Text Processing and Search with R

Introduction:

Given a corpus of Amazon food reviews, Use R programming language to process the data and build a simple command line search engine. When given a query, it should retrieve top 25 documents relevant to the query.

Objectives:

- Learn basics of text processing in R
- Indexing Documents (TF-IDF)
- Plotting in R

Task:

1. Perform stemming on the given data set.
   Stemming is mapping different forms of a word to a single form. An e.g. is, running and ran to run. A library called “snowball” is available in R to perform the stemming, use it. Learning how to use a library in R is a part of the task.
2. Stopword removal.
   Stopwords are words like: “the”, “of”, “in” etc. They occur nearly in every document and their occurrence convey no information about the document. A list of all the stopwords can be obtained by one simple function call in R. Find the list and remove all these words from all the documents.
3. Construct Vector space model of the given corpus. The words corpus and vector space model are described below one by one. The construction is also explained in the following sections.
4. Construct an Rscript, that when given a query as input, retrieves the relevant documents.
5. For each query find 25 most relevant documents for that query. Use cosine similarity to get relevance of query to a document. Cosine similarity is explained in the following sections.

6. Find the cosine similarity between all the possible pairs of documents, and visualise the similarity result as heatmap

Corpus:

A collection of documents is called a **corpus**. Our data set is a collection of reviews of some food items. We consider each review as a document. Thus, the collection of reviews is our corpus. Each document (review) contains all or some of the following entries:

- product/productId: B0009XLVG0
- review/userId: A2725IB4YY9JEB
- review/profileName: A Poeng "SparkyGoHome"
- review/helpfulness: 4/4
- review/score: 5.0
- review/time: 1282867200
- review/summary: My cats LOVE this "diet" food better than their regular food
- review/text: One of my boys needed to lose some weight and the other didn't. I put this food on the floor for the chubby guy, and the protein-rich, no by-product food up higher where only my skinny boy can jump. The higher food sits going stale. They both really go for this food. And my chubby boy has been losing about an ounce a week.

  For the assignment consider only **summary** and **text**, ignore other entries

Searching:

An important factor which contributes to search time is how the objects (documents in our case) are stored or organised. An example of how the storage method affects the search time can be: sorted vs unsorted integers in an array, i.e. binary vs linear search.

Over data organized in a particular fashion indexing can used for easy and fast searching, as in the case of index of a book. We shall also index corpus, since we are building a **search** engine, though small.

Vocabulary:

The collection all the words in the corpus without duplications is called **Vocabulary**. Thus, we can assign a unique integer to every word in a Vocabulary by listing all the words in some order. This integer-word binding is important because we will refer to words by their respective integers.

Example on Vocabulary and numbering of words:

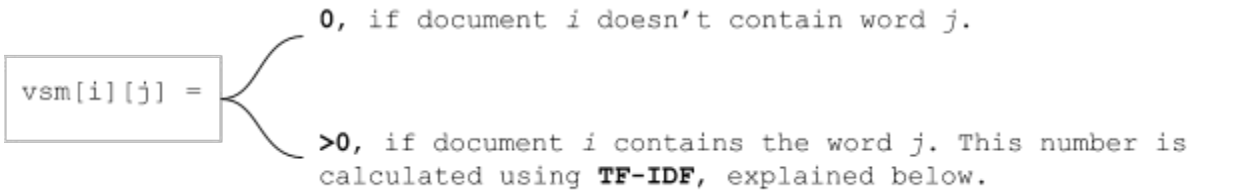
Say the corpus consists of two documents **X.txt** and **Y.txt**. X.txt contains only one word “Computing” and Y.txt contains “Computing Lab One”. The Vocabulary for this corpus consists of three words: “Computing”, “Lab” and “One”. Now we list the Vocabulary in some arbitrary order, say “Lab”, “Computing” and “One”. The order in which the words are considered is not relevant for the vocabulary. We call such a model as bag of words model.

Vector space model:

It is a method for representing documents as vectors. The number of vectors is number of documents in the corpus. Thus, one vector for each document. This implies that the documents should also be numbered in some order, providing a unique integer for each document, like in case of the Vocabulary.

