Information retrieval and pattern recognition

Denis Festa

University of West Bohemia
Faculty of Applied Sciences

June 7, 2020

Introduction

Scopes

What is this project about? The analysis of unstructured data (like text files, different from databases which are considered structured data). With analysis I mean that the user will be able to write a query and as an answer he will receive what is the document in the collection of data that is most likely the one he was looking for (somewhat like a search engine, of course not as complex as a real search engine would be).
The other thing that the user can get is the topic that is more likely to be dealt with from the query he gave (as a text classifier would do, only, this is not as complex as a real classifier would be).
Due to the presence of the classifier feature, the user can also write as a query a document ID instead of writing the whole text of the document, but the result would conceptually be the same.

Data

As I wrote about data, those data are a corpus of English documents called Reuters-21578, a collection of documents that appeared on Reuters newswire in 1987, available at http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html. All of the documents have an ID, but if you look at any of those articles (in the reut folder) you will see two XML attributes called OLDID and NEWID, the latter is the one I used.

Setting up

If you didn't receive 3GB of data within the source code, then it's likely that you will have to generate them manually. How to do that is explained in the comments in the source code,
mainly it's about calling functions inside the .py files, I suggest you to call them from a python console.
The two most important files from which you will have to call the functions are:

- indexer.py: this one will create indexes and dictionaries that will be used by:
  - Naive Bayes classifier
  - Optimized cosine distance classifier

- encoder.py: this one will need some results from the previous one, so it's easier if you follow this order; it will create for each document in the collection (7063 are the one I used for training) three different embedded vectors:
  - One hot / bag of word encoding,
  - Term frequency encoding,
  - tfidf encoding.

  The encoder.py will be useful for:
  - Cosine distance classifier,
  - Rocchio's classifier,
  - Neural network classifier.

The two files indexer.py and encoder.py are the most important in relation to the creation of necessary data, but there are still other things needed, they will explained inside each of the .py files related to each specific classifier, for example, in rocchio.py there are two functions to be used, that are

- write_encoded_vectors_for_all_topics()
- write_all_centroids().

Natural language processing

Before doing any computation on data, every document undergoes a small preprocessing, that is:

- lowercase,
- lemmatization,
- filtering out stopwords.

The last two are done using NLTK library: even if the imports are already done in the source code, it's likely that you'll face an error about nltk library, it will ask you to download WordNetLemmatizer and stopwords giving you the command to do that, it will be something like nltk.download().
Another thing you'll need is BeautifulSoup4 library, which is used to read the content of the Reuters collection, which is provided in .sgm format (somewhat similar to XML).
Everything related to preprocessing and handling .sgm files can be found in preprocesser.py and navigator.py .

Naive Bayes

Naive Bayes

Dependencies:

- topic_inverted_index.json
- topic_nonverted_index.json
- topics_dictionary.json
- topics_frequencies.json
- dictionary.json

All of those can be created using indexer.py

Optimizes cosine distance

Optimized cosine distance

Dependencies:

- inverted_index.json
- idf.json
- inv_tfidf.json

All of those can be created using indexer.py
Another resource that is not conceptually related to cosine distance but it's used in main.py to get the topics from the closest documents to the query is

- document_topic_dictionary.json, that you can create using the function write_document_topic_dictionary() inside indexer.py .

Cosine distance

Cosine distance

Dependencies:

- dictionary.json
- idf.json
- all the vectors: three vectors for each document in the collection (7063 documents were used for training).

The first two can be created using indexer.py.
The vectors can be created using write_encodings() from encoder.py.
Another resource that is not conceptually related to cosine distance but it's used in main.py to get the topics from the closest documents to the query is

- document_topic_dictionary.json, that you can create using the function write_document_topic_dictionary() inside indexer.py .

Rocchio

Rocchio

Dependencies:

- dictionary.json
- idf.json
- document_topic_dictionary.json
- all the vectors: three vectors for each document in the collection (7063 documents were used for training), they can be created using write_encodings() from encoder.py
- use write_encoded_vectors_for_all_topics() from rocchio.py
- use write_all_centroids() from rocchio.py .