LAB 3: Text processing + Apache OpenNLP

1. Motivation: The text that was derived (e.g., crawling + using Apache Tika) must be processed before being used in an information retrieval system. Text processing usually involves several tasks, such as: tokenization, elimination of stop words, part-of-speech tagging, lemmatization, stemming, or normalization. Some of these tasks are not trivial and have a strong influence on the system's behaviour.

2. Text processing:

a. Tokenization: Tokenization is a process by which the text is divided into single tokens – parts separated by some given set of characters such as spaces or punctuation characters. For example:

Input: Friends, Romans, Countrymen, lend me your years;

Output:

Friends	**Romans**	**Countrymen**	**lend**	**me**	**your**	**ears**

Tokenization is not always a trivial task. Different strategies can generate different tokens, and thus different results in further text processing, such as indexing or querying. For example:

Input: aren't

Possible outputs:

aren't	
aren't	
are	**n't**
aren	**t**

b. Stop words: Stop words are words that do not contain any specific information about the content of the text. These are, e.g., conjunctions or prepositions ("a", "at", "the", "in", etc.). To generate lists of stop-words, one may count the occurrences of the words in documents. It can be assumed that the words that occur frequently in many documents do not yield any important information.

c. Part-of-speech tagging (POS tagging): POS tagging is a process by which tags are assigned to the tokens. These tags inform about a role of a token (word) in a sentence (see https://www.clips.uantwerpen.be/pages/mbsp-tags). POS tagging can be useful for further information retrieval process, e.g., disambiguation:

Cats **like** milk. vs. This cat is white **like** milk.
Like – verb vs. Like – preposition

d. Lemmatization and stemming: **Lemmatization** and **stemming** are processes for finding the base form of a given word.

Stemmers use **heuristics** for finding the base form. As a consequence, the method may fail to find the true base form, and thus the outcome of the stemmer may be invalid. The most popular stemmer for English is Porter Stemmer (see https://tartarus.org/martin/PorterStemmer/def.txt). The examples of the stemming results are presented below.

SSES → SS	caresses →caress
IES → I	ponies →poni ties →ti
SS → SS	caress →caress
S →	cats →cat

Lemmatizers use full-morphological analysis and dictionaries to find the lemma of the word. In practical use, replacing a stemmer by a lemmatizer does not give much gain when it comes to English language, but the results of lemmatizers are significantly better than the results of stemmers for languages with more complex morphology, e.g. Polish, German, or Spanish.

e. Chunking: is a subdivision of sentences into clusters – so-called chunks – according to prosodic patterns and pauses in reading, e.g.:

Input: She put the big knives on the table.

Output:

She	put	the big knives	on	the table
NP	**VP**	**NP**	**PP**	**NP**

NP – noun phrase, VP – verb phrase, PP – preposition phrase.

3. Apache Open NLP: open source Java library for Natural Language Processing (NLP). It supports the most common NLP tasks such as:

- tokenization,
- sentence segmentation,
- part-of-speech tagging,
- named entity extraction,
- chunking,
- parsing,
- language detection,
- coreference resolution,

OpenNLP provides pre-build models for the above tasks. These models are available for several languages, e.g., English, German, Danish, or Spanish. Additionally, OpenNLP provides a machine learning tool for training your own models. Pre-build models can be downloaded from http://opennlp.sourceforge.net/models-1.5/.

Programming assignment (Java, deadline +1 week)

Exercise 1: In this exercise, you are shown how to use OpenNLP for most common NLP applications.

(1) **Library configuration:** Download **openNLP.java**, **opennlp-tools-1.8.3.jar**, and **models.zip** from the lab directory. Create a Java project using any Java IDE. The provided **openNLP.java** file is a template for the following tasks. It contains several methods for text processing (e.g., *private void tokenization()*) which must be finished. Unpack models.zip in the root directory of your project. This archive contains several models that have to be loaded by OpenNLP in order to perform the tasks. Consider the method *run()*. It invokes subsequent methods (languageDetection(), sentenceDetection(), etc.). If you wish to run and verify results of a single step only, just comment out some of the calls.

(2) **Language detection:** OpenNLP provides a tool for language detection. See https://www.apache.org/dist/opennlp/models/langdetect/1.8.3/README.txt for the list of supported languages.

a. As you may notice, a **LanguageDetectorModel** object is created. It uses the langdetec-183.bin model for language detection.
b. Create **LanguageDetectorME** object.
c. Call **predictLanguage** (or **predictLanguages)** method to perform language detection for a given text. This method returns a **Language** object. It contains information about in which language(s) the text is written in along with the confidence value(s).
d. Run this method for the exemplary texts provided in the code. Observe the relation between the predicted language(s) (and confidence values) and the length of the input. How does the probability change if the input text is written in two languages?

(3) **Tokenization:**

a. Use instance of **TokenizerME** and **TokenizerModel** classes. When creating **TokenizerModel** object, load and pass **en-token.bin** model.
b. Use tokenize() method to tokenize input strings provided in the method. In addition, use getTokenProbabilities() to obtain values of probabilities (analogously to confidence values of language detector).
c. Compare the received tokens. Are the outcomes different? Why?
d. Load another model – **de-token.bin** – for German language (but you can try any other model from http://opennlp.sourceforge.net/models-1.5/).
e. Run the method and compare the obtained results with the outcomes that you obtained by **en-token.bin.** Are there any differences?

