Practical Assignment 2: Aquisition

- This assignment is due on **November 20th, 2023 (23:59, CET)**
- You can discuss the problems with other groups of this course or browse the Internet to get help. However, copy and paste is cheating.
- You need at least 50% of the points of every assignment to take part in the exam.
- To qualify to sit in exam, you are required to attain a minimum of 70% on all your assignments collectively.
- There are total of 20 assignments, with 13 being theoretical and 7 being practical ones.
- Submit at https://elearn.informatik.uni-kiel.de/course/view.php?id=215
- There will be a competion of all your search engines at the end of the course.
  - only pdf files
  - one file per group per assignment (assignment2.pdf)
  - put your names and matriculation numbers on *each* page in the pdf file

Task 1: Exploratory Data Analysis

While our main focus is on constructing the index for our search engine, it's a good idea to perform some initial data analysis to better understand our data. Let's create programs to accomplish this task. For this analysis, we will be working with text documents from the "collection.tsv" file.

15 P

a) Unique Words Analysis: Calculate the number of unique words or tokens in the whole text and save this information in any data structure. Additionally, create a word cloud to visually represent this information, this will helps understand the vocabulary richness.

b) Word Frequency Analysis: Find the most frequent words in the whole text. You can create a word frequency distribution table or dictionary to see which words appear most often. Additionally, consider using a visualization method that you find most suitable for presenting this information.

c) N-gram Analysis: Plot bar chart that shows how often certain word combinations, like bigrams (pairs of words) and tri-grams (groups of three words), appear. This will help you grasp commonly used phrases and expressions.

d) Text Length Distribution: Plot a density plot to visualize the distribution of each document text lengths and gain insight into how text lengths are spread.

e) Word Categories (Zipf's Law): Categorize the words found in the text from the "collection.tsv" files into different groups based on their types (nouns, verbs, adjectives). After categorization, generate separate Zipfian plots in the form of log-log graphs for each of these word categories.

Task 2: Text similarity and duplication

We may need to identify duplicate text within our collection.tsv file. In this context, we will investigate and apply certain functions that accept two documents as input and provide similarity and duplication scores as output. **10 P**

- Write a function that calculates the Jaccard similarity score, which takes two documents as input parameters.
- Create a function that calculates the Simhash score by taking two documents as input arguments. It's important to use an 8-bit binary representation for each token in this process.

  Hint: You can utilize the md5 hashing method (hashlib library python) to generate hash values for individual tokens, which can then be converted into binary form.

p.s. again, don't do this for all text documents ! Pick only a few ones for testing!

a) Compute the jaccord similarity score among passage id=' 6213662 and '4092752'?

b) Compute the simhash duplication score among passage id=' 7130335' and '8771941'?

Task 3: Preprocessing

Now we continue with our main processing. Given the collection.tsv file contain the text, we now need to process and transform the text to index terms. Design a modular system which allows to easily exchange individual components in a most flexible way, i.e. you should be able to configure your indexing process easily using parameters. The components that you should implement (or use suitable libraries) for preprocessing the text of each document from collection.tsv file. **5 P**

- Tokenizer: split text into tokens and remove special characters.
- Normalizer: down-case all or some tokens.
- Stopword remover: filter out common English stopwords using a stopword list.
- Stemmer: stem the tokens using an existing stemmer from a library.

a) Which stemmer did you use?

b) How did you treat abbreviations?

c) Print the preprocessed text of passage id= 8428575 from collection.tsv file.

d) Print the preprocessed text of the passage id 8771939 from collection.tsv file.