PA2 Report

MinHash and Locality Sensitive Hashing

Alex Huynh

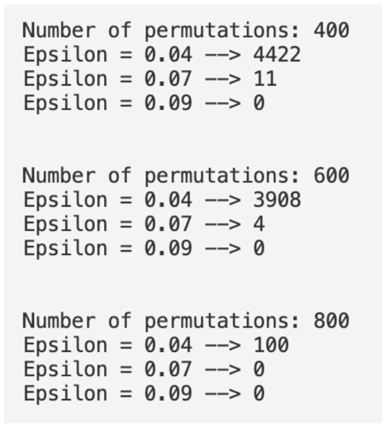
Com S 435
Iowa State University

1. ***What was my procedure to collect all terms of the documents and the data structure used for this?*** Firstly, I preprocessed every document according to point 2 on the assingment outline. Then, I looped through every document. For each document, I would maintain three data structures: $termsFreqMap$, $docTermsSet$, and $docTermsMultiSet$. The first is a HashMap from strings to integers, and it keeps track of words and their frequency in that document. Following that, every time a word is added to that map, or its value is updated in that map, then the word plus its frequency is added to $docTermsMultiSet$ (a HashSet of strings), which is the multiset of words in this document. For example, if the words were "hello hello" in the document, then $docTermsMultiSet$ would contain $hello1$ and $hello2$. Also, all words in the document are added to $docTermsSet$, which is a normal set of terms in the document. Once a document has been fully processed, I dump all of the terms from $docTermsSet$ into the main HashSet called $termsSet$. This main data structure is thus the set of all words in all the documents. I also dump all of the terms from $docTermsMultiSet$ into $termsMultiSet$, the multiset of all terms in all the documents. Furthermore, I map each document to its respective $termsFreqMap$. This mapping occurs in $mappingDocToTermFreqMap$, and it is used for the term document matrix (because I need to keep track of frequencies). Lastly, I map each document to its $docTermsMultiSet$. This mapping occurs in $mappingDocToMultiSet$, and it is used for the MinHash matrix (because this is how Jaccard similarity is estimated for multisets).

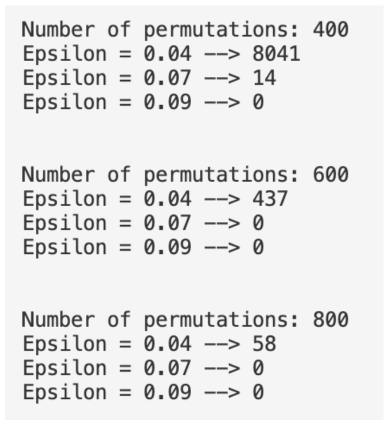
2. ***What was my procedure to assign an integer to each term?*** I made a list of numbers called $range$ that had numbers from 1 to $|termsMultiSet|$. Then, I would randomly assign each term in $termsMultiSet$ to a number in $range$. This assignment was onto and one-to-one. More specifically, each mapping would be stored in a HashMap called $permutation$, and it mapped not the term, but the term's hashcode to a random integer from 1 to $|termsMultiSet|$.

3. ***What were my permutations used, and the process used to generate random permutations?*** Firstly, I made an ArrayList $range$ that contained numbers from 1 to $|termsMultiSet|$ as described earlier. Then I shuffled these numbers using java.util.Collections.shuffle($range$). As a result, all of the numbers are mixed up. Then I looped through every term in $termsMultiSet$. Inside this loop, I would map the term's hash code to the ith value (ith being the ith iteration of the loop) of the shuffled list $range$ into a HashMap $permutation$. I make k of these HashMaps (each time shuffling $range$), where $k = numPermutations$. I finally stored all of these HashMaps into the an ArrayList called $permutations$ (note the plural).

4. In assessing the class *MinHashAccuracy*, the results below show that when keeping the number of permutations constant, increasing epsilon would reduce the number of bad approximations. This makes sense as we allow the difference in Jaccard similarity to be greater. Thus, we accept larger differences, lowering the number of "detected" bad approximations. Now, when number of permutations is increased, the number of bad approximations is also reduced. That is because the approximate Jaccard similarity is better estimated when the number of permutations increases, which makes it more close to the exact Jaccard similarity. Thus, there are less bad approximations. When you combine the two (increase both), then there are barely (0 in this example) any bad approximations.



