain all the token in query will not be chosen as the search result, thus, these score in red will not be shown in the result.

The query document similarity score's result calculated by the system are below:

1> query: test

0.0473385289265	title	test
0.0473385289265	title	test
0.00471357413586	apple title	huge test

2> query: title

0.0473385289265	title	test
0.0473385289265	title	test
0.00471357413586	apple title	huge test

3> query: apple

SCORE	TITLE	TEXT
0.530426891256	apple apple	huge
0.473059231476	apple apple	huge huge
0.470708315143	apple title	huge test

4> query: huge

SCORE	TITLE	TEXT
0.473059231476	apple apple	huge huge
0.470708315143	apple title	huge test
0.407697664945	apple apple	huge

5> query: search

Unknown search term: search

6> query: test title

SCORE	TITLE	TEXT
0.094677057853	title	test
0.094677057853	title	test
0.00942714827171	apple title	huge test

7> query: test title apple huge

Search films

test title apple huge

Optional

director: starring: location:

Search

Find 1 results. This page shows the result from 1 to 1

SCORE	TITLE	TEXT
0.950843778557	apple title	huge test

8> query: test title apple huge search

Search films

test title apple huge search

Optional

director: starring: location:

Search

Find 0 results. This page shows the result from 1 to 0

Unknown search term: test title apple huge search

9> query: test test test test test title

0.127765269622	title	test
0.127765269622	title	test

10> query: apple apple huge

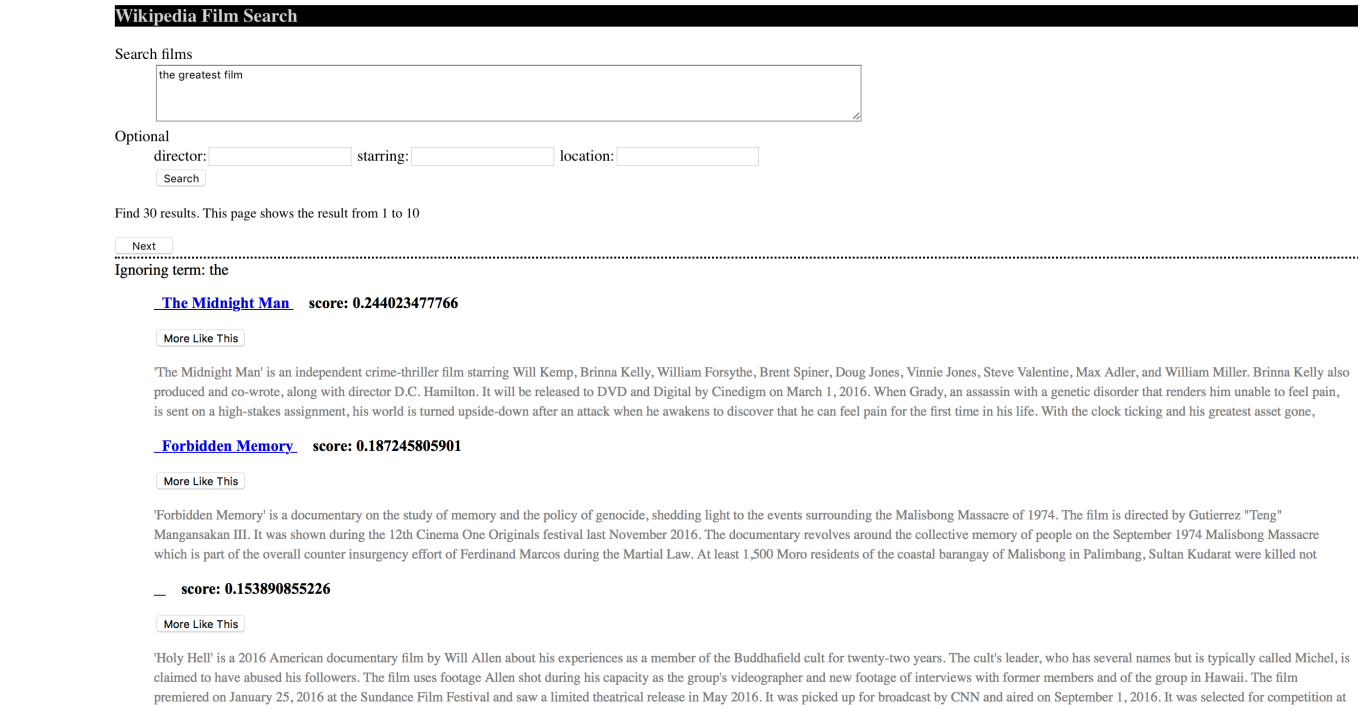
SCORE	TITLE	TEXT
1.09779896098	apple apple	huge
1.08852348135	apple apple	huge huge
1.08311395235	apple title	huge test