Each vector has size equal to the number of words in the Vocabulary. Thus for each word there is an entry in the vector. This entry is a real number greater than or equal to zero. The entry is zero if the corresponding document doesn’t contain the corresponding word, non zero otherwise. “What exactly this non zero value is?”, this value is **TF-IDF**, explained after this section. Below is a pictorial description of the vector space model. Let’s call our vector space model to be **vsm**.

```
positive_real vsm [Number of documents] [Number of unique words].
```



TF-IDF:

It is a value used to determine importance of a word w.r.t a document in a corpus. It is a value calculated from **TF** and **IDF**. These terms are defined below one by one.

TF:

Term frequency. It **reflects** how many times a term occurred in a document. Thus, tf exists for each combination of documents and words. It is a function of both document and word, written as: $tf(d, w)$.

Let $f(d, w)$ be frequency of a word, i.e. total number occurrences of a word *w* in a document *d*.

For simplicity use $tf(d, w) = f(d, w)$. The function value $f(d, w)$ can be operated before being assigned to *tf (d, w)*.

tf can be viewed as a 2D matrix as below.

	Word 0	Word 1	....	Word j	...
Document 0					
Document 1					
...					
Document i				$tf(i, j)$	
...					

IDF:

Inverse document frequency. Is a measure of how much information a word provides, i.e. whether the word is common or rare across all documents. It is calculated for each word, hence $idf(w)$ is a positive real number for word w.

idf is calculated as follows:

$$idf(w) = \left(\frac{Total\ number\ of\ documents}{(1 + Number\ of\ documents\ "w"\ appears\ in)} \right)$$

idf can be viewed as a linear array of size number of words in the Vocabulary.

	Word 0	Word 1	…	Word i	...	Word n
idf values				$idf\,(i)$		

tf-idf calculation:

It is a value computed by combining *tf* and *idf* using some mathematical operator. tf-idf of word *w* in document *d*, $tf-idf\,(w,\ d)$, is to be calculated as:

$$tf-idf\,(w,d)\ =\ log\,(tf\,(w,\ d)\ *\ idf\,(w))$$

Since *tf* exists for each document and word, *tf-idf* also exists for each document and word. *tf-idf* values can be viewed as a 2d matrix just like that for *tf*.

Vector space model OR tf-idf values matrix:					
	Word 0	Word 1	....	Word j	...
Document 0					
Document 1					
...					
Document i				$tf-idf(i,\ j)$	
...					

This tf-idf values matrix is nothing but the vector space model.

Querying:

For a query construct the tf-idf vector. Use it to find the cosine similarity between the given query and a document as stated under title “Cosine similarity”. Construct an array of cosine similarities of the query with all the documents. Sort the array in decreasing order. Print names of the top 25 documents.

Cosine similarity:

Of multiple methods to find similarity between two vectors cosine similarity is one. Calculate the cosine similarity between two vectors say **A** and **B** as follows.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