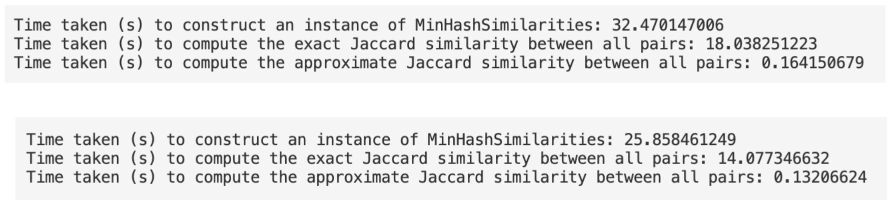
```
Number of permutations: 400
Epsilon = 0.04 --> 4422
Epsilon = 0.07 --> 11
Epsilon = 0.09 --> 0


Number of permutations: 600
Epsilon = 0.04 --> 3908
Epsilon = 0.07 --> 4
Epsilon = 0.09 --> 0


Number of permutations: 800
Epsilon = 0.04 --> 100
Epsilon = 0.07 --> 0
Epsilon = 0.09 --> 0
```



```
Number of permutations: 400
Epsilon = 0.04 --> 8041
Epsilon = 0.07 --> 14
Epsilon = 0.09 --> 0


Number of permutations: 600
Epsilon = 0.04 --> 437
Epsilon = 0.07 --> 0
Epsilon = 0.09 --> 0


Number of permutations: 800
Epsilon = 0.04 --> 58
Epsilon = 0.07 --> 0
Epsilon = 0.09 --> 0
```

Figure 1: Two examples of running MinHashAccuracy for 9 inputs

5. Regarding the class $MinHashTime$, the results below show that constructing an instance of MinHashSimilarities takes a decent amount of time. That is the same for computing the Jaccard similarity between every document, which makes sense because the term document matrix is large–$O(NT)$, where N is the number of documents and T is $|termsSet|$. On the other hand, approximating the Jaccard similarity between all document pairs is fast because the MinHash matrix is significantly smaller–$O(Nk)$, where k is the number of permutations. Thus when comparing the two in the benchmark of computing the Jaccard similarity between every single document pair, the term document matrix's runtime is significantly slower.

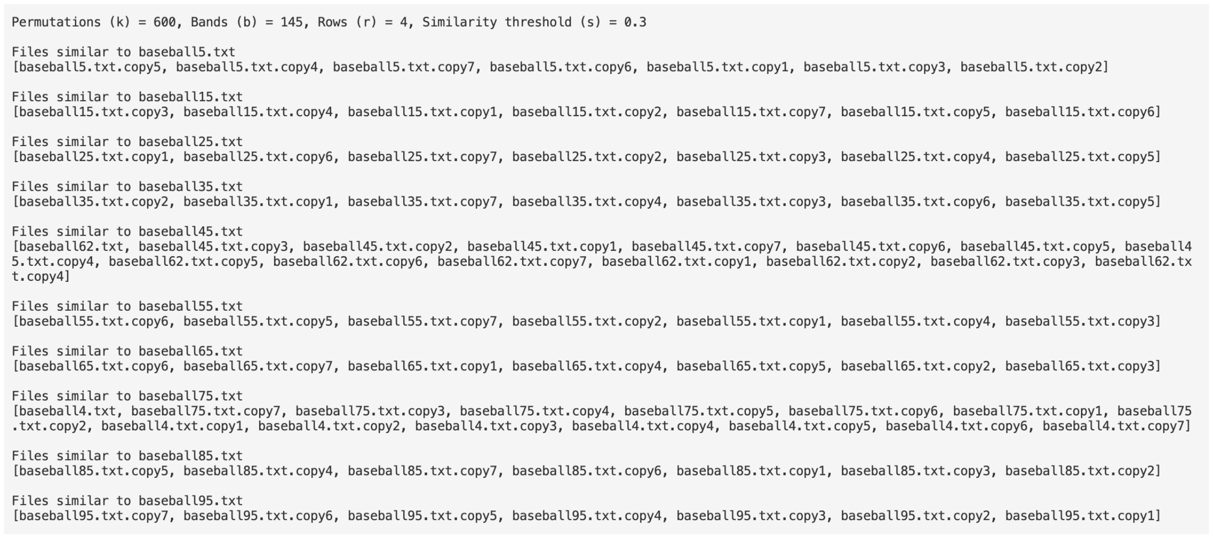


```
Time taken (s) to construct an instance of MinHashSimilarities: 32.470147006
Time taken (s) to compute the exact Jaccard similarity between all pairs: 18.038251223
Time taken (s) to compute the approximate Jaccard similarity between all pairs: 0.164150679
```

```
Time taken (s) to construct an instance of MinHashSimilarities: 25.858461249
Time taken (s) to compute the exact Jaccard similarity between all pairs: 14.077346632
Time taken (s) to compute the approximate Jaccard similarity between all pairs: 0.13206624
```

