Project 2

Building a Boolean Search Engine

Data Mining

Instructor: Arjun Mukherjee

1. Boolean Retrieval

We learnt about Boolean text retrieval in class. Please refer to slides/lecture notes given in class and algorithm details in these drafts (with links) [Schutze, 2014], [IIR Chap 01]

2. Building a Toy Search Engine

Our goal is to implement the methods AND, OR, NOT and perform some queries. We have a small corpus (document collection) of Amazon.com reviews on electronics. The Java codebase is an Eclipse project available at: http://www2.cs.uh.edu/~arjun/courses/dm/hw_proj/proj_2.zip .You are required to download the codebase and work with it.

Note: Students who decide to not use Java need to start with the document matrix and associated vocabulary (detailed below).

Corpus (document collection): mini_proj/all.txt
This contains the raw reviews. Each line is a review/document in our context. There are total 1248 documents. Documents in the posting list should be indexed from 1 (and not 0) where the first document refers to the first line in mini_proj/all.txt.
The corpus is also available in document matrix format: mini_proj/docs.txt with associated vocabulary mapping mini_proj/vocab_map.txt generated using a helper method DatasetFormatter.

Pre-processing: DatasetFormatter.java
This class provides preprocessing (parsing, generating the vocabulary map, filtering stopwords based on counts, etc.) already implemented to get you started. You are required to work on the formatted data produced by this class. Your best bet is to use it directly within the constructor of BooleanRetrieval.java class as follows:

```
public BooleanRetrieval() throws Exception{
        // Initialize variables and Format the data using a pre-processing class and
set up variables
        invIndex = new HashMap<String, Set<Integer>>();
        DatasetFormatter formater = new DatasetFormatter();
        formater.textCorpusFormatter("./all.txt");
        docs = formater.getDocs();
        vocab = formater.getVocab();
```

If you don't wish to call the preprocessing class [those not using Java/my helper methods], then you are also provided the docs and vocabulary mapping (starts at 0) at mini_proj/docs.txt and mini_proj/vocab_map.txt respectively.
It should also be noted that we work in lowercases in this homework. That is all documents are converted to lower cases (see line 74 in DatasetFormatter.java)

3. Methods to implement

(1) `void createPostingList()`
You are required to build the inverted index in the method `void createPostingList()` for this preprocessed corpus. Note that you are required to work with the pre-processed data, i.e., output of `DatasetFormatter.textCorpusFormatter()` defined in Section 2 OR the document matrix/vocabulary defined by the files mini_proj/docs.txt and mini_proj/vocab_map.txt (which are the same). Do NOT work with the raw corpus mini_proj/all.txt and re-parse!

Hint:

```
void createPostingList() {
```

```
//Initialze the inverted index with a SortedSet (so that the later additions become
easy!)
        for(String s:vocab){
              invIndex.put(s, new TreeSet<Integer>());
        }
        //for each doc
        for(int i=0; i<docs.length; i++){
              //for each word of that doc
              for(int j=0; j<docs[i].length; j++){
                    //Get the actual word in position j of doc i
                    String w = map.get(docs[i][j]);
                    //Now simply update the posting list of this word w in the invIndex
by adding the document i
```

(2) `Set<Integer> intersection(Set<Integer> a, Set<Integer> b)`
This method accepts two posting lists using the Set collections and returns the intersection of them. You are required to implement it using the algorithm for intersection discussed in class.

Hint:

```
Set<Integer> intersection(Set<Integer> a, Set<Integer> b){
        /*
        First convert the posting lists from sorted set to something we
        can iterate easily using an index. I choose to use ArrayList<Integer>.
        Once can also use other enumerable.
         */
        ArrayList<Integer> PostingList_a = new ArrayList<Integer>(a);
        ArrayList<Integer> PostingList_b = new ArrayList<Integer>(b);
        TreeSet result = new TreeSet();

        //Set indices to iterate two lists. I use i, j
        int i = 0;
        int j = 0;

        while(i!=PostingList_a.size() && j!=PostingList_b.size()){
   …
```

Then we can use the result of intersection to perform a AND query as follows:

```
Set <Integer> evaluateANDQuery(String a, String b){
        return intersection(invIndex.get(a), invIndex.get(b));
   }
```

4. Sample input and output

A. Evaluating/checking posting lists in the inverted index

Let us get a feel of the corpus by exploring some examples. Assuming you have implemented the inverted index correctly. Numbering document indices starting at 1 (in our corpus), the posting lists for the following words should be as follows:

```
freeze -> [129, 313, 442, 493, 912] (i.e., the term "freeze" is present in documents
129, 313, etc.)
ctrl -> [23, 47, 138, 232, 498, 882, 993, 1106, 1124, 1218]
wifi -> [46, 66, 113, 158, 746, 1248]
cpu -> [604, 800, 959, 1156]
```

B. Evaluating/checking AND queries

Assuming you have implemented the intersection method correctly, then we should expect the following results for the following queries:

```
mouse AND wifi -> [113, 158] (i.e., there are two documents in our corpus which contain
both the terms "mouse" and "wifi". And these are document at line 113 and 158 in all.txt)
mouse AND scrolling -> [80, 86, 348, 1029]
errors AND report -> [] (i.e., there are no documents in the collection containing both
the terms "errors" and "report")
```

5. Project Requirements

You will be evaluated on two aspects in this homework when you demo it. In the actual demo new terms will be given and you are required to run your implementation and produce the result.