Test conclusion:

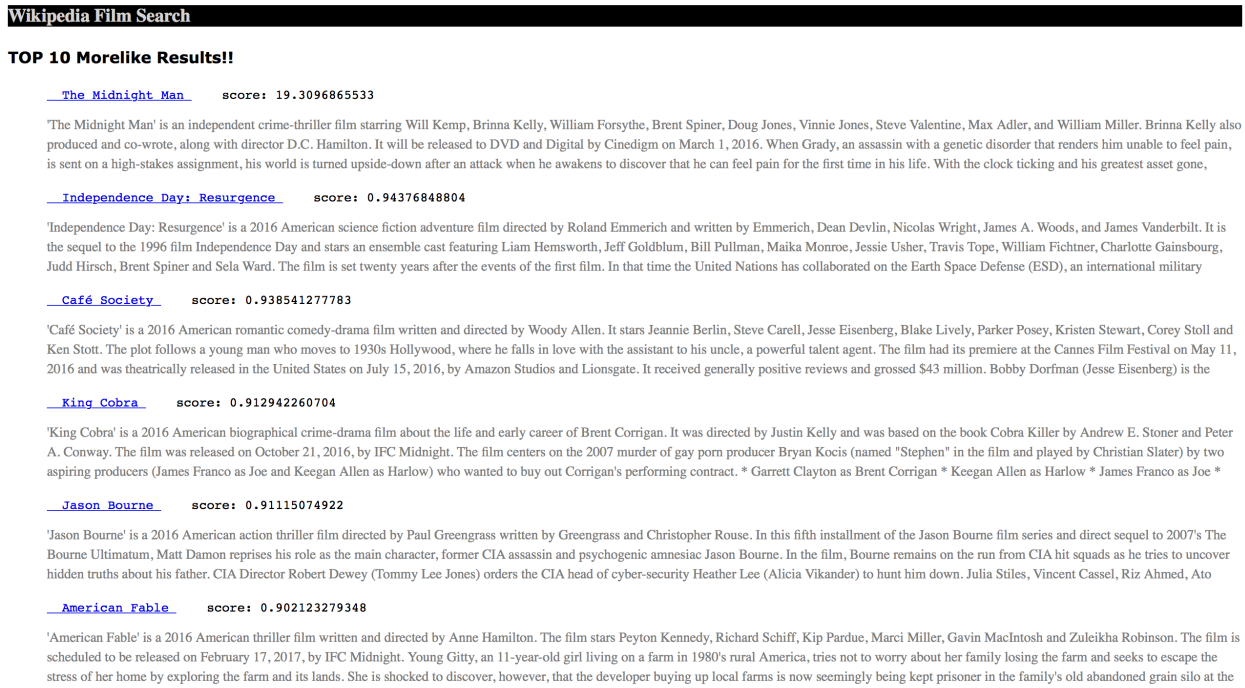
1. The scores in (1) ~ (10) are calculated as correctly as the result calculated manually.
2. Different queries will result in different ranking. (3) and (4)
3. The term frequency in query can change the ranking result according to (6) and (9). This also denotes the direction of vector of query and document is a main factor to determine cosine similarity score.
4. If one of the tokens in the query does not exist in the collection of documents as (5) and (8), according to conjunction match, there is no result will be returned.
5. The results are ranked by the descending order of cosine similarity score as shown in (10)

System output results

1. the ranked research results with similarity score



2. more like this to show the top 10 most similar movies



APPENDIX stop words list

a
about
above
after
again
against
ain
all
am
an
and
any
are
aren
as
at
be
because
been
before
being
below
between
both
but
by
can
couldn
d
did
didn
do
does
doesn
doing
don
down
during
each
few
for
from
further
had
hadn
has
hasn
have
haven
having
he
her
here
hers
herself
him
himself
his
how
i
if
in
into
is
isn
it
its
itself
just
ll
m
ma
me
mightn
more
most
mustn
my
myself
needn
no
nor
not
now
o
of
off
on
once
only
or
other
our
ours
ourselves
out
over
own
re
s
same
shan
she
should
shouldn
so
some
such
t
than
that
the
their
theirs
them
themselves
then
there
these
they
this
those
through
to
too
under
until
up
ve
very
was
wasn
we
were
weren
what
when
where
which
while
who
whom
why
will
with
won
wouldn
y
you
your
yours
yourself
yourselves