Programming Assignment 2

Nathan Friend

COM S 535: Algorithms for Large Data Sets:
Theory and Practice

2018/10/28

MinHash

Your procedure to collect all terms of the documents and the data structure used for this.

I built a class named `TermExtractor` that is responsible for extracting terms from a particular document. The class contains a method named `extractfromFile` that takes a `File` object as input as returns a multiset of terms (in the form of a `List<String>`) that appear in the document.

An instance of `TermExtractor` holds a number of preprocessing classes that implement the `IFilter` interface. Each `IFilter` accepts a list of terms as input and outputs a list of processed terms. In its current form, the `TermExtractor` class uses these `IFilters` in the following order:

- `PatternExtractorFilter`: Extracts terms using a regular expression
- `LowercaseFilter`: Converts all terms to lowercase
- `RemovePatternFilter`: Remove characters from the provided tokens
- `WordLengthFilter`: Remove terms of a certain length
- `StopFilter`: Removes specific terms

The output of these `IFilters` is piped into my `TermDocumentMatrix` class using the `loadTerms` method. This method stores the terms in a `HashMap<String, HashMap<String, Integer>>`, where the first `String` is the name of the document, the second `String` is the term, and the `Integer` is the term frequency. This data structure is then converted into an `int[][]` once all files have been processed.

Your procedure to assign an integer to each term.

To convert a term to an integer, I use Java's built-in `String.hashCode()` method.

The permutations used, and the process used to generate random permutations.

To generate *k* hash functions, I use the hash function $(ax + b)\%p$ with *k* randomly chosen values for *a* and *b*. These random constants are validated to ensure there are no duplicate hash functions. These *k* values of *a* and *b*, along with the prime *p*, are stored in an instance of `PermutationParams`.

MinHashAccuracy

Run the program with following choices of NumPermutations: 400, 600, 800 and following choices for ε: 0.04, 0.07, 0.09. Run the program on the files from space.zip. Report the number of pairs for which approximate and exact similarities differ by more than ε for each combination.

Here is the output of `MinHashAccuracy` using the specified parameters:

```
Number of bad approximations with numPermutations=400 and epsilon=0.040000: 11505
Number of bad approximations with numPermutations=600 and epsilon=0.040000: 6847
Number of bad approximations with numPermutations=800 and epsilon=0.040000: 1628
Number of bad approximations with numPermutations=400 and epsilon=0.070000: 182
Number of bad approximations with numPermutations=600 and epsilon=0.070000: 70
Number of bad approximations with numPermutations=800 and epsilon=0.070000: 46
Number of bad approximations with numPermutations=400 and epsilon=0.090000: 14
Number of bad approximations with numPermutations=600 and epsilon=0.090000: 3
Number of bad approximations with numPermutations=800 and epsilon=0.090000: 6
```

What can you conclude from these numbers?

The quality of the Jaccard similarity approximations is directly related to the number of permutations used to construct a MinHash matrix. The more permutations, the better the estimates.

MinHashTime

Use 600 permutations on files from space.zip. Report the total run time to calculate exact Jaccard similarities and approximate Jaccard similarities (between all possible pairs).

Here is the output of `MinHashTime` using the specified parameters:

```
Number of seconds to construct a MinHashSimilarities instance: 43.939
Number of seconds to compute the exact Jaccard similarity between all documents: 5.925
Number of seconds to compute the approximate Jaccard similarity between all documents: 2.074
```

NearDuplicates

Finally, run nearDuplicateDetector on the files from F17PA2.zip (with at least two choices of s). Run the program on at least 10 different inputs. For each input: List all the files that are returned as near duplicates in your report.

Below is the output from the `nearDuplicateDetector` method. The program was run against five different files (space-0.txt, baseball0.txt, hockey0.txt, space91.txt, hockey53.txt) and with two different threshold values (0.85 and 0.4):

```
Found the following near duplicates of file space-0.txt using 600 permutations and threshold
value 0.85: space-0.txt.copy1, space-0.txt.copy2, space-0.txt.copy3, space-0.txt.copy4, space-
0.txt.copy5, space-0.txt.copy6, space-0.txt.copy7

Found the following near duplicates of file baseball0.txt using 600 permutations and threshold
value 0.85: baseball0.txt.copy1, baseball0.txt.copy2, baseball0.txt.copy3,
baseball0.txt.copy4, baseball0.txt.copy5, baseball0.txt.copy6, baseball0.txt.copy7

Found the following near duplicates of file hockey0.txt using 600 permutations and threshold
value 0.85: hockey0.txt.copy1, hockey0.txt.copy2, hockey0.txt.copy3, hockey0.txt.copy4,
hockey0.txt.copy5, hockey0.txt.copy6, hockey0.txt.copy7
```

```
Found the following near duplicates of file space-91.txt using 600 permutations and threshold
value 0.85: space-91.txt.copy1, space-91.txt.copy2, space-91.txt.copy3, space-91.txt.copy4,
space-91.txt.copy5, space-91.txt.copy6, space-91.txt.copy7

Found the following near duplicates of file hockey53.txt using 600 permutations and threshold
value 0.85: hockey53.txt.copy1, hockey53.txt.copy2, hockey53.txt.copy3, hockey53.txt.copy4,
hockey53.txt.copy5, hockey53.txt.copy6, hockey53.txt.copy7

Found the following near duplicates of file space-0.txt using 600 permutations and threshold
value 0.40: space-0.txt.copy1, space-0.txt.copy2, space-0.txt.copy3, space-0.txt.copy4, space-
0.txt.copy5, space-0.txt.copy6, space-0.txt.copy7, space-531.txt, space-531.txt.copy1, space-
531.txt.copy2, space-531.txt.copy3, space-531.txt.copy4, space-531.txt.copy5, space-
531.txt.copy6, space-531.txt.copy7

Found the following near duplicates of file baseball0.txt using 600 permutations and threshold
value 0.40: baseball0.txt.copy1, baseball0.txt.copy2, baseball0.txt.copy3,
baseball0.txt.copy4, baseball0.txt.copy5, baseball0.txt.copy6, baseball0.txt.copy7

Found the following near duplicates of file hockey0.txt using 600 permutations and threshold
value 0.40: hockey0.txt.copy1, hockey0.txt.copy2, hockey0.txt.copy3, hockey0.txt.copy4,
hockey0.txt.copy5, hockey0.txt.copy6, hockey0.txt.copy7

Found the following near duplicates of file space-91.txt using 600 permutations and threshold
value 0.40: space-537.txt, space-537.txt.copy1, space-537.txt.copy4, space-537.txt.copy5,
space-537.txt.copy6, space-537.txt.copy7, space-91.txt.copy1, space-91.txt.copy2, space-
91.txt.copy3, space-91.txt.copy4, space-91.txt.copy5, space-91.txt.copy6, space-91.txt.copy7,
space-950.txt, space-950.txt.copy1, space-950.txt.copy2, space-950.txt.copy3, space-
950.txt.copy4, space-950.txt.copy5, space-950.txt.copy6, space-950.txt.copy7

Found the following near duplicates of file hockey53.txt using 600 permutations and threshold
value 0.40: hockey53.txt.copy1, hockey53.txt.copy2, hockey53.txt.copy3, hockey53.txt.copy4,
hockey53.txt.copy5, hockey53.txt.copy6, hockey53.txt.copy7
```

Documentation

Documentation for all classes can be found in Javadoc format in the `doc` folder. To view the Javadoc documentation, open `doc/index.html` in a web browser.