Project 6b: Questions

Write an AI to answer questions.

```
$ python questions.py corpus
Query: What are the types of supervised learning?
Types of supervised learning algorithms include Active learning , classification and regression.

$ python questions.py corpus
Query: When was Python 3.0 released?
Python 3.0 was released on 3 December 2008.

$ python questions.py corpus
Query: How do neurons connect in a neural network?
Neurons of one layer connect only to neurons of the immediately preceding and immediately following layers.
```

When to Do It

By Saturday, December 31, 2022, 10:59 PM CST.

How to Get Help

1. Ask questions via Ed!
2. Ask questions via any of CS50's communities!

Background

Question Answering (QA) is a field within natural language processing focused on designing systems that can answer questions. Among the more famous question answering systems is Watson, the IBM computer that competed (and won) on *Jeopardy!*. A question answering system of Watson's accuracy requires enormous complexity and vast amounts of data, but in this problem, we'll design a very simple question answering system based on inverse document frequency.

Our question answering system will perform two tasks: document retrieval and passage retrieval. Our system will have access to a corpus of text documents. When presented with a query (a question in English asked by the user), document retrieval will first identify which document(s) are most relevant to the query. Once the top documents are found, the top document(s) will be subdivided into passages (in this case, sentences) so that the most relevant passage to the question can be determined.

How do we find the most relevant documents and passages? To find the most relevant documents, we'll use tf-idf to rank documents based both on term frequency for words in the query as well as inverse document frequency for words in the query. Once we've found the most relevant documents, there many possible metrics for scoring passages, but we'll use a combination of inverse document frequency and a query term density measure (described in the Specification).

More sophisticated question answering systems might employ other strategies (analyzing the type of question word used, looking for synonyms of query words, lemmatizing to handle different forms of the same word, etc.) but we'll leave those sorts of improvements as exercises for you to work on if you'd like to after you've completed this project!

Getting Started

- Download the distribution code from https://cdn.cs50.net/ai/2020/x/projects/6/questions.zip and unzip it.
- Inside of the `questions` directory, run `pip3 install -r requirements.txt` to install this project's dependency: `nltk` for natural language processing.

Understanding

First, take a look at the documents in `corpus`. Each is a text file containing the contents of a Wikipedia page. Our goal is to write an AI that can find sentences from these files that are relevant to a user's query. You are welcome and encouraged to add, remove, or modify files in the corpus if you'd like to experiment with answering queries based on a different corpus of documents. Just be sure each file in the corpus is a text file ending in `.txt`.

Now, take a look at `questions.py`. The global variable `FILE_MATCHES` specifies how many files should be matched for any given query. The global variable `SENTENCES_MATCHES` specifies how many sentences within those files should be matched

for any given query. By default, each of these values is 1: our AI will find the top sentence from the top matching document as the answer to our question. You are welcome and encouraged to experiment with changing these values.

In the `main` function, we first load the files from the corpus directory into memory (via the `load_files` function). Each of the files is then tokenized (via `tokenize`) into a list of words, which then allows us to compute inverse document frequency values for each of the words (via `compute_idfs`). The user is then prompted to enter a query. The `top_files` function identifies the files that are the best match for the query. From those files, sentences are extracted, and the `top_sentences` function identifies the sentences that are the best match for the query.

The `load_files`, `tokenize`, `compute_idfs`, `top_files`, and `top_sentences` functions are left to you!

Specification

> An automated tool assists the staff in enforcing the constraints in the below specification. Your submission will fail if any of these are not handled properly, if you import modules other than those explicitly allowed, or if you modify functions other than as permitted.

Complete the implementation of `load_files`, `tokenize`, `compute_idfs`, `top_files`, and `top_sentences` in `questions.py`.

