Part A - LSA Search *[40 marks]*

The goal of the first part of the assignment is to try and re-create the results given by the LSA example in class. Code has been provided for you that does most of the work but the mathematical component needs to be filled out by you in order to make it functional.

First, a bit of recap.

Recall that with LSA we first generate a $n \times m$ term by document matrix, A, where:

$$\begin{aligned} n &= \text{Number of terms} \\ m &= \text{Number of documents} \\ A_{i,j} &= \text{Number of times term } i \text{ occurs in document } j \end{aligned}$$

After the A matrix has been generated the next step is to perform *Singular Value Decomposition* (SVD) on the A matrix. This factors the matrix into three components thusly:

$$A = TSD^{\mathrm{T}} \tag{1}$$

where:

$$T = n \times r \text{ matrix where } r \text{ is the rank of } A. \tag{2}$$

$$S = r \times r \text{ diagonal matrix of singular values.} \tag{3}$$

$$D^{\mathrm{T}} = r \times m \text{ matrix where } r \text{ is the rank of } A. \tag{4}$$

Roughly speaking this *semantic space* we have created has the features such that the rows of T correspond to the *terms* in the semantic space and the columns of D^{T} correspond to the *documents*. Typically we perform a dimensional reduction stage where we select a value, say k, and restrict ourselves to the first k values of S as well as the columns of T and rows of D^{T} with the hope that this draws out the underlying latent semantic features of our text to make semantic comparisons between the two more meaningful.

One feature of LSA is using it as a form of *information retrieval* or *search* with the hopes of avoiding the *vocabulary gap problem*. A *query* is first selected and processed and transformed in a document in the sense of how it would look as an additional column in the A matrix. Obviously this would be restricted to the vocabulary of the A matrix to begin with (that is, only the terms represented in the rows of A could be represented in this new psedo-document).

What you will need to do is to transform the query into the semantic space by *folding it in* to its linear space.

NOTE, in the files given to you, `partA.py` contains the driver where it indicates the methods that need to be filled out by you. All of the methods that need to be completed are inside of the `semantic_space.py` source file. These can be found easily via a search for comments containing **TODO** in the source files of interest. The components to be filled out are clearly indicated thusly.

[20 marks] (a) In order to *fold* a document into the semantic space (make it an additional "column" of D^{T} in essence) you will need to complete the `create_query_vector()` method in `semantic_space.py` by creating the additional document in the A matrix (column) form the query passed (the query has been processed for you already in this method so the terms are provided to you in a list).

Once the query vector, $\vec{q}$, has been formulated the `fold_in_query` method will need to be completed in order to transform it into the semantic space properly (that is, in the same vector space as D^{T}). The formulae for this is:

$$\text{New Query Vector} \quad = \quad \vec{q}^{\mathrm{T}} T S^{-1}$$

Don't worry about the dimensional reduction at this point, calculate this as if you were using the full T and S^{-1} matricies[1].

[8 Marks creation of $\vec{q}$, 12 Marks transformation into semantic space].

[10 marks] (b) Once the query has been transformed into the SVD vector space a comparison operator needs to be created. The cosine operator is a popular comparison operators in vector space models to ascertain semantic similarity between two vectors (typically documents). In order to achieve this you will need to complete the `cosine_with_doc` method in the `semantic_space.py` file. This method takes both the vector created in part (a) above as well as an index into the D^{T} matrix (column/document) to calculate the similarity score. For reference, the formula for this is (where, in this context, $\vec{q}$ is the output of part (a) above) where D_i^{T} represents the i^{th} column of the matrix:

$$cos(\theta) \quad = \quad \frac{\vec{q} \cdot D_i^{\mathrm{T}}}{\|\vec{q}\| \|D_i^{\mathrm{T}}\|}$$

Remember, before you compute the similarity scores between the query vector and the appropriate column in the D^{T} matrix both vectors must be *scaled* by S first.

Note that at this point we would take into consideration our dimensional reduction value. This can be found in the constructor as the `max_dimension` value (set to 2) for the object contained in the `semantic_space.py` file. In practice what dimension reduction means is only using the first `max_dimension` columns in T, values in S (and its variants), and rows in D^{T}. Be sure to use this value to calculate your values using the correct dimensionality (which has been set for you in the `max_dimension` value)!

[10 marks] (c) The formula above can also be used for terms in the T matrix from the SVD computation. You will need to complete the `cosine_with_term` method in the `semantic_space.py` file to complete this. This method takes two parameters which are indicies (rows) of the T matrix which represent *terms*. From the resulting cosine calculation (similar to the above) we can see the similarity of terms between one another in our semantic space (this enables us to see terms that are viewed as semantically similar in our semantic space). The code supplied tries to to calculate the similarity between each pair of terms. Complete the above method to see the results! Note, be sure to use the `max_dimension` value

[1] Note, in practice we represent S and S^{-1} as vectors as they are diagonal matrices

as outlined above for this code. Don't forget to scale the vectors by S before making the computation.

Part B - Term Weightings [*20 marks*]

Consider the matrix shown below. This is the term by document matrix for the dataset we have been using thus far. As you can see each entry in the matrix, A_{ij} corresponds to the number of times term i appears in document j. For example the term "eps" occurs in document "c3" exactly once.

	c1	c2	c3	c4	c5	m1	m2	m3	m4
computer	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
eps	0.00	0.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00
graph	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00
human	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00
interface	1.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
minors	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00
response	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
survey	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
system	0.00	1.00	1.00	2.00	0.00	0.00	0.00	0.00	0.00
time	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
trees	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00	0.00
user	0.00	1.00	1.00	0.00	1.00	0.00	0.00	0.00	0.00

The matrix is considered to be using a *term frequency* model (TF). That is, it only records the number of times a term occurs in a document. This can lead to problems with larger datasets as it is possible that important terms don't occur too many times and get drowned out in the noise created or, alternatively, there are very popular terms that may tend to dominate any calculations made.

There are two weighting schemes we will examine that can be used as alternatives to try and minimise this problem. Typically the term by document matrix is first constructed then a weighting scheme is applied to it before calculating the SVD. The two schemes we are going to employ with this matrix are called:

- Term Frequency - Inverse Document Frequency (TF-IDF)
- Log-Entropy (LE)

[10 marks] (a) **TF-IDF**

For the matrix above apply the TF-IDF weighting scheme to it. Each element in the matrix should have the following formula applied to it:

$$A_{ij} = f_{ij} \log \left(\frac{n}{\sum_j \chi(f_{ij})} \right)$$

where

$$\begin{aligned} f_{ij} &= \text{The term frequency of term } i \text{ in document } j \\ \chi(v) &= 1 \text{ if } v > 0, \text{ 0 otherwise (binary indicator)} \\ n &= \text{Total number of documents} \end{aligned}$$

NOTE: for the sake of consistency please use the *natural* logarithm function in your calculations.

[10 marks] (b) **Log-Entropy**

For the matrix above apply the LE weighting scheme to it. Each element in the matrix should have the following formula applied to it:

$$A_{ij} = \log(1 + f_{ij}) \left[1 + \left(\sum_j \frac{p_{ij} \log(p_{ij})}{\log(n)} \right) \right]$$

where

f_{ij} = The term frequency of term i in document j

$p_{ij} = \frac{f_{ij}}{\sum_j f_{ij}}$
Proportion of term i in document j with respect to the total frequency count of f_{ij}

n Total number of documents

NOTE: for the sake of consistency please use the *natural* logarithm function in your calculations.