Assignment 3: Boolean Information Retrieval System

Description:

Supported by the 2018_movie corpus, the program generate inverted_index for each fields of the movie page, offers conjunctive queries over the terms in specific field, and returns search results with snippets and movie descriptions.

Dependencies:

MacOS High Sierra version 10.13.2
Python 3.6
nltk==3.3
Flask==1.0.2

Run Instructions:

Execute the following command to generate the inverted_index shelves, movie description shelves, and test_corpus.json:

```
python3  boolean_index.py
```

Execute the following command to run the Boolean information retrieval system:

```
python3  boolean_query.py
```

Modules and methods:

preprocessing.py

PreProcessing

A class contains the methods to preprocess the text loaded from corpus, which is later used for building inverted index.

__init__(self):

contains nltk stopwords list and most frequent and unhelpful terms from the text

flatten(self, x):

leave 1D list unchanged, strings to a list, multi-D list to 1D list
:param x:
:return: 1D list

normalize(self, token):

do case-folding, removing stopwords, and stemming

test_corpus(self, filename='test_corpus.json')

Create a test containing 10 hand-made documents corpus in json file

boolean_index.py

This module generate inverted index for each query box, generate the article shelve, offer conjunctive query.

main_query_inverted_index(shelvename1,shelvename2,corpus_name):

create a title+free text inverted index, and put it into 2 shelve files
:param shelvename1:
:param shelvename2:
:param corpus_name: a json file

fields_inverted_index(shelvename, field, corpus_name):

create a shelve containing the inverted index of given field, from the json file
:param shelvename:
:param field:
:param corpus_name:

article_shelve(filename='2018_movies.json', shelvename='article_shelve'):

Get all the doc data(displayed after clicking on a film title in the result page) and store it in a shelve.

intersection(posting_lists):

:param posting_lists: a list of multiple postings lists to be intersected, the postings are sorted
:return: a list containing intersected docIDs

conjunctive_query(tokens, shelve_name):

:param tokens: tokenized query
:param shelve_name: the shelve that contained inverted index of certain field
:return: possibly a list of matching docIDs, a list of stopwords, and a list of unknown words

boolean_search.py

This module offer the functionality of returning the searched movie ids, movie field data and movie snippets

dummy_search(query, shelve):

return a list of movie ids that match the query.

dummy_movie_data(doc_id, shelvename='article_shelve'):

return data fields for a movie.

dummy_movie_snippet(doc_id):

return a snippet for the results page

boolean_query.py

query():

generate the welcome page

results(page_num):

Generate a result set for a query and present the 10 results starting with <page_num>

movie_data(film_id):

Given the doc_id for a film, present the title and text and structured fields for the movie

Files in the folder:

2018_movies.json: the movie corpus that supported the boolean IR system
test_corpus.json: a hand-made ten documents corpus

Text_inverted_index_1.db, Text_inverted_index_2.db: inverted index with dictionary terms from the freetext and title

Director_inverted_index.db: inverted index with dictionary terms from the Director field

Starring_inverted_index.db: inverted index with dictionary terms from the Starring field

Location_inverted_index.db: inverted index with dictionary terms from the Location field

movie_page.db: dictionary containing the movies information, including information from Title, Text, Director, Starring, and Location.

templates/query_page.html, templates/results_page.html, templates/error_page.html, templates/doc_data_page.html: html files used to create the web page

Text normalization details:

nltk.stop_words, most frequent words that potentially appear in every movie and not helpful with queries are removed from the tokens. The tokens are converted into lowercase and stemmed.

Testing:

Utilizing the test_copus created in proprocessing.py, different words with same stems are all matched. Only the matched movie titles and snippets are returned in the search result page. The movie page does contain the title, director, starring, location and text.