- The `load_files` function should accept the name of a `directory` and return a dictionary mapping the filename of each `.txt` file inside that directory to the file's contents as a string.
  - Your function should be platform-independent: that is to say, it should work regardless of operating system. Note that on macOS, the `/` character is used to separate path components, while the `\` character is used on Windows. Use `os.sep` and `os.path.join` as needed instead of using your platform's specific separator character.
  - In the returned dictionary, there should be one key named for each `.txt` file in the directory. The value associated with that key should be a string (the result of `read`ing the corresonding file).
  - Each key should be just the filename, without including the directory name. For example, if the directory is called `corpus` and contains files `a.txt` and `b.txt`, the keys should be `a.txt` and `b.txt` and not `corpus/a.txt` and `corpus/b.txt`.
- The `tokenize` function should accept a `document` (a string) as input, and return a list of all of the words in that document, in order and lowercased.
  - You should use `nltk`'s `word_tokenize` function to perform tokenization.
  - All words in the returned list should be lowercased.
  - Filter out punctuation and stopwords (common words that are unlikely to be useful for querying). Punctuation is defined as any character in `string.punctuation` (after you `import string`). Stopwords are defined as any word in `nltk.corpus.stopwords.words("english")`.
  - If a word appears multiple times in the `document`, it should also appear multiple times in the returned list (unless it was filtered out).
- The `compute_idfs` function should accept a dictionary of `documents` and return a new dictionary mapping words to their IDF (inverse document frequency) values.
  - Assume that `documents` will be a dictionary mapping names of documents to a list of words in that document.
  - The returned dictionary should map every word that appears in at least one of the documents to its inverse document frequency value.
  - Recall that the inverse document frequency of a word is defined by taking the natural logarithm of the number of documents divided by the number of documents in which the word appears.
- The `top_files` function should, given a `query` (a set of words), `files` (a dictionary mapping names of files to a list of their words), and `idfs` (a dictionary mapping words to their IDF values), return a list of the filenames of the the `n` top files that match the query, ranked according to tf-idf.
  - The returned list of filenames should be of length `n` and should be ordered with the best match first.
  - Files should be ranked according to the sum of tf-idf values for any word in the query that also appears in the file. Words in the query that do not appear in the file should not contribute to the file's score.
  - Recall that tf-idf for a term is computed by multiplying the number of times the term appears in the document by the IDF value for that term.
  - You may assume that `n` will not be greater than the total number of files.
- The `top_sentences` function should, given a `query` (a set of words), `sentences` (a dictionary mapping sentences to a list of their words), and `idfs` (a dictionary mapping words to their IDF values), return a list of the `n` top sentences that match the query, ranked according to IDF.
  - The returned list of sentences should be of length `n` and should be ordered with the best match first.
  - Sentences should be ranked according to "matching word measure": namely, the sum of IDF values for any word in the query that also appears in the sentence. Note that term frequency should not be taken into account here, only inverse document frequency.

- If two sentences have the same value according to the matching word measure, then sentences with a higher "query term density" should be preferred. Query term density is defined as the proportion of words in the sentence that are also words in the query. For example, if a sentence has 10 words, 3 of which are in the query, then the sentence's query term density is `0.3`.
- You may assume that `n` will not be greater than the total number of sentences.

You should not modify anything else in `questions.py` other than the functions the specification calls for you to implement, though you may write additional functions, add new global constant variables, and/or import other Python standard library modules.

Hints

- In the `compute_idfs` function, recall that the `documents` input will be represented as a dictionary mapping document names to a list of words in each of those documents. The document names themselves are irrelevant to the calculation of IDF values. That is to say, changing any or all of the document names should not change the IDF values that are computed.
- Different sources may use different formulas to calculate term frequency and inverse document frequency than the ones described in lecture and given in this specification. Be sure that the formulas you implement are the ones described in this specification.

How to Submit

You may not have your code in your `ai50/projects/2020/x/questions` branch nested within any further subdirectories (such as a subdirectory called `questions` or `project6b`). That is to say, if the staff attempts to access `https://github.com/me50/USERNAME/blob/ai50/projects/2020/x/questions/questions.py`, where `USERNAME` is your GitHub username, that is exactly where your file should live. If your file is not at that location when the staff attempts to grade, your submission will fail.

1. Visit this link, log in with your GitHub account, and click **Authorize cs50**. Then, check the box indicating that you'd like to grant course staff access to your submissions, and click **Join course**.
2. Install Git and, optionally, install `submit50`.
3. If you've installed `submit50`, execute

   ```
   submit50 ai50/projects/2020/x/questions
   ```

   Otherwise, using Git, push your work to `https://github.com/me50/USERNAME.git`, where `USERNAME` is your GitHub username, on a branch called `ai50/projects/2020/x/questions`.
4. Submit this form.

You can then go to https://cs50.me/cs50ai to view your current progress!