**Programming Assignment 2 – Sander VanWilligen**

**MinHash:**

To calculate the total number of terms, I start by looking at each document in the collection. For each document, I read each line, and add all non-stop words on that line to a hashSet. I use a hashSet so that we only collect unique terms.

Based on the specifications supplied, it is not required to assign an integer to every single term in the set of terms for the document in order for the methods to function correctly. If you are referring to how I convert the terms to integers for the minHashSig method, I use the String.hashCode() method. I set up a separate function initially, but the hashCode function worked just as well and was faster than the method I designed.

All of the permutations are of the form ax + b % p. They all use the same p value, so we only need to generate that once. We do need to generate k a values and b values, so I use random to generate those based on p, and store the a-b pairs in an ArrayList.

**MinHashAccuracy:**

Table showing number of differences based on different values for NumPermutations and e:

|  |  |  |  |
| --- | --- | --- | --- |
|  | E = 0.04 | E = 0.07 | E = 0.09 |
| NumPerm = 400 | 1621 | 3 | 1 |
| NumPerm = 600 | 180 | 0 | 0 |
| NumPerm = 800 | 317 | 0 | 0 |

Obviously higher values of E lead to a lower average difference between the exact and approximate jaccard similarities. What surprised me was that values of NumPerm that were too high seemed to perform worse than lower ones for certain values (at least when I tested it, it could have just been variation in the hash functions).

**MinHashTime:**

For 600 permutations on the space folder:

The total time to compute the exact Jaccard similarity was 116667 ms. The total time to compute the approximate Jaccard similarity (including matrix computation time) was 733 ms.

**LSH:**

I actually did not find it that difficult to set up b hash tables. I created an arrayList of b hash tables. I had to initialize each element of this arraylist before trying to add elements to those tables, and I had to be careful about which rows were going into which table, but otherwise it was fairly simple. I also saw a good implementation online that used a single hashtable, but included the band index in the key elements for that table (essentially creating b hash tables). I tried out that code as well, but my code (using separate tables) seems to run faster in general.

To hash the tuple I do the following:

Start by setting the hash values to 1;

For each row in the tuple:

Set hash = hash + ((a \* rowValue) % p)

NearDuplicatesOf Pseudocode is as follows:

Index = index of the document in the collection (retrieved through iterative loop)

Results = {}

For each I in band:

Int hash = 1

For each row in a band:

Hash = hash + ((a \* rowValue) % p)

If HashMapList(i) contains the key, result:

Add every value associated with that key to results

Return results

**NearDuplicates:**

Table showing near duplicates for 10 different files (for the F17PA2 folder, with num permutations = 400 and similarity = 0.9):

|  |  |
| --- | --- |
| Original File | Similar Files |
| Space-0.txt | space-286.txt.copy2  space-713.txt.copy2  space-945.txt.copy5  space-713.txt  space-551.txt.copy4  hockey266.txt.copy7  hockey266.txt.copy5  hockey266.txt  hockey266.txt.copy2  space-0.txt.copy7  space-0.txt  space-0.txt.copy6  space-0.txt.copy5  space-0.txt.copy4  space-0.txt.copy3  space-0.txt.copy2  space-0.txt.copy1  hockey509.txt.copy1  space-375.txt.copy6 |
| Space-1.txt | baseball727.txt.copy1  baseball727.txt.copy2  space-1.txt  space-1.txt.copy3  space-1.txt.copy4  space-1.txt.copy5  space-1.txt.copy6  space-1.txt.copy1  space-1.txt.copy2  baseball727.txt  space-1.txt.copy7  space-445.txt.copy6  space-689.txt.copy1  baseball921.txt.copy5  baseball727.txt.copy6 |
| Space-2.txt | space-2.txt.copy3  space-2.txt.copy2  space-2.txt.copy5  space-2.txt.copy4  space-2.txt.copy7  space-2.txt.copy6  space-2.txt |
| Space-3.txt | space-3.txt.copy2  space-3.txt.copy1  space-3.txt.copy3  space-3.txt  space-3.txt.copy6  hockey783.txt.copy1  space-3.txt.copy5  space-3.txt.copy7 |
| Space-4.txt | space-4.txt.copy6  space-4.txt.copy7  space-4.txt.copy4  space-4.txt.copy5  space-4.txt  space-4.txt.copy2  space-4.txt.copy3  space-4.txt.copy1  space-286.txt.copy2  space-713.txt.copy2  space-945.txt.copy5  space-713.txt |
| Space-5.txt | space-5.txt  space-5.txt.copy6  space-5.txt.copy4  space-5.txt.copy3  space-5.txt.copy2  hockey183.txt.copy1  space-5.txt.copy1  space-651.txt.copy7 |
| Space-6.txt | baseball416.txt.copy6  space-6.txt.copy1  hockey897.txt  space-6.txt.copy6  space-6.txt.copy7  space-6.txt.copy2  hockey897.txt.copy4  space-6.txt.copy3  space-6.txt  hockey897.txt.copy2  space-6.txt.copy5 |
| Space-7.txt | space-7.txt.copy1  space-7.txt.copy2  space-7.txt.copy3  space-7.txt.copy4  space-7.txt.copy5  space-7.txt.copy6  space-7.txt.copy7  hockey64.txt.copy5  space-7.txt |
| Space-8.txt | hockey891.txt.copy2  hockey891.txt.copy3  hockey891.txt.copy1  space-8.txt.copy5  space-8.txt.copy4  hockey891.txt  space-8.txt.copy6  space-8.txt.copy3  space-8.txt.copy2  space-8.txt  hockey891.txt.copy6  hockey891.txt.copy7  hockey891.txt.copy4  hockey891.txt.copy5 |
| Space-9.txt | space-9.txt.copy7  space-9.txt.copy1  space-9.txt.copy2  space-9.txt.copy5  space-9.txt  space-9.txt.copy6  space-9.txt.copy3  space-9.txt.copy4 |

**Final Thoughts:**

Overall, I felt this project was straightforward. The notes, along with a larger supply of online documentation on the subject made it simpler to find possible solutions to issues and different parts. There were a couple of other similar classes/methods that I could compare my own against to see if I was getting realistic results. The discussion board was helpful as well. Overall, I thought my resulting code complied with the specification almost perfectly.