(4) **Sentence detection:**

   a. Use **SentenceModel** and **SentenceDetectorME** with **en-sent.bin** to detect sentences in a given phrase.
   b. Find a proper method of **SentenceDetectorME** for determining sentences in the provided text and for getting the probabilities of this segmentation.
   c. Call these methods for exemplary texts provided in code and print the results.
   d. Do you see any invalid sentence segmentation? Observe what happens with short sentences like “Hi.”.
   e. Add some punctuation or question marks (or double them e.g., “??”). Run the method. How did it affect the results?

(5) **Part-of-speech tagging**: Part-of-speech (POS) tagging allows detecting part-of-speech for each token in the given sentence. The list of available POS tags, their meaning, and examples, is available at https://www.clips.uantwerpen.be/pages/mbsp-tags.

   a. Use **POSModel** and **POSTaggerME** with **en-pos-maxent.bin** to POS tagging.
   b. Determine part-of-speech for sentences “Cats like milk” and “Cat is white like milk” provided in the exercise code.
   c. Observe the obtained tags. Are these tags correct? Consider the word “like”. Is the tag assigned correctly in both cases?

(6) **Lemmatization and stemming:**

   a. Use **PorterStemmer** and **DictionnaryLemmatizer** classes and **en-lemmatizer.dict** model to find the base form of the tokens given in the exercise.
   b. Compare results of the stemmer and the lemmatizer. Do you see some differences? What happened with the word “are”? Why the lemmatizer uses POS tags?
   c. Check what happens when a given token does not exists in the dictionary (provide a random string). Compare the results of the stemmer and the lemmatizer.

(7) **Chunking:**

   a. Use **ChunkerModel** and **ChunkerME** classes and **en-chunker.bin** model to chunk sentences into clusters. Why the POS tags are required?
   b. Run the method and observe the results. How are the chunks marked (see https://www.clips.uantwerpen.be/pages/mbsp-tags)? What does the “B-” preffix mean? What does the “I-” preffix mean? How many chunks do you see? Do you think that the obtained results are correct?

(8) **Named entity extraction:** The Name Finder can be used to detect named entities and numbers in text. There are different models for different entities. OpenNLP provides

pre-build models for, e.g., finding dates, locations, names of people, or names of organizations. We will focus on finding names of people.

a. Use **TokenNameFinderModel** and **NameFinderME** classes**.** To identify names of people, use **en-ner-person.bin** as the model.
b. Use the above classes to find people in the provided paragraph. Are the results correct?
c. Use **en-ner-xyz.bin** model and run the method again. What kind of entities do you think are found now?

Exercise 2: Given is the collection of 20 movie descriptions (movies.zip). These are text files. Your task is to process these files using OpenNLP and compute/derive the following statistics/features:

<u>For each movie:</u>

- number of sentences,
- number of tokens,
- number of **unique** stems (stemming),
- number of **unique** words (lemmatization),
- list of people, locations, organizations.

<u>Overall statistics (POS tagging):</u>

- percentage number of adverbs,
- percentage number of adjectives,
- percentage number of verbs,
- percentage number of nouns.

(9) Download **MovieReviewStatistics.java** template, which reads data from file, prints, and stores statistics. Your task is to complete the two methods (see TODOs): **initModelsStemmerLemmatizer(),** and **processFile()**. The first method should load required models (for English language!) and create Stemmer and Lemmatizer objects. Use Porter Stemmer. The **processFile()** method should compute/update the statistics.

(10) Before you complete these methods, run the program. You should see that **statistics.txt** file was generated. For each movie, it should contain entries in the following form:

Movie: Some movie
Sentences: 10
Tokens: 100
Stemmed forms (unique): 58
Words from dictionary (unique): 30
People: John, Mr. Smith
Locations: Poland
Organizations: XYZ,

(11) The overall statistics are contained at the end of the file. Obviously, the code is not finished, thus there are no results. All these logs are also printed to the console when executing the program. Now, complete the code. Pay attention to the TODOs and provided (commented) hints.

(12) Run the code. Compare the computed statistics (the sum does not equal 100%) with these for English conversation language from “Longman Grammar of Spoken and Written English”:

Adverbs 0.5%
Adjectives: 2.5%
Verbs: 12.5%
Nouns: 15%

Think about why there is such a difference in adjectives percentage between the obtained results and the results for general English language.

OpenNLP imports:

```
import opennlp.tools.chunker.ChunkerME;
import opennlp.tools.chunker.ChunkerModel;
import opennlp.tools.langdetect.Language;
import opennlp.tools.langdetect.LanguageDetectorME;
import opennlp.tools.langdetect.LanguageDetectorModel;
import opennlp.tools.lemmatizer.DictionaryLemmatizer;
import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.namefind.TokenNameFinderModel;
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;
import opennlp.tools.stemmer.PorterStemmer;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;
import opennlp.tools.util.Span;

import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
```

MovieReviewStatistics imports:

```java
import opennlp.tools.lemmatizer.DictionaryLemmatizer;
import opennlp.tools.namefind.NameFinderME;
import opennlp.tools.namefind.TokenNameFinderModel;
import opennlp.tools.postag.POSModel;
import opennlp.tools.postag.POSTaggerME;
import opennlp.tools.sentdetect.SentenceDetectorME;
import opennlp.tools.sentdetect.SentenceModel;
import opennlp.tools.stemmer.PorterStemmer;
import opennlp.tools.tokenize.TokenizerME;
import opennlp.tools.tokenize.TokenizerModel;
import opennlp.tools.util.Span;

import java.io.File;
import java.io.IOException;
import java.io.PrintStream;
import java.nio.file.Files;
import java.text.DecimalFormat;
import java.util.Arrays;
import java.util.Comparator;
import java.util.HashSet;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;
```