Task 1: Inverted index construction and posting list. In this task you will be asked to produce the posting list of any word in our vocabulary (i.e., mini_proj/ vocab_map.txt) and you should generate its posting list in the sorted order as it appears in the examples of Section 4(A). For practice try to generate posting lists for these terms:

```
tossed
excited
great
```

Task 2: Implementation of intersection method to compute 2 term AND queries. In this task you will be asked to produce the result of performing a Boolean AND query with two terms in our corpus and you should be able to generate the result (preferably in sorted order) as in Section 4(B). For practice try these queries:

```
plastic AND mouse
mouse AND keyboard
mouse AND ergonomic
```

6. Implementing UNION and NOT operations on posting lists

Task 3: Implement the method `Set<Integer> union(Set<Integer> a, Set<Integer> b)` to evaluate and retrieve documents using the OR Boolean query. Union operation on queries `mouse OR keyboard` like simply means that we should retrieve all documents that appear either in the posting list of the term "mouse" or "keyboard".
NOTE: You are required to implement OR and NOT extending the ideas of OR to ensure linearity. Hence you cannot use library methods (e.g., .addAll in Java Collections for set union)

You can use the `evaluateORQuery()` method implemented to return the result of an OR query containing two terms.

```
Set <Integer> evaluateORQuery(String a, String b){
                    return union(invIndex.get(a), invIndex.get(b));
   }
```

Assuming you have implemented union correctly, we should expect to get the following results for these queries:

```
youtube OR reported -> [19, 67, 122, 227, 313, 342, 377, 507, 659, 825, 846]
errors OR report -> [115, 251, 471, 508, 674, 784, 821, 1068, 1080, 1111, 1158, 1245]
hell OR movie -> [3, 89, 147, 192, 235, 262, 336, 342, 558, 766, 864, 882, 1120, 1244]
```

Task 4: Implement the method `Set<Integer> not(Set<Integer> a)` to evaluate NOT queries. For example, if we wish to get the list of documents NOT containing the word/term "mouse", then we can use the following function:
`not(invIndex.get("mouse"))` which first retrieves the posting list for mouse form the inverted index and then sends this list to be complemented by the `not(Set<Integer> a)` method. And then use the following method to return results for queries like mouse AND (NOT keyboard):

```
Set <Integer> evaluateAND_NOTQuery(String a, String b){
        return intersection(invIndex.get(a), not(invIndex.get(b)));
   }
```

Assuming you have implemented this correctly, we should expect to get the following results for these queries:

```
scroll AND (NOT mouse) -> [40, 43, 55, 130, 213, 310, 441, 462, 477, 508, 529, 565, 773,
852, 879, 901, 902, 933, 1060, 1102, 1103, 1157, 1221]
lenovo AND (NOT logitech) -> [360, 373, 379, 451, 517, 540, 787, 869, 942, 1055, 1146]
```

7. Final Output and Submission Guidelines

You will be evaluated on 4 kinds of queries: PLIST (posting list), AND, OR, AND-NOT. PLIST will contain one query term and other types will include at most 2 query terms. Make sure to provide a README file with your implementation which details on how to run your code with the specified inputs. You are also required to make sure that your results are generated in the **exactly same format** as the examples provided in this project description. More specifically, your code should take command line arguments as follows:

$> project_executable query_type query_string output_file_path

where query_type takes on one of the 4 values: PLIST, AND, OR, AND-NOT. Query_string takes the actual query and output_file_path is the filename with path in ./ that should report the result of your query.

IMP evaluation requirement: You are required to demo the code in class with example test queries provided on the due date.

Example runs:

$> project_executable PLIST cpu cpu_plist.txt
Should produce
`cpu -> [604, 800, 959, 1156]`
as the contents of cpu_plist.txt

$> project_executable AND mouse AND scrolling and_result.txt
Should produce
`mouse AND scrolling -> [80, 86, 348, 1029]`
as the contents of and_result.txt

$> project_executable AND-NOT Lenovo AND (NOT logitech) and_not_result.txt
Should produce
`lenovo AND (NOT logitech) -> [360, 373, 379, 451, 517, 540, 787, 869, 942, 1055, 1146]`
as the contents of and_not_result.txt

These guidelines are hard and no alterations will be possible as output will be evaluated using automated scripts. **If your output does not abide by the instructions, you risk your score as our automated checking will not be able to evaluate your results.**