Exercise A

The given data was stored into an Excel spreadsheet, which calculated things like each term's *idf* (inverse document frequency). This spreadsheet is attached to this submission.

Exercise A Part a) – Vector Space Model Cosine Similarity Ranking

This ranking system models each query as an n-dimensional vector, where n is the number of terms in the query. For queries $Q_1 = "university\ Riverside"$ and $Q_2 = "California\ university"$, each term gets its own dimension. To clarify the calculation, the Excel sheet puts the queries into 3-dimensional vectors of the form $\{University, California, Riverside\}$, where each value is the term frequency of that term within the query. So, $Q_1 = \{1,0,1\}$, and $Q_2 = \{1,1,0\}$.

We then set up the document vector, of the same 3-dimensional form as the query vectors. Here, each value is that term's $tf \cdot idf$ value, where tf (term frequency) is the frequency of the term within the document, and where idf (inverse document frequency) is given by the following formula:

$$idf = \log_2 \frac{N}{df}$$

Here, N is the number of documents in the collection C, and df is the document frequency (the number of documents that term appears in).

Once the product $tf \cdot idf$ is calculated, we took the cosine of the angle between the document vector $\vec{D}$ and each query vector $\overrightarrow{Q_1}$ and $\overrightarrow{Q_2}$. This calculation used the definition of the dot product to calculate the cosine. For each query vector $\overrightarrow{Q_i}$,

$$\cos\theta_{D,Q_i} = \frac{\vec{D} \cdot \overrightarrow{Q_i}}{\|\vec{D}\|\|\overrightarrow{Q_i}\|}$$

I calculated $\cos\theta_{D,Q_1} \approx 0.695$ and $\cos\theta_{D,Q_2} \approx 0.964$. See the spreadsheet for the precise calculation, under the tab marked "A. a) tf-idf".

Exercise A Part b) – BM25 Ranking

In general, we use the following formula for a BM25 ranking between a document D and a query Q:

$$\text{BM25}(D,Q) = \sum_{i\in Q} \log_2 \frac{{r_i + 0.5}/{R - r_i + 0.5}}{{n_i - r_i + 0.5}/{N - n_i - R + r_i + 0.5}} \times \frac{(k_1+1)f_i}{K+f_i} \times \frac{(k_2+1)qf_i}{k_2+qf_i}$$

Here, R and r_i are relevance factors detailing how relevant a word is to a query. Since these weren't given, we use $R = r_i = 0$ and algebraically simplify our formula to the following:

$$\text{BM25}(D,Q) = \sum_{i\in Q} \log_2 \frac{N - n_i + 0.5}{n_i + 0.5} \times \frac{(k_1+1)f_i}{K+f_i} \times \frac{(k_2+1)qf_i}{k_2+qf_i}$$

We use the following definitions here:

- N: number of documents
- qf_i: query frequency of term i
- n_i: document frequency of term i (we previously called this term df_i)
- f_i: term frequency of term i (we previously called this term tf_i)
- We use the TREC values for K, k_1, k_2, and b:
    - $K = k_1\left[(1-b) + \left(b \times \frac{dl}{dl_{avg}}\right)\right]$
    - $k_1 = 1.2$
    - $k_2 = 100$
    - $b = 0.75$
    - dl: length of document D

- dl_{avg}: average document length in collection C
- Assuming that $dl = dl_{avg}$, the formula for K simplifies down to $K = k_1 = 1.2$.

Applying this data to our given D, Q_1, and Q_2 yielded $\text{BM25}(D, Q_1) \approx 6.544$ and $\text{BM25}(D, Q_2) \approx 7.306$. Again, see the spreadsheet for the calculations, under the tab labeled "A. b) BM25".

Exercise A Part c) – Unigram Language Model (No Smoothing)

In lieu of an intimidating formula, we simply search for a YouTube tutorial. It tells us that we can measure document-query relevance by taking the probability of finding each query term in the document. So, with our D, Q_1, and Q_2, we'll calculate $P(q_i in\ D)$, where $P(D, Q_1) = \prod_{i \in Q} \frac{df_i}{N_D}$, where df_i is the document frequency of term i, and where N_D is the total word count (non-distinct) in D.

Using this, $P(Q_1, D) \approx 5.917 \times 10^{-3}$ and $P(Q_2, D) \approx 1.775 \times 10^{-2}$. The data for this calculation can be found in the spreadsheet under the tab labeled "A. c) Uniform Language Model".

Exercise B Part 1) – SimHash

We implement the SimHash algorithm for the documents based on the slides (or this tutorial). When hashing the terms, one can choose to either mod the ASCII sum by 255 or 256. I wrote a program to hash the given documents D_1 through D_3, and defined D_0 to be the document about tropical fish in the slides, for fun. I then calculated the Hamming Distance between each document, both using $mod\ 255$ and $mod\ 256$. The findings are summarized in the table below.

