CSI 5360 Natural Language Processing

Spring 2018

Programming Homework 1

Due date: 2/5 (Mon) 11:59pm

In NLTK, there is a basic class called CorpusReader which allow one to read a set of text files into a structure that can be manipulated. You can see the following documentation for examples:

- http://www.nltk.org/api/nltk.corpus.reader.html
- http://www.nltk.org/howto/corpus.html

The basic corpus class in NLTK does not support TF-IDF representation of document. Your task for this project is to build a class that incorporate a corpus reader that also produce the corresponding TF-IDF representation.

TF-IDF: overview

(All logarithms used is of base 2, unless otherwise stated)

Given a corpus, with n documents and m distinct words, we can define a vector $v_i = (w_i)$ for each document j, such that for term i, the weight w_{ij} = (term frequency of i in document j) * (inverse document frequency of term i)

As indicated in class, there are multiple ways of defining term frequencies and inverse document frequnecies. For this program you are to work with the following variations:

- Term frequency: raw, log (log normalized), binary
- Inverse document frequency: inverse, smoothed (inverse smoothed), probabilistic (pro babilistic inverse)

There are other consideration as well

- whether stopwords should be removed
- whether stemming should be done
- whether cases should be ignored

To-do list

You are to define a class called CorpusReader_TFIDF. The class will take a corpus object in NLTK and construct the td-idf vector for each document.

The specific details of the class is as follows:

- Constructor: Your constructor should have the following parameters
    - corpus (required): should be a corpus object

    - tf (keyword): the method used to calculate term frequency. Default is frequency (use the name specified above)
    - idf (keyword): the method used to calculate inverse document frequency. Default is Base (use the name specified above)
    - stopword (keyword): if specified as "none", then do not remove any stopwords. Otherwise this should treat as a filename where stopwords are to be read. Default is using the standard English stopwords corpus in NLTK. You should assume any word inside the stopwords file is a stopword. Otherwise you should not assume any pre-defined format of the stopword file.
    - stemmer (keyword): the stemming function to be used. Default is to use the Porter stemmer (nltk.stem.porter)
    - ignorecase (keyword) if specified as "no", then do NOT ignore case. Default is ignore case
- Standard method in corpus reader class.
    - See Table1.3 in http://www.nltk.org/book/ch02.html. You just need to call the corresponding method for the corpus readers. If the method is not supported by the reader and an error is returned, you are not responsible.
- Methods specific to tf-idf:
    - tf_idf(): return a list of ALL tf-idf vector (each vector should be a list) for the corpus, ordered by the order where fields are returned (the dimensions of the vector can be arbitrary, but need to be consistent with all vectors)
    - tf_idf(fileid=fileid): return the tf-idf vector corresponding to that file
    - tf_idf(filelist=[fileid]): return a list of vectors, corresponding to the tf-idf to the list of fileid input
    - tf_idf_dim(): return the list of the words corresponding to each vector of the tf-idf vector
    - tf_idf_new([words]): the input should be a list of words (treated as a document). The function should return a vector corresponding to the tf_idf vector for the new document (with the same stopword, stemming, ignorecase treatment applied, if necessary). You should use the idf for the original corpus to calculate the result (i.e. do not treat the document as a part of the original corpus)
    - cosine_sim(fileid=[fileid1, fileid2]): return the cosine similarity between two documents in the corpus
    - cosine_sim_new([words], fileid): the [words] is a list of words as is in the parameter of tf_idf_new() method. The fileid is the document in the corpus. The function return the cosine similarity between fileid and the new document specify by the [words] list. (Once again, use the idf of the original corpus).

You should also write a separate python program that load your class, and use it to calculate and display the td-idf (with default parameters) for the following corpus: ***brown, Shakespeare texts, state of the union.*** You should also output the cosine similarity between every pair of documents.

Your program output should be the following format.

For each corpus

First line: Name of the corpus

Second line: first fifteen word of your tf_idf_dim() output

Next line(s): <fileid>, <the fifteen tf-idf value for the vector corresponds to the fifteen dimensions above> (separted by blank space(s))

Next line(s): <fileid> <fileid>: cosine similarity between the pair of documents (you should also output the cosine similarity of a document with itself)

You should output the corpus in the order of Brown, Shakespeare text and state of the union

What to hand in

You should zip your python source code as a zip file and upload it to Canvas. Your python class should be appropriately commented:

- each function should have the description of its input/output (copy text from this handout is acceptable)
- while I do not expect you to comment every block/line, any blocks that do a major distinctive task (e.g. “calculate term frequency”) should be commented.

You will probably need your program for subsequent homework/projects.

Extra credit

I will write the program to test the running time of your class. Namely, the time for the constructor and the tf-idf() method. Bonuses will be given to the top 5 students will the fastest program.