Figure 2: Two examples of MinHashTime output

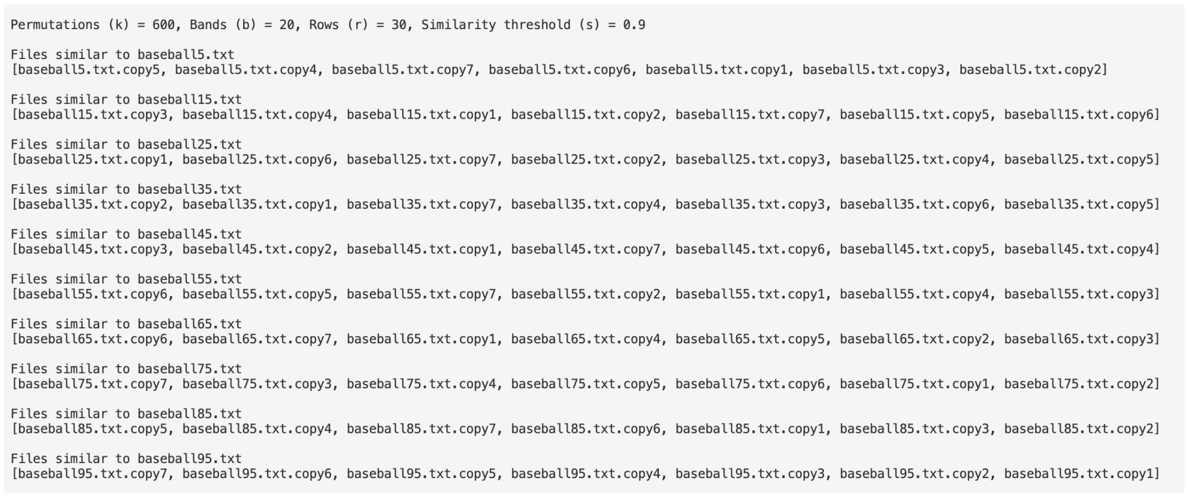
6. For testing similarity search, I used numPermutations $k = 600$ and similarity threshold values $s = 0.3$ and $s = 0.9$. We see that increasing the similarity threshold reduces the output of similar documents from similarity search, which makes sense (in particular, notice how $s = 0.3$ led to some false positives). Lower similarity threshold means there will be more bands and thus a higher rate of documents to hash to the same bucket. Also, less similar items will be viewed as being "similar". We furthermore see that we received no false positives due to how accurate our Jaccard approximation is when $s = 0.9$ (when using $k = 600$ at least). Another observation is that we successfully found all the true positives. On another note, to compute the optimal number of bands, we needed to solve $s = (\frac{1}{b})^{1/r}$ for b. To do so, I simplified the equation to $y(b) = b\ln(b) + k\ln(s) = 0 \implies \frac{dy}{db} = \ln(b) + 1$. Thus, I can use the Newton Raphson Method [1] to approximate b.



```
Permutations (k) = 600, Bands (b) = 145, Rows (r) = 4, Similarity threshold (s) = 0.3

Files similar to baseball5.txt
[baseball5.txt.copy5, baseball5.txt.copy4, baseball5.txt.copy7, baseball5.txt.copy6, baseball5.txt.copy1, baseball5.txt.copy3, baseball5.txt.copy2]

Files similar to baseball15.txt
[baseball15.txt.copy3, baseball15.txt.copy4, baseball15.txt.copy1, baseball15.txt.copy2, baseball15.txt.copy7, baseball15.txt.copy5, baseball15.txt.copy6]

Files similar to baseball25.txt
[baseball25.txt.copy1, baseball25.txt.copy6, baseball25.txt.copy7, baseball25.txt.copy2, baseball25.txt.copy3, baseball25.txt.copy4, baseball25.txt.copy5]

Files similar to baseball35.txt
[baseball35.txt.copy2, baseball35.txt.copy1, baseball35.txt.copy7, baseball35.txt.copy4, baseball35.txt.copy3, baseball35.txt.copy6, baseball35.txt.copy5]

Files similar to baseball45.txt
[baseball62.txt, baseball45.txt.copy3, baseball45.txt.copy2, baseball45.txt.copy1, baseball45.txt.copy7, baseball45.txt.copy6, baseball45.txt.copy5, baseball4
5.txt.copy4, baseball62.txt.copy5, baseball62.txt.copy6, baseball62.txt.copy7, baseball62.txt.copy1, baseball62.txt.copy2, baseball62.txt.copy3, baseball62.tx
t.copy4]

Files similar to baseball55.txt
[baseball55.txt.copy6, baseball55.txt.copy5, baseball55.txt.copy7, baseball55.txt.copy2, baseball55.txt.copy1, baseball55.txt.copy4, baseball55.txt.copy3]

Files similar to baseball65.txt
[baseball65.txt.copy6, baseball65.txt.copy7, baseball65.txt.copy1, baseball65.txt.copy4, baseball65.txt.copy5, baseball65.txt.copy2, baseball65.txt.copy3]

Files similar to baseball75.txt
[baseball4.txt, baseball75.txt.copy7, baseball75.txt.copy3, baseball75.txt.copy4, baseball75.txt.copy5, baseball75.txt.copy6, baseball75.txt.copy1, baseball75
.txt.copy2, baseball4.txt.copy1, baseball4.txt.copy2, baseball4.txt.copy3, baseball4.txt.copy4, baseball4.txt.copy5, baseball4.txt.copy6, baseball4.txt.copy7]

Files similar to baseball85.txt
[baseball85.txt.copy5, baseball85.txt.copy4, baseball85.txt.copy7, baseball85.txt.copy6, baseball85.txt.copy1, baseball85.txt.copy3, baseball85.txt.copy2]

Files similar to baseball95.txt
[baseball95.txt.copy7, baseball95.txt.copy6, baseball95.txt.copy5, baseball95.txt.copy4, baseball95.txt.copy3, baseball95.txt.copy2, baseball95.txt.copy1]
```

