Assignment 1

Information Retrieval and Text Mining 23/24

Publication: 2023-10-26
Submission Deadline: 2023-11-07
Discussion Session: 2023-11-14

Roman Klinger
Yarik Menchaca Resendiz
irtm-teachers@ims.uni-stuttgart.de

- **Groups:** Working in groups of up to three people is encouraged, up to four people is allowed. More people are not allowed. Copying results from one group to another (or from elsewhere) is not allowed. Changing groups during the term is allowed.
- **Grading:** Passing the assignments is a requirement for participation in the exam in all modules IRTM can be part of. Altogether 80 points need to be reached. There are five assignments with 20 pen & paper points and 10 programming points each. That means, altogether, 150 points can be reached.
- **Submission:** First make a group in Ilias, then submit the PDF. Write all group members on the first page of the PDF. Only submit *one* PDF file. If you are technically not able to make a group (it seems that happens on Ilias from time to time), do not submit a PDF multiple times by multiple people – only submit it once. Submission for the programming tasks should also be in the same PDF.
- **Make it understandable:** Do the best you can such that we can understand what you mean. Explain your solutions, comment your code. Print the code in a readable format, write your solutions in a way we can read them.
- **Handwriting:** We typically receive some submissions which are handwritten. That is fine, but if you submit handwritten solutions, make sure that they are well organized, easy to read and to understand, and that there is not doubt about the interpretation of letters. If you think that this might be hard, please typeset the solutions with a computer. We might reduce points if it's really tough for us and cannot read your submission properly.
- **Language:** Please submit your solutions in English. We have limited capacity of correcting German submissions.

Pen and Paper Task 1 (5 points)

Given the following documents:

- Document 1: `water nice`
- Document 2: `water`
- Document 3: `water`
- Document 4: `water`
- Document 5: `sun nice`
- Document 6: `sun is nice`
- Document 7: `water sun`
- Document 8: `sun`
- Document 9: `sun`

Subtask A

Build the full inverted index (without positional information) for these documents. Do not perform normalization of terms.

Subtask B

Add skip pointers to the index from the previous task. Provide an example query which demonstrates the usefulness of skip pointers for your index and explain why this query can be answered in a more efficient way with skip pointers than without.

Pen and Paper Task 2 (6 points)

Write an algorithm `tokenize(text)` in pseudo code which takes a string as input and outputs a list or array of tokens (where each token is also a string). Your algorithm should not remove or omit any characters except for white space. Split tokens on all symbols except for letters or numbers. Further, the algorithm should have a linear runtime in the number of input characters n ($\mathcal{O}(n)$).

Write down the algorithm as pseudo code. Explain the algorithm based on the example "O'Really isn't in S." Explain why your algorithm has a linear runtime is general (not just based on this example).

Pen and Paper Task 3 (9 points)

(content will be discussed in the lecture on Tuesday, 31st)
To calculate edit distances, we initialized a matrix as follows in class:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & \dots \\ 1 & & & & \\ 2 & & & & \\ 3 & & & & \\ \vdots & & & & \end{bmatrix}$$

That means, the values in the cells are given as $m = |u|$, $n = |v|$, $D_{0,0} = 0$, $D_{i,0} = i, 1 \leq i \leq m$, $D_{0,j} = j, 1 \leq j \leq n$. u and v are the input sequences.

In class, we calculated the Levenshtein distance by filling each cell in a matrix with the following rule:

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} & +0 \text{ if } u_i = v_j \\ D_{i-1,j-1} & +1 \text{ (replace)} \\ D_{i,j-1} & +1 \text{ (insert)} \\ D_{i-1,j} & +1 \text{ (delete)} \end{cases}$$

For the Damerau-Levenshtein distance, the cells are filled by the following rule:

$$D_{i,j} = \min \begin{cases} D_{i-1,j-1} & +0 \text{ if } u_i = v_j \\ D_{i-1,j-1} & +1 \text{ (replace)} \\ D_{i,j-1} & +1 \text{ (insert)} \\ D_{i-1,j} & +1 \text{ (delete)} \\ D_{i-2,j-2} & +1 \text{ (transpose, if } u_i = v_{j-1} \text{ and } u_{i-1} = v_j) \end{cases}$$

Calculate the Damerau-Levenshtein distance for the two terms “Mouse” and “Muose”. Write down the complete matrix and explain.

Programming Task 1 (10 points)

In the Ilias exercise session you found this PDF in, there is a file available called twitter.csv.bz2. Unpack this:

```
bunzip2 twitter.csv.bz2
```

Each line corresponds to one Tweet. The first column is an id, the second column is the Twitter handle (a user ID), the third column is the name of the user, the fourth column is the Tweet text (with special tokens [NEWLINE] and [TAB]).

Implement a method `index(filename)` which takes the path to the file as an argument and puts all documents into a non-positional inverted index. You can assume that your computer's memory is sufficient to store all postings lists. For the case that it is not, only index a subset of the documents, but it would be better if you think about how to make your program more efficient to make it fit (note that you do not need to store the document text as part of your index).

The index should consist of a dictionary and postings lists. Each entry of the dictionary should contain three values: The (normalized) term, the size of the postings list, the pointer to the postings list. Your data structure should be prepared to be able to store the postings lists separately from the dictionary, therefore do not just put a List data structure as value into a HashMap/Tree or Python dictionary. Instead, put the postings lists into a data structure that you could store elsewhere (for instance, use a separate id-to-value mapping for that). It is your decision if and how you normalize tokens and terms. You can also decide to filter out Tweets if you think they are not relevant, to clean the data. Please describe your decisions in your submission.

The postings list itself consists of postings which contain each a document id and a pointer to the next postings. For the dictionary, you can use hashing methods included in your programming language (like dictionary in Python or HashMap in Java) or tree structures as available in your programming language (for instance TreeMap in Java). For the postings lists, you can either implement the lists from scratch or use existing data structures (like lists in Python or LinkedList in Java).

Then implement a method `query(term)`, where the argument represents one term as a `string`. It should return the postings list for that term.

Then, implement a method `query(term1, term2)`, where you assume that both terms are connected with a logical AND. Implement the intersection algorithm as discussed in the lecture for intersecting two postings lists. Do not access the lists array-style (for instance `listname[5]` where 5 is the position of the element you want to get). Use an iterator (in Python `listiter = iter(listname); next(listiter)` or in Java `iterator.next()`).

You can choose the programming language. Comment your code! Submit all code in the same PDF as the other tasks (pretty printed). Please note, you won't receive all points if the code is not commented properly.

In addition, please query your index for the information need "show me tweets of people who talk about the side effects of malaria vaccines". Provide us with your query and (a subset) of results. The results should be minimally represented by the Tweet-ID and optionally also the Tweet text.