The code for this exercise is attached to this submission. Stopwords were omitted.

Terms Hashed using $mod\ 256$					
Document	**SimHash**	**Hamming Distance to D_0**	**Hamming Distance to D_1**	**Hamming Distance to D_2**	**Hamming Distance to D_3**
D_0	10101100	0	4	4	4
D_1	00110101	4	0	4	0
D_2	00000000	4	4	0	4
D_3	00110101	4	0	4	0

Terms Hashed using $mod\ 255$					
Document	**SimHash**	**Hamming Distance to D_0**	**Hamming Distance to D_1**	**Hamming Distance to D_2**	**Hamming Distance to D_3**
D_0	10101111	0	6	7	5
D_1	00010001	6	0	1	1
D_2	00010000	7	1	0	2
D_3	00110001	5	1	2	0

Exercise B Part 2) – SimHash Trade-offs

One might use shorter signatures to save space. At Internet-level scales, the difference between an 8-bit signature and 64-bit signature is significant.

However, longer signatures have the benefit of storing more "information" about a document's contents, which allows for more precise Hamming Distance similarity calculations.

Exercise B Part 3) – SimHash Ideal Bit Threshold

The $mod\ 256$ results were unexpected, but going off of the $mod\ 255$ results, I'd say a Hamming Distance of 1 or less is close enough to call two documents near-identical.

Documentation

The code and spreadsheets are attached below, and are also attached to this submission.

Term	Term Frequency (tf)	Document Frequency (df)	Inverse Document Frequency (idf)	TF-IDF Weight (tf * idf)	N (number of documents in the collection)	Document Vector {University, California, Riverside}	Q_1 Vector "university Riverside" {University, California, Riverside}	Q_2 Vector "California university" {University, California, Riverside}	Q_1 Cosine Similarity	Q_2 Cosine Similarity
university	4	200	2.322	9.288	1000	9.288	1	1	0.695	0.964
California	3	150	2.737	8.211		8.211	0	1		
Riverside	1	100	3.322	3.322		3.322	1	0		
one	2	100	3.322	6.644						
10	1	100	3.322	3.322						
within	1	100	3.322	3.322						
prestigious	1	100	3.322	3.322						
system	1	100	3.322	3.322						
only	1	100	3.322	3.322						
UC	1	100	3.322	3.322						
locate	1	100	3.322	3.322						
inland	1	100	3.322	3.322						
southern	1	100	3.322	3.322						
widely	1	100	3.322	3.322						
recognize	1	100	3.322	3.322						
most	1	100	3.322	3.322						
ethinic	1	100	3.322	3.322						
diverse	1	100	3.322	3.322						
research	1	100	3.322	3.322						
nation	1	100	3.322	3.322						

Term	Term Frequency (f_i)	Document Frequency (n_i)	Number of documents in the collection (N)	Q_1 "university Riverside" {University, California, Riverside} $(qf_i)_1$	Q_2 "California university" {University, California, Riverside} $(qf_i)_2$	k_1	k_2	b	dl	dl_{avg}	K	$BM25_i(D,Q_1)$
university	4	200	1000	1	1	1.2	100	0.75	26	26	1.2	3.38
California	3	150		0	1							0.00
Riverside	1	100		1	0							3.16
one	2	100										
10	1	100										
within	1	100										
prestigious	1	100										
system	1	100										
only	1	100										
UC	1	100										
locate	1	100										
inland	1	100										
southern	1	100										
widely	1	100										
recognize	1	100										
most	1	100										
ethinic	1	100										
diverse	1	100										
research	1	100										
nation	1	100										