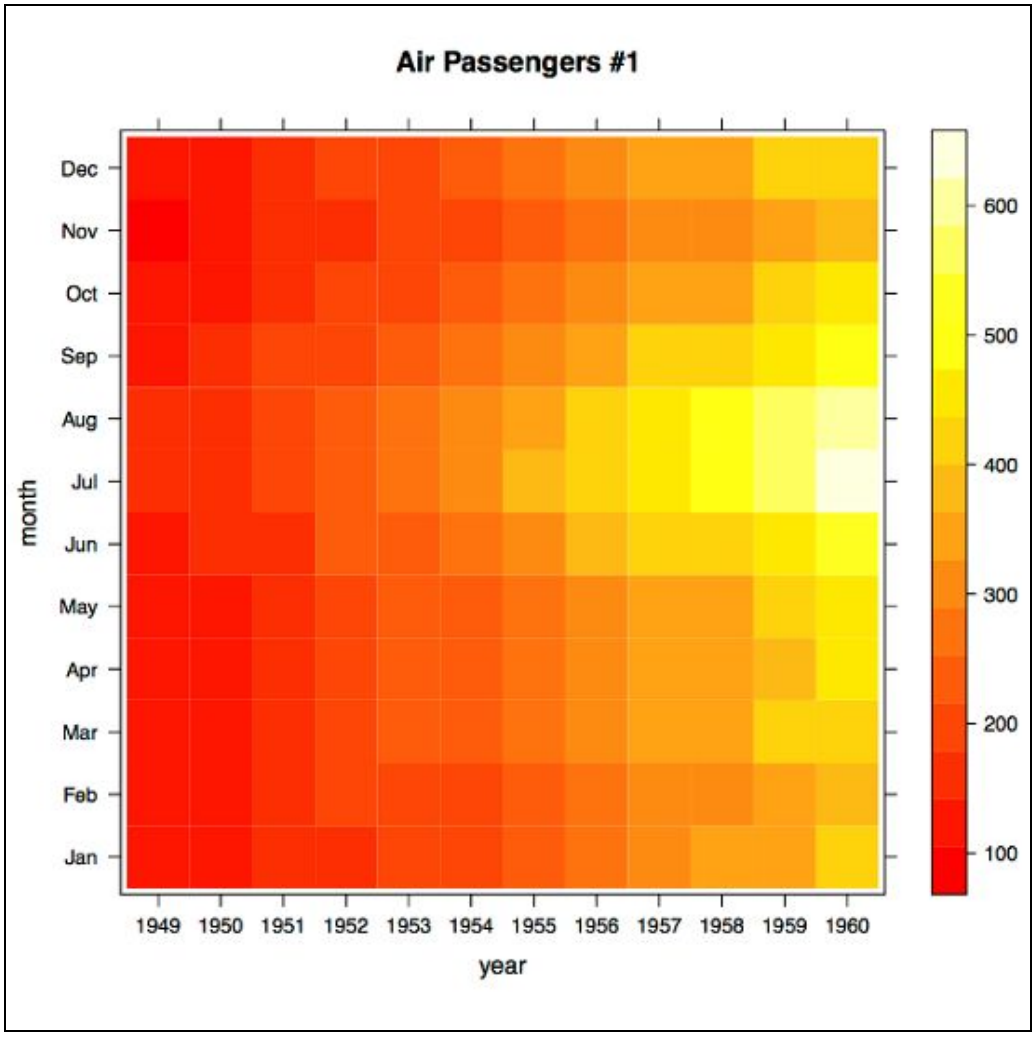
Heatmap:

It is a representation of data stored in a matrix to an image in which colors varying with the values of cells. Thus data stored in the 2d matrix called vector space model or tf-idf matrix can be printed as an image where the similarity between two documents will be shown difference in colours.

In the heatmap the x-axis should contain document names and y-axis the words. The color intensity shall be say max at +1 and least for -1 tf-idf value.

R provides a simple function call to plot heat map of a matrix. You just have to read the documentation of the function.

Example of a heat map:



Extensions:

It is often the case that user might search as a phrase, say “United States”, here the user is certainly not looking for a document containing a sentence like, “Finally in the movie 2 **states**, both the hero and heroine are **united**.” It is often the case that phrases represent more than what the individual words of the phrases do. To tackle such issues extend your model, such that the vocabulary will now contain every possible two word combinations that is appearing in the corpus. Such 2 word combinations are called bigrams.

For example, if the document is
“Finally in the movie 2 states, both the hero and heroine are united.”
The text after stop words removal and stemming will be
final movie 2 states, hero heroine united

All possible bigrams are:
final movie
movie 2
2 states
states hero
hero heroine
heroine united

In the VSM, just like the individual words, calculate the TF-IDF for every possible bigram along with the existing unique words (which are also called as unigrams). During the retrieval weigh the bigram tf-idfs with a greater weight than the unigrams.

Calculate the rank using both unigram and bigram vector space models. Merge the results using $\alpha * Rank_{unigram} + \beta * Rank_{bigram}$. α and β are the weights given to unigrams and bigrams respectively.

A set of queries will be provided. Also plot the count of set difference between bigram VSM and unigram VSM for each query results.

Deliverables:

Source code along with the plots. Also the search results for unigram and bigram VSM for the set of queries that will be provided.

Note:

You may use external libraries like tm and snowballC for steps 1 and 2. But for building the VSM you are not allowed to use tm library. Before using any external library (package), get prior permission from the TAs.