Figure 3: Similarity search with $k = 600, s = 0.3$

[1] `https://web.mit.edu/10.001/Web/Course_Notes/NLAE/node6.html`



```
Permutations (k) = 600, Bands (b) = 20, Rows (r) = 30, Similarity threshold (s) = 0.9

Files similar to baseball5.txt
[baseball5.txt.copy5, baseball5.txt.copy4, baseball5.txt.copy7, baseball5.txt.copy6, baseball5.txt.copy1, baseball5.txt.copy3, baseball5.txt.copy2]

Files similar to baseball15.txt
[baseball15.txt.copy3, baseball15.txt.copy4, baseball15.txt.copy1, baseball15.txt.copy2, baseball15.txt.copy7, baseball15.txt.copy5, baseball15.txt.copy6]

Files similar to baseball25.txt
[baseball25.txt.copy1, baseball25.txt.copy6, baseball25.txt.copy7, baseball25.txt.copy2, baseball25.txt.copy3, baseball25.txt.copy4, baseball25.txt.copy5]

Files similar to baseball35.txt
[baseball35.txt.copy2, baseball35.txt.copy1, baseball35.txt.copy7, baseball35.txt.copy4, baseball35.txt.copy3, baseball35.txt.copy6, baseball35.txt.copy5]

Files similar to baseball45.txt
[baseball45.txt.copy3, baseball45.txt.copy2, baseball45.txt.copy1, baseball45.txt.copy7, baseball45.txt.copy6, baseball45.txt.copy5, baseball45.txt.copy4]

Files similar to baseball55.txt
[baseball55.txt.copy6, baseball55.txt.copy5, baseball55.txt.copy7, baseball55.txt.copy2, baseball55.txt.copy1, baseball55.txt.copy4, baseball55.txt.copy3]

Files similar to baseball65.txt
[baseball65.txt.copy6, baseball65.txt.copy7, baseball65.txt.copy1, baseball65.txt.copy4, baseball65.txt.copy5, baseball65.txt.copy2, baseball65.txt.copy3]

Files similar to baseball75.txt
[baseball75.txt.copy7, baseball75.txt.copy3, baseball75.txt.copy4, baseball75.txt.copy5, baseball75.txt.copy6, baseball75.txt.copy1, baseball75.txt.copy2]

Files similar to baseball85.txt
[baseball85.txt.copy5, baseball85.txt.copy4, baseball85.txt.copy7, baseball85.txt.copy6, baseball85.txt.copy1, baseball85.txt.copy3, baseball85.txt.copy2]

Files similar to baseball95.txt
[baseball95.txt.copy7, baseball95.txt.copy6, baseball95.txt.copy5, baseball95.txt.copy4, baseball95.txt.copy3, baseball95.txt.copy2, baseball95.txt.copy1]
```

Figure 4: Similarity search with $k = 600, s = 0.9$