CPSC 67 Lab #3: Vector Space Retrieval

Due Wednesday, Feb. 11 (11:59pm)

The goal of this lab is to perform vector space retrieval on the document collection you built in Labs #1 and #2. You will support bag of word queries and you will need to evaluate the performance of your queries using a number of evaluation metrics.

Fortunately, you will not need to make any updates to previous code in order to complete this lab. However, you will need to use the database and the files you have already downloaded. You will not download any additional artist data in this lab.

1 Content-Type

Not all of the web pages you downloaded are HTML pages, though up to now you have assumed that they have been. Some of the pages may be PDF files, Word documents, etc. In this lab, you would like to exclude any documents from your searches unless they are HTML pages. Fortunately, the header information that you have been caching (in the `raw/header/` directory) contains the information about the "Content-Type" of the page. Before you begin, you should parse each cached header file and update the CacheURL table to include the doc_type. Sample code to do this for you is provided on the wiki. Assuming your file structure and database are setup the way they should be, the code should work out of the box. Be sure to backup your database and read through the code before running it, so you understand what it does and to verify that it will work with your setup.

Though you don't need to make any changes to your previous code, if you'd to like to anticipate a change you'll need to make in Lab #4, you can go and fix your Lab #2 code so that when a page is downloaded the doc_type of the page is automatically filled into the CacheURL table.

2 Vector Space Queries

You will implement a vector space retrieval model to enable you to search through the cache of clean `text/html` pages. See Section 6.2 through 6.4.3 for further details on vector space retrieval.

Figure 6.15 (page 188) of the text book introduces the SMART notation for tf-idf variants. The *ddd.qqq* format is used to specify the weighting for the document vectors (*ddd*) and the weighting for the query vector (*qqq*). You will need to support four variants: *nnc.nnc*, *ntc.ntc*, *ltc.ltc*, and *lnc.ltc*.

You will need a way of evaluating the performance of the queries you make. By using the artist name as the search query, you can map the name of the artist directly to the pages that you have cached.[1] Then, for each artist, you can evaluate the performance of your query by comparing it against the set of documents you expected to have retrieved. For each artist search, you will need to compute the Average Precision, the R-Precision, the 10-Precision, and the AUC (area under the ROC curve). See Section 8.4 for further details on evaluation of ranked retrieval results.

Using each variant, and performing 40 artist queries (for each of the artists you have already downloaded), you will calculate the Mean Average Precision, Mean R-Precision, Mean 10-Precision, and Mean AUC. As an example, to calculate Mean Average Precision, you will calculate the Average Precision for each of the 40 artist queries and then take the mean of those 40 scores.

[1] You will need to use the JOIN operator to form a query that will allow you to map artist names to cached pages.

3 API

You will implement two classes with the following minimum API:

```
class Vector(object):
    """
    A Vector represents the contents of one document or query.
    """

    def __init__(self, wordlist):
        """
        Create a Vector based on a list of words.  The list may contain duplicate words.
        """

class VectorCollection(object):
    """
    A collection of Vectors representing many documents
    """

    def __init__(self):
        """
        Create an empty VectorCollection.
        """

    def addVector(self, vector, cache_id):
        """
        Add a vector derived from a document id to the collection of vectors.
        """

    def queryCollection(self, query, ddd, qqq):
        """
        Given a string query, perform a vector space comparison against all of the documents
        in the collection using tf-idf variant ddd.qqq.

        Returns a sorted list of (cache_id, score) tuples.  The list should be sorted so that
        the highest scoring document is first in the list.
        """

    def compareDocs(self, cache_id1, cache_id2, ddd):
        """
        Compare two documents in the VectorCollection.   The return value has the same
        format as queryCollection().
        """
```

You will actually not use the `compareDocs()` method at all in this lab; however, you will need it in a future lab and knowing that you'll eventually have to write that should help you modularize both the `compareDocs()` and `queryCollection()` methods since they are largely the same.

You will also need to implement four evaluation functions. Each function takes a list of `True` and `False` values which correspond to whether the document should have been returned (`True`) or not (`False`). For example, the list `[True, False, True]` would indicate that the first and third documents were correctly retrieved, but that the second should not have been retrieved. Given this list of boolean values, shown as the `graded` parameter in each of the functions below, you will implement the following:

- `avgPrecision(graded)` - computes the average precision
- `rPrecision(graded)` - computes the R-Precision
- `tenPrecision(graded)` - computes the 10-Precision, or "Precision at 10"

- `areaUnderCurve(graded)` - computes the area under the ROC curve

4 Results

On the wiki, you will report the following:

1. Report the MAP, Mean R-Precision, Mean 10-Precision, and Mean AUC using TF-IDF (ntc.ntc)
2. Report the Mean AUC using each of the four variants (nnc.nnc, ntc.ntc, ltc.ltc, lnc.ltc)
3. For each of these three queries, report the artist whose page is returned first using TF-IDF: 'alternative rock', 'classic rock', 'acoustic guitar'. (Though it is possible to do so, there is no need to report an evaluation of the two genre searches, but you may find it informative to do so. It is fairly straightforward to do this once you've implemented the code to do items 1 and 2 above.)

Report your results on the wiki.