LING 165 Lab 7: Document Retrieval

Synopsis

Implment a simple document retrieval system using the ideas of vector space models and singular value decomposition (SVD).

Task

Write a program that finds documents in a collection relevant to a user's query assuming the following:

(1) vocab = `/home/ling165/lab7/vocab`

(2) query = `oil industry`

(3) collection = `/home/ling165/lab7/wsj100.zip`

(4) document = a file in the collection

(5) Both query and documents are first represented as vectors of term frequencies but later reduced to vectors of 50 numbers using SVD.

(6) relevant documents = top five documents with high cosine similarity scores compared to the query using the reduced vectors

Your program should run as follows:

```
[hahnkoo@gray lab7]$ python search.py
WSJ_2325
WSJ_0437
WSJ_0870
WSJ_1470
WSJ_1766
```

Writing this program requires you to use the `numpy` library. See the section on implementation notes below.

Email me your code, or where I can find it, when you're done.

Data

I'm providing two files for this assignment: `vocab` and `wsj100.zip`. They are in `/home/ling165/lab7/` on the gray server.

`vocab` lists words, one word per line. This is what you should use to represent queries and documents as vectors.

`wsj100.zip` is a zip file containing 100 files from the Wall Street Journal corpus. Unzipping it will create a `wsj100` directory containing the 100 files. This is a subset of the WSJ files you used for Lab 6. But as a reminder, each file in the directory has just one line consisting of words that are separated from each other by white space. For example, `WSJ_2325` looks like this:

```
oil industry middling profits persist rest year major oil companies next few days ...
```

I've already pre-processed the text for you. Do not process it any further.

FYI, the original file from the Wall Street Journal corpus looked like this:

```
.START

The oil industry's middling profits could persist through the rest of the year.

Major oil companies in the next few days are expected to report much less robust
 earnings than they did for the third quarter a year ago, largely reflecting det
eriorating chemical prices and gasoline profitability.
...
```

Basically, I extracted nouns, verbs, adjectives, and adverbs from the original, lower-cased them, and dumped them into a single line. If you're curious, see the originals under `/data/TREEBANK/RAW/WSJ/` on the gray server.

Implementation Notes

`numpy`

You need to use `numpy`. It is a very useful Python package for doing maths, especially matrix operations. The package is already installed on the gray server. But you'll have to install it on your own computer. See the instructions at `http://www.scipy.org/install.html`. It explains how to install `numpy` and `scipy`, which is another useful package for math and science.

Once `numpy` is installed, import it, i.e. `import numpy`.

To create a numpy matrix object, you apply `numpy.matrix` to a list of lists of numbers, one list per row in the matrix. For example, the following will create a matrix `z` which has `x` and `y` as rows:

```
x = [2, 0, 3, 1, 0]
y = [1, 0, 1, 0, 0]
z = numpy.matrix([x, y])
```

Once you have a matrix, you can use attributes `T` and `I` to get its transpose and its inverse. For example, `z.T` for transpose of `z` and `z.I` for inverse of `z`.

You can use the `*` operator for matrix multiplication. For example, `A*B` to multiply matrix `A` by matrix `B`.

There are other useful functions to work with matrices. I'll explain more as I suggest how to write your program below.

Term-document matrix

You'll first have to create a term-document matrix with term-frequency weighted scores. To that end,

(1) List words in the vocabulary file.

(2) For each document in the collection, create a list of numbers, where each number specifies how often a word in the vocabulary appears in the document.

(3) Create a list containing the lists from (2). Call it `C` (for collection).

(4) Convert `C` to a numpy matrix object, i.e. `C = numpy.matrix(C)`.

(5) Transpose `C`, i.e. `C = C.T`.

Singular value decomposition

You want to get a lower-rank approximation of the term-document matrix. Use the following equation where k (= 50 for this assignment) is the rank you want the matrix to be reduced to:

$$C = U\Sigma V^T \approx U_k \Sigma_k V_k^T \tag{1}$$

To that end,

(1) Run `numpy.linalg.svd(C, full_matrices=False)`. This should return the following three objects:

  (a) a numpy matrix corresponding to U in equation (1); call it `U`.

  (b) a numpy array consisting of the diagonals in Σ in equation (1); call it `s`.

  (c) a numpy matrix corresponding to V^T in equation (1); call it `VT`.

(2) Take the first k numbers in `s` and create a diagonal matrix, i.e.

  (a) `s_k = s[:k]`

  (b) `S_k = numpy.matrix(numpy.diag(s_k))`

(3) Take the first k columns of `U`, i.e. `U_k = U[:, :k]`.

(4) Take the first k rows of `VT`, i.e. `VT_k = VT[:k, :]`.

The resulting `U_k`, `S_k`, `VT_k` respectively correspond to U_k, Σ_k, V_k^T in equation (1).

Query vector in the reduced space

You want to represent the query as a vector in the reduced space. Use the following equation:

$$q_k = \Sigma_k^{-1} U_k^T q \tag{2}$$

where q is the query vector in the original space and q_k is the corresponding vector in the reduced space. To that end,

(1) Think of the query – `oil industry` in this case – as a document of just two words. Create a list of numbers for the query just like you did for the documents. Call it `q`.

(2) Change that list into a numpy matrix that has only one row, which is `q`. That is, `q = numpy.matrix([q])`.

(3) Transpose the matrix, i.e. `q = q.T`.

(4) Apply equation (2), i.e. `q_k = S_k.I * U_k.T * q`.

Document vectors in the reduced space

You want to represent the documents as vectors in the reduced space. Use the following equation:

$$D_k = \Sigma_k V_k^T \tag{3}$$

The resulting matrix D_k consists of k rows and a column for each document in the collection. How do you implement this? Just do `D_k = S_k * VT_k`.

Calculating similarity between the query and the documents

You want to calculate the cosine similarity between the query vector and each document vector in the reduced space. Use the following formula where D_{ki} is the ith column in D_k:

$$\cos(q_k, D_{ki}) = \frac{q_k \cdot D_{ki}}{|q_k| \times |D_{ki}|} \tag{4}$$

The numerator is the dot product of q_k and D_{ki}. Here's how to calculate it using `numpy`:

```
float(q_k.T * D_k[:, i])
```

The denominator is the product of the norms (lengths) of the two vectors. Here's how to calculate it using `numpy`:

```
numpy.linalg.norm(q_k) * numpy.linalg.norm(D_k[:, i])
```