Documentation for the QA recommender system, CS 410

Group: chadyuu

1 Team member

Captain: Yutaro Nishiyama (yutaron2@illinois.edu)

2 An overview of the function

This Google Chrome extension retrieves three Q&A pages from Stack Overflow that relates the most to a current page on Stack Overflow about text retrieval, text mining, and NLP.

We can deepen our understanding from related Q&A discussions, which shed light on the topic from different perspectives. However, we usually feel tedious to search for related questions. The Google Chrome extension developed in this project requires only one click to realize this and will help us enhance our understanding much easier.

3 Create the Q&A corpus of Stack Overflow

The codes are in the `extension` folder. Below are the technical details to implement the Chrome extension.

3.1 Create a SQLite database

First, we created a table `questions` in the SQLite database as `tis.db` to store Q&A data from Stack Overflow.

```
CREATE TABLE "questions" (
    "id"    INTEGER,
    "link"  TEXT,
    "text"  TEXT,
    PRIMARY KEY("id")
)
```

3.2 Retrieve related Q&A URLs from Stack Overflow

Then, we executed `get_link.py` to retrieve Q&As on Stack Overflow that relate to text information systems, using Stack Exchange API (`/search/advanced`).

```
py get_link.py
```

Specifically, we set the following keywords as queries for the API.

```
keywords = [
      "text mining"
     ,"text retrieval"
     ,"text information"
     ,"natural language processing"
     ,"BM25"
     ,"POS tagging"
     ,"pivoted length normalization"
     ,"document frequence"
     ,"term frequence"
     ,"query likelihood"
     ,"PL2"
     ,"vector support machine"
     ,"TF-IDF"
     ,"inverse document frequency"
     ,"okapi"
     ,"search engine"
     ,"zipf's law"
     ,"f-measure"
     ,"normalized discounted culumative gain"
     ,"unigram"
     ,"dirichlet prior smoothing"
     ,"Jelinek-Mercer"
     ,"Rocchio Feedback"
     ,"Kullback-Leibler"
     ,"pagerank"
     ,"hypertext induced topic search"
     ,"content based filtering"
     ,"beta-gamma threshold learning"
     ,"collaborative filtering"
]
```

The question IDs and URLs are stored in the `questions` table.

3.3 Retrieve texts from Q&A pages

Next, we executed `get_text.py` to extract texts from each Q&A page.

```
py get_text.py
```

To extract main contents from Stack Overflow pages, we defined the `html_text` function in `common.py`. Here, we utilized `BeautifulSoup`, a Python package for parsing HTML documents, and extracted the content with `id=questions-header` and `id=mainbar` as main contents. The extracted texts are stored in the `questions` table.

3.4 Generate corpus

Then, we executed `generate_corpus` to generate corpus from the Q&A documents.

```
py generate_corpus.py
```

In this script, we first preprocessed the text of each Q&A page by the `preprocess` function defined in `common.py`. It utilized `nltk`, the natural language toolkit or a suite of libraries for NLP in Python, to remove punctuation, convert to lower case, remove stop words and numbers, and lemmatize.

From the preprocessed texts, we generated `dictionary` which contained word and ID information by `gensim`, an open-source library for NLP in Python.

Finally, we created `corpus` which stores word frequency information for each Q&A document.

To access these data, we saved three objects of `dictionary`, `corpus`, and `ids` which contains question IDs, using the `pickel` module to covert them into a byte stream.

4 Implement the REST API server

We implemented the REST API sever that returns the most related Q&As in `app.py`, which is composed of `Flask`, a micro web framework in Python, and the calculation of cosine similarity by `gensim`.

4.1 The web application using Flask

Our web application accepts the parameter `link` as the current Stack Overflow URL to the path of `/get` and returns the three most related Q&A questions IDs. Below is the core codes of Flask:

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/get')
def index():
  link = request.args.get('link', default = '', type = str)
  ...
  return top_ids

if __name__ == "__main__":
```

```
    app.run(debug=True)
```

We can run the API server by the following command.

```
flask run
* Running on http://127.0.0.1:5000
```

Now we can access the server on http://127.0.0.1:5000.

4.2 Calculate cosine similarity

The `app.py` script also calculates the cosine similarity between the current Stack Overflow page and the Q&A corpus, using `gensim.models.TfidfModel` and `gensim.similarities.Similarity`, after loading three objects `dictionary`, `corpus`, and `ids` with `pickle`.

5 Implement the Chrome extension

We implemented the extension in the `extension` folder with the following files:

- `index.html`: call `open.js`
- `open.js`: open `questions.html` in a new tab with the parameter of current Stack Overflow URL.
- `questions.html`: call `fetch_qa.js` and display three links to the most related articles.
- `fetch_qa.js`: fetch the four most related Q&A articles.
- `manifest.json`: specify basic metadata for the extension.

The Chrome extension has several limitations to improve user experience; the popup disappears after one click and we are not allowed to open multiple windows. Therefore, we have to open another page (i.e., `questions.html`) that displays the three links to related Q&A articles instead of showing them directly on the popup on `index.html`.

5.1 `index.html`

This html just calls `open.js`.

5.2 `open.js`

This script opens a new tab of `questions.html` with a parameter of `current_url`. Note that we should get the current Stack Overflow URL by the `chrome.tabs.query` API because the chrome extension cannot get it directly.

```
chrome.tabs.query(
    { active: true, lastFocusedWindow: true },
    tabs => {
    current_url = tabs[0].url # get the current URL of Stack Overflow
    ...
```

5.3 `questions.html`

This is a simple HTML file with `<ul id = 'data'></ul>` in the body element, which will be filled with the three links to the most related articles.

5.4 `fetch_qa.js`

This JavaScript fetches the four most related Q&A articles from the API server running on http://127.0.0.1:5000. Note that we retrieve four articles, not three, taking into account the case where they include the same article as the current page. Then, the script appends three `li` elements of the links for the most related questions to `<ul id = 'data'></ul>`.

6 How to install the Chrome extension

1. First, clone the repository.

```
git clone https://github.com/chadyuu/QA-recommender-extension.git
```

2. Open chrome://extensions/ on Google Chrome.
3. Turn on the developer mode by checking "Developer mode" checkbox in the top right-hand corner.
4. Click "Load unpacked" and select the `QA-recommender-extension/extension` folder.

7 The usage

Here is one instance for the usage of the extension.

1. Launch the API server. TODO: docker?

```
cd QA-recommender-extension/extension
flask run
```

2. Open a page with a title "BM25 Similarity in Elasticsearch" on Stack Overflow.
3. Click the QA recommender extension on the top-right of the Chrome browser.
4. You can see the following three most related Q&A articles in the opened tab.
    1. BM25 Similarity Tuning in Elasticsearch

2. How can I configure my index to use BM25 in ElasticSearch using the JAVA API?
3. elasticsearch 5.0 and index template

You can confirm that all of three articles similar to the original one. Please try this Chrome extension on other Stack Overflow articles!