Assignment 1

Information Retrieval and Text Mining 20/21

Publication: 2020-11-10
Submission Deadline: 2020-11-19
Discussion Video Online: 2020-11-26
Live Q&A Session: 2020-12-03

Roman Klinger
Valentino Sabbatino, Tordis Daum, Tobias Schmid

- **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.

Pen and Paper Task 1 (3 points)

Given the following documents:

- Document 1: `sun nice`
- Document 2: `sun`
- Document 3: `sun`
- Document 4: `sun`
- Document 5: `water nice`
- Document 6: `water is nice`
- Document 7: `sun sun`
- Document 8: `water`
- Document 9: `water`
- Document 10: `beer`

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'Kennedy hasn't gone to S."
Explain that the runtime is linear (in general, not based on this example).

Pen and Paper Task 3 (5 points)

Consider the following fragment of a positional index in the format
`term:  (freq.)  docID: <position, position, position, ...>; docID: ...:`

```
Gates:     (4) 1: <3>; 2: <6>;    3: <2,17>; 4: <1>;
IBM:       (2) 4: <3>; 7: <14>;
Microsoft: (4) 1: <1>; 2: <1,21>; 3: <3>;    5: <16,22,51>;
```

The $/k$ operator, word1 $/k$ word2 finds occurrences of word1 within k words of word2 (on either side) within the same document, where k is a positive integer argument. Thus k = 1 means that word1 should be directly next to word2.

Which comparisons (on document ids or on the positions) are made when querying for `Gates /2 Microsoft`? (any interpretation of /2, including the words or not, is acceptable)

What is the result?

Pen and Paper Task 4 (6 points)

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 “who” and “woh”. 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 postillon.csv.bz2. Unpack this:

```
bunzip2 postillon.csv.bz2
```

Each line corresponds to one Postillon article. The first column is an id, the second column is the original URL, the third column is a date, the fourth column the headline and the fifth column the main text. The columns are tab-separated.

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 should assume that your computer's memory is sufficient to store all postings lists.

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 into an attribute of the 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.

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, show the output of your program for the queries

- `weiß` AND `maße`
- `weiß` AND `masse`
- `weiss` AND `maße`
- `weiss` AND `masse`

This output should at least show the ID and the text of all documents fulfilling the query. Look into why the results that you obtained are shown and explain the differences between the queries.