$BM25_i(D,Q_2)$	$BM25(D,Q_1)$	$BM25(D,Q_2)$
3.38	6.544	7.306
3.93		
0.00		

Term	Term Frequency (df_i)	Term Probability (df_i/N_D)	Q_1 "university Riverside" {University, California, Riverside} $(qf_i)_1$	Q_2 "California university" {University, California, Riverside} $(qf_i)_2$	$P(q_i \text{ in } D)_1$	$P(q_i \text{ in } D)_2$	$P(Q_1,D)$	$P(Q_2,D)$
university	4	0.154	1	1	0.154	0.154	5.917E-03	1.775E-02
California	3	0.115	0	1	0.000	0.115		
Riverside	1	0.038	1	0	0.038	0.000		
one	2	0.077						
10	1	0.038						
within	1	0.038						
prestigious	1	0.038						
system	1	0.038						
only	1	0.038						
UC	1	0.038						
locate	1	0.038						
inland	1	0.038						
southern	1	0.038						
widely	1	0.038						
recognize	1	0.038						
most	1	0.038						
ethinic	1	0.038						
diverse	1	0.038						
research	1	0.038						
nation	1	0.038						

```java
import java.util.Map;
import java.util.Set;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;

public class ExerciseB {

    public ExerciseB() {

        stopwords = new HashSet<String>();
        stopwords.add("and");
        stopwords.add("or");
        stopwords.add("the");
        stopwords.add("is");
        stopwords.add("a");
        stopwords.add("in");
        stopwords.add("for");
        stopwords.add("");

        documents = new ArrayList<String>();
        documents.add(DOCUMENT_0);
        documents.add(DOCUMENT_1);
        documents.add(DOCUMENT_2);
        documents.add(DOCUMENT_3);
    }

    private byte asciiHash(String str) {

        int sum = 0;

        for (int i = 0; i < str.length(); i++)
            sum += str.charAt(i);

        return (byte) Math.floorMod(sum, MOD_VALUE);
    }

    private Map<String, Integer> getFrequencies(String document) {
```

```java
        Map<String, Integer> freqs = new HashMap<String, Integer>();
        String[] words = document.split("[ \\p{Punct}]");    // Remove spaces and punctuation

        for (int i = 0; i < words.length; i++) {
            words[i] = words[i].toLowerCase();
        }

        for (String word : words) {

            if (!stopwords.contains(word)) {

                freqs.put(word, freqs.getOrDefault(word, 0) + 1);
            }
        }

        return freqs;
    }

    private boolean getBit(byte b, int posFromLeft) {

        return getBit(Byte.toUnsignedInt(b), posFromLeft);
    }

    private boolean getBit(int i, int posFromLeft) {

        i = i >> (7 - posFromLeft);
        return i % 2 == 1;
    }

    private String byteToBinaryString(byte b) {

        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 8; i++)
            sb.append(getBit(b, i) ? '1' : '0');

        return sb.toString();
    }
```

```
    private byte booleanArrToByte(boolean[] arr) {

        int x = 0;
        for (int i = 0; i < 8; i++)
            x += arr[i] ? Math.pow(2, i) : 0;

        return (byte) x;
    }

    public byte simHash(String document) {

        Map<String, Integer> freqs = getFrequencies(document);
        int[] sumOfHashes = new int[8];

        for (Map.Entry<String, Integer> str : freqs.entrySet()) {

            byte hash = asciiHash(str.getKey());
//            System.out.println("Hashed \"" + str.getKey() + "\" to "
//                    + byteToBinaryString(hash));

            // Do this for every occurrence of str
            for (int i = 0; i < str.getValue(); i++) {

                // Iterate bit by bit
                for (int bit = 0; bit < 8; bit++) {

                    sumOfHashes[bit] += getBit(hash, bit) ? 1 : -1;
                }
            }
        }

        boolean[] weightSum = new boolean[8];

        for (int bit = 0; bit < 8; bit++)

            weightSum[7 - bit] = sumOfHashes[bit] > 0;
```

```java
        return booleanArrToByte(weightSum);
    }

    public int hammingDistance(byte b1, byte b2) {

        String str1 = byteToBinaryString(b1);
        String str2 = byteToBinaryString(b2);
        int sum = 0;

        for (int i = 0; i < str1.length(); i++) {

            sum += str1.charAt(i) == str2.charAt(i) ? 0 : 1;
        }

        return sum;
    }

    public void testStuff() {

//        for (String document : documents) {
//
//            System.out.print("Simhash(\"" + document + "\"): ");
//            System.out.println(byteToBinaryString(simHash(document)));
//        }

        List<Byte> simHashes = new ArrayList<Byte>();
        for (String document : documents)
            simHashes.add(simHash(document));

        int lo_bound = 0;
        int hi_bound = 3;

        for (int i = lo_bound; i <= hi_bound - 1; i++) {
            for (int j = i + 1; j <= hi_bound; j++) {

                System.out.println("SimHash(D" + i + "): " + byteToBinaryString(simHashes.get(i)));
                System.out.println("SimHash(D" + j + "): " + byteToBinaryString(simHashes.get(j)));

                System.out.println("hammingDist(D" + i + ", D" + j + "): "
```

```java
                                            + hammingDistance(simHashes.get(i), simHashes.get(j)) + "\n");
            }
        }
    }

    private final Set<String> stopwords;
    private final List<String> documents;

    private final int MOD_VALUE = 255; // 255 or 256

    private final String DOCUMENT_0 = "Tropical fish include fish found "
                                    + "in tropical environments around the world, "
                                    + "including both freshwater and salt water species.";
    private final String DOCUMENT_1 = "Basketball is a sport played in many countries. "
                                    + "NBA is a basketball league.";
    private final String DOCUMENT_2 = "NBA is a basketball tournament "
                                    + "in the US and other countries.";
    private final String DOCUMENT_3 = "Basketball is a sport played in many countries. "
                                    + "NBA is a basketball league. "
                                    + "Click here for more.";
}
```

Driver.java

```
public class Driver {



    public static void main(String[] args) {

        ExerciseB b = new ExerciseB();
            b.testStuff();
    }

}
```