README

Jianing Li (JL5543) Donghai Xu (DX2193)

1. A list of all the files that you are submitting
2. Provide all commands necessary to install the required software and dependencies for your program.
3. A clear description of the internal design of your project, explaining the general structure of your code (i.e., what its main high-level components are and what they do), as well as acknowledging and describing all external libraries that you use in your code
4. A detailed description of your query-modification method (this is the core component of the project); this description should cover all important details of how you select the new keywords to add in each round, as well as of how you determine the query word order in each round

Test cases

1. Look for information on the Per Se restaurant in New York City, starting with the query [per se]. -> 2 v.s.
2. Look for information on Google cofounder Sergey Brin, starting with the query [brin]. -> 2
3. Look for information on the animal jaguar, starting with the query [jaguar]. -> 4 v.s. 2

Introduction

We made this command line interface program of relevance feedback on search results of Google using Python (version 3.6). All the codes are put in one file “main.py”. As followed we present a list of functions we implemented.

Project structure

Files structure

- proj1/feedback.py: the main pipeline and it also contains all helper functions.
- README.pdf: the file you are reading.
- test_cases_transcript.pdf: transcript of the 3 test cases

Project dependency

Below are the python packages this program depends on:

- requests: for making Custom Search API calls.

- sklearn: for word tokenization and vectorization
- beautifulsoup4

Important Function Signature

def get_precision()
    Calculate the precision based on feedback
    :param results: list of results(10)
    :return: precision

def generate_new_words()
    :param: search item list [{ ‘title’, ‘url’, ‘description’, ‘relevant’} …]
    :return: two new words

def get_doc_list()
    :param: search item list [{ ‘title’, ‘url’, ‘description’, ‘relevant’} …]
    :return: ls_relev
    :return: ls_irrel

def rocchio_relevance_feedback()
    Using Rocchio algorithm to update query using relevance feedback.
    :param query: string of query
    :param ls_relev: list of relevant documents strings
    :param ls_irrel: list of irrelevant documents strings
    :param weights: (a,b,c) weights for query, relevant and irrelevant terms respectively.
    :return: dict of (token -> weight in the resulting vector)

Steps to run the program

1. Environment setup

```
pip3 install sklearn requests beautifulsoup4
```

2. Run the program
    a. Unzip the project
    b. Change directory to root of this project
3. Run the feedback.py script

python3 feedback.py

4. Type in the query, precision and keys sequentially according to the prompts instructed.

Program design

First, we parse the json object response from HTTP request. We need the “item” dictionary and get “title”, “url” and “summary”.

Interactive interface

The script will show the query result from Google Custom Search API and interactively prompt the user to mark the top 10 results as relevant or not. The user can respond to these prompts by entering Y if the result is relevant or N if the result is not relevant.

```
Google Custom Search API Result:

======================

======================

Result 1 is as following:

 Title: Jaguar Sedans, SUVs and Sports Cars - Official Site | Jaguar USA

 URL: https://www.jaguarusa.com/index.html

 Description: The official home of Jaguar USA. Explore our luxury sedans, SUVs and sports

cars. Build Your Own, Book a Test Drive or Find a Retailer near you.

Is This Document Relevant? (Y/N)
```

After the user does this for all 10 results the *precision@10* is computed and displayed. If the *precision@10* is below the target precision and greater than 0, then we use rocchio algorithm to generate new key words (following for detail) as well as a new query is displayed and executed, with a new round of 10 relevance prompts displayed to the user.

Query-modification

For the 10 labelled results from each query, we partition the doc into relevant and irrelevant docs. Each doc includes words of summary and title (we ignore the normally meaningless words in url and “.html” files) During our test when we set number of iterations as parameter to judge the performance, we conclude that the words in summary and title is more useful to get fast convergence to satisfactory result. So these words separated by while space(also remove stop words) are combined into relevant and irrelevant doc strings as input for Rocchio algorithm. (detailed description is following)

Tokenization and vectorization

We used TfidfVectorizer of sklearn package to transform the list of documents into a matrix. The rocchio_relevance_feedback function takes original query and two lists of text, one for relevant documents and the other for irrelevant ones. Only the snippets of each search result are taken as text.
Within the pipeline, first the text collection is de-capitalized and tokenized (including only the alphabet words with at least 3 letters) the tf-idf weights for every tokens are calculated. Then the base query, relevant documents and irrelevant documents are each transformed into a vector (each with a length of 3206).

Rocchio algorithm

Then the pipeline would apply the Rocchio algorithms, with super-parameters (a, b, c) = (1, 0.75, 0.25), adding up those vectors. The resulting vector would be zipped into a dictionary, where each word is mapped to its weight in the vector.

Generating new query

In the end, the two words that have highests weight and are not included in the base query are added to the query. As the user query words can not be deleted, we append the two new words of highest weights(sorted in weights decreasing order) to the old query.

Credentials for testing

```
Search engine ID:
001155980652667309729:bgrnl8tabzj

API key:
AIzaSyCuBu6cTmO0GQaAvxSY991zB_nntU1aCB0
```