Assignment 1: Information Retrieval
Chin Zhe Jing 22221970

1. In class, we briefly discussed pre-processing techniques such as stemming, stop-word removal and thesaurus construction. Given a text document, suggest any three additional pre-processing techniques that may be used. Explain the approach and outline the potential benefit of the approach. (10 marks)

   The below 3 additional pre-processing techniques are suggested:

   1. Lower Casing is to convert all the words in the text to be lower case. As words are having the same meaning disregard the letter case, so in order to avoid words like: Damaged and damaged being categorised as 2 different words and increasing the dimensionality, we could apply lower casing to the text.

   2. Tokenization is to break the text into smaller units as they are more easily being understood by the machine hence helping the machine to understand the whole text.

   3. We should remove the punctuations in text as most of them are meaningless and removing them could remove some of the noise in the data. However, we should take note on the contraction words as removing the punctuation from them could possibly create noise for the model.

2. Given the following small sample document collection:
   a. D1: Shipment of gold damaged in a fire , length = 7 words
   b. D2: Delivery of silver arrived in a silver truck
   c. D3: Shipment of gold arrived in a truck

   Calculate the term weightings for terms in D1. Show your workings and state any assumptions you make. (10 marks)

   $w_{i,j} = f_{i,j} \times log_2(\frac{N}{N_i})$

   with,
   $f_{i,j}$ = Tf = occurrence of word / total words
   N = number of documents = 3
   N_i = occurrence of term in given set of documents

   Shipment : N_i= 2, $f_{i,j}$ = 1/7
   $w_{i,j}$ = 1/7 × $log_2 (\frac{3}{2})$ = 0.0836

   Of: N_i= 3, $f_{i,j}$ = 1/7
   $w_{i,j}$ = 1/7 × $log_2 (\frac{3}{3})$ = 0

   Gold: N_i= 2, $f_{i,j}$ = 1/7
   $w_{i,j}$ = 1/7 × $log_2 (\frac{3}{2})$ = 0.0836

Damaged: N_i= 1, $f_{i,j}$ = 1/7
$w_{i,j} = 1/7 \times log_2 \left(\frac{3}{1}\right) = 0.2264$

In: N_i= 3, $f_{i,j}$ = 1/7
$w_{i,j} = 1/7 \times log_2 \left(\frac{3}{3}\right) = 0$

A: N_i= 3, $f_{i,j}$ = 1/7
$w_{i,j} = 1/7 \times log_2 \left(\frac{3}{3}\right) = 0$

Fire: N_i= 1, $f_{i,j}$ = 1/7
$w_{i,j} = 1/7 \times \log\left(\frac{3}{1}\right) = 0.2264$

Based on the calculations, we can conclude that the term “Damaged” and “Fire” is more important in getting a more related results.

3. In class we discussed the document collection as term-document matrix, where each cell in the matrix indicates the usefulness of term i in describing document j. We also discussed how we could evaluate the similarity of a query and document.

   Outline a suitable indexing structure to store the information in the matrix (note that matrix is sparse). (10 marks)

   We could index the information in the matrix in key-value pairs (dictionary). For each term, keep only documents that consist that term and has weight > 0. Hence comparison between query and documents could be done by simply retrieving the term IDs.

Term - document	Term - query

{
T1: [
D1: 0.03,
D2: 0.1,
D3: 0.002,
D100: 0.833
],
T2: [
D2: 0.54
],
T3: [
D1: 0.67,
D100: 0.34
]
}
	{
	T1: 0.03,
	T2: 0.442,
	T3: 0.23
	}

Outline at a high level, in pseudo-code, an algorithm to calculate the similarity of a document to a query. (10 marks)

This algorithm calculate the similarity of a document to a query.

Assume: Terms extracted from query and there will be at least and more than 1 term

```
Similarity(Wiq, Wid, doc_id):
Input:  Wiq: key-value pairs of weight of term in query (refer to indexing structure
        above)
        Wid: key-value pairs of weight of term in all documents (refer to indexing
        structure above)
        doc_id: ID of the document

Declare:
Wiq_Wid = []
Wiq_square = []
Wid_square = []

Foreach key, value in Wiq:
        If key in Wid_keys:
                If value = 0 or Wid[key][doc_id] = 0
                        then continue to the next key, value
                Else:
                        Append value * Wid[key][doc_id] to Wiq_Wid
                        Append value ^2 to Wiq_square
                        Append Wid[key][doc_id] ^2 to Wid_ square

Return the similarity with sum(Wiq_Wid) / (sum(Wiq_square) * sum(Wid_square))
```