Vector Space Retrieval

Gohil Dwijesh 2017CS50407

October 2020

1 File Layout

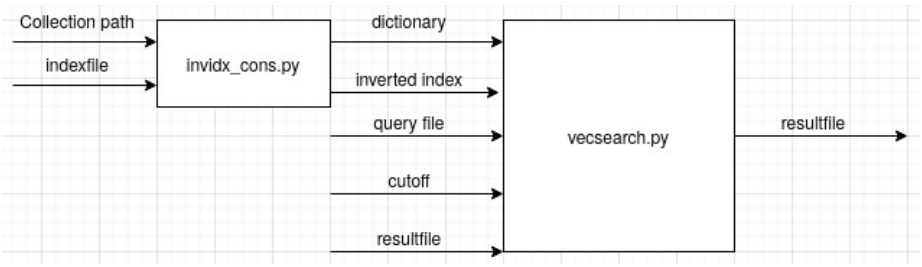

Figure 1: Input and Output description of invidx_cons.py and vecsearch.py files

2 Implementation Details

2.1 invidx_cons.py

consider Figure-2 for the following explanations.

- `DocumentProcessing` class is the hero of invidx_cons.py file. Given command line arguments as parameters in constructor, it does end to end task. Starting from reading training dataset docs to dumping the vocabulary and posting list.
- `DocumentProcessing` class uses `Parser` and `PostingListParser` class (which are inherited from `HTMLParser` class) for parsing the training dataset docs.
- `Parser` class' task is to maintain unique words seen while parsing the training dataset docs.
- On parsing a tag of <PERSON >, <LOCATION >, OR <ORGANIZATION >`Parser` class appends 'person', 'location', or 'organization' respectively to corresponding word. For example `Parser` class will map <PERSON >Rahul </PERSON >to Rahulperson.

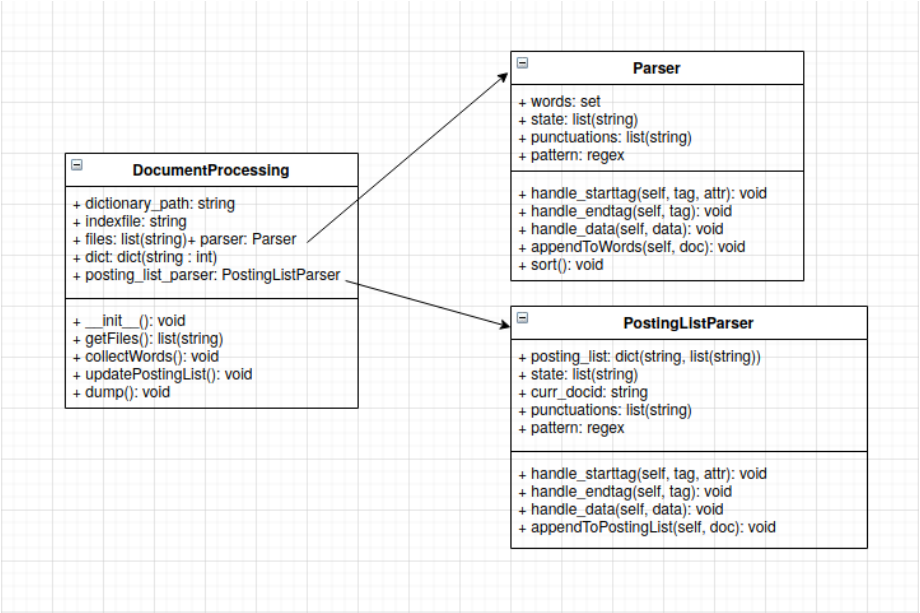

Figure 2: Classes and their dependencies in invidx_cons.py file

- `PostingListParser` class' task is to store posting list for all unique words stored in `Parser` class. It does so by traversing the all training dataset docs.
- Finally the *dump()* method in `DocumentProcessing` class will write vocabulary and posting list (wich are stored in different parser classes) to disk in json format.
- Experiments with different formats to store data on disk revealed that *json* format among *json*, *numpy*, *pickle*, and *shelve* is better.

2.2 vecsearch.py

Consider Figure-3 for following explanation.

- `TfIdf` class is the hero of vecsearch.py file. Given command line arguments as input parameter to its constructor, it does end to end job. Starting from reading data from disk to dumping relevant ranked docs to disk.
- `TfIdf` class first manipulates the query for tagging. Queries are tagged with same tags as in training dataset docs. Once tagged, tags are concatenated the same way they are done for training dataset docs(explained above with example).

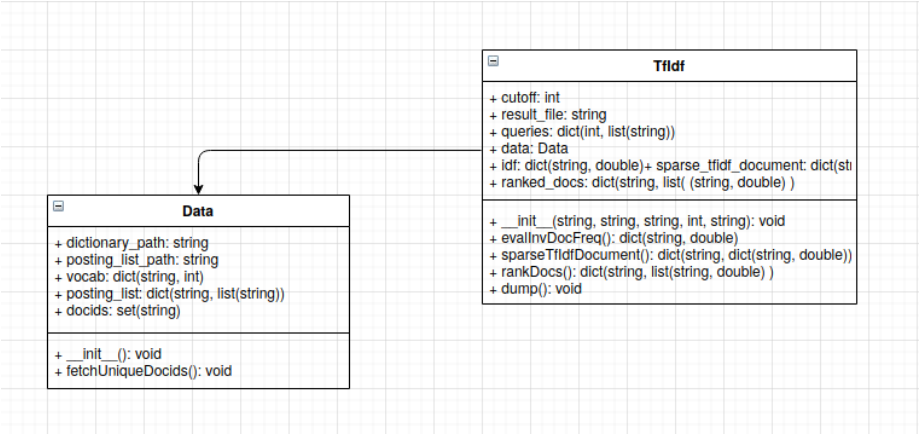

Figure 3: Classes and their dependencies in vecsearch.py file

- `Data` class' task is to store vocabulary and posting list. It also stores all unique document ids seen in the posting list.
- Next `TfIdf` class calculates inverse document freq for all words in vocabulary(stored in `Data` class). `TfIdf` class also calculates normalized sparse tfidf vector for all documents.
- For ranking purpose, `TfIdf` class considers cosine of angle between query vector and document vector. Calculating exact score value ranking is not necessary. For a given query, query vector is constant for all documents. Ignoring query vector, and summing normalized tfidf values in docs for words in query will give the same relative results that the original method would have given. `TfIdf` class exploits this property.
- Finally `TfIdf` class writes retrieved documents in a format that *trec_eval* tool accepts.

3 Tuning

- **Words** in queries and training dataset docs are transformed to **lower case**. Each such word is **stemmed** using `NLTK` library. **Stop words** and **punctuations** are also removed.
- **Compressing** occurs at two levels. invidx_cons.py file writes dictionary in `JSON` format. Then both indexfile.dict and indexfile.idx are compressed using **7z** compressor.