IST 2019/2020

Processamento e Recuperação de Informação

Course Project: Part 1

One important information extraction task relates to the extraction of relevant topical phrases from textual documents, commonly known as **automatic keyphrase extraction**.

For a given document (or for a given document collection), the extracted keyphrases should be relevant to the major topics being described, and the set of keyphrases should provide a good coverage of all the major topics. The fundamental difficulty lies thus in determining which keyphrases are the most relevant and provide the best coverage.

The course project for *Information Processing and Retrieval* will explore different alternatives for addressing the task of automatic keyphrase extraction.

General instructions

For each of the following four exercises that are proposed, you should develop a Python program (i.e., create a file named `exercise-number.py` with the necessary instructions, eventually using implementations for particular algorithms from external libraries) that addresses the described challenges. When delivering the project, you should present a `.zip` file with all four Python programs, together with a PDF document (e.g., created in LaTeX) describing your approaches to address each of the proposed exercises.

The PDF file should have a maximum of two pages and, after the electronic project delivery at Fénix, you should also deliver a printed copy of the source code plus the project report (e.g., using two-sided printing and two pages per sheet).

1 Simple approach based on TF-IDF

Automatic keyphrase extraction is typically a two-step process: first, a set of words and phrases that could convey the topical content of a document are identified through a simple heuristic. Then, these candidates are scored/ranked, so that the *best* candidates (i.e., the top-ranked ones) are selected as a document's keyphrases. As a first exercise, you should implement a simple baseline approach for keyphrase extraction, corresponding to the following ideas:

- Candidate phrases are selected by considering all word tokens (i.e., ignoring stop words, punctuation symbols, and numbers), word bi-grams (i.e., sequences of two consecutive words), and word tri-grams.
- Candidates are scored according to an heuristic based on TF-IDF, where the TF-IDF score of each candidate is multiplied by the candidate's length. You can decide to use either the number of characters or the number of words as the length, and you can consider also a scaling factor in connection to the length parameter.

- The top-5 most relevant candidates are selected as a document's keyphrases.

Your program should apply the keyphrase extraction method to an English textual document of your choice, reading it from a text file stored on the hard drive, and considering character lower-casing as an initial pre-processing step.

For estimating the IDF scores, you can use the set of documents from the 20 newsgroup collection, introduced in the practical classes. Notice that you can use Python libraries such as `scikit-learn`, `spaCy`, or `nltk` in the development of your program.

2 Evaluating the simple approach

The keyphrase extraction method should now be applied to documents from one of the datasets available at `https://github.com/boudinfl/ake-datasets`. Notice that the datasets in the aforementioned URL are already tokenized, and available in a simple XML format that considers different elements for each word token. The standard Python library includes functions for processing XML documents (e.g., see `xml.dom.minidom`).

In this exercise, the IDF scores should be estimated with basis on the entire set of documents from the dataset of your choice, instead of relying on the 20 newsgroups collection. You can decide to use either the original words, or the word lemmas.

Using one of the aforementioned datasets, for which we know what are the most relevant keyphrases that should be associated to each document, your program should print the precision, recall, and F1 scores achieved by the simple extraction method, per individual document and also in terms of the average over the entire collection of test documents. Your program should also print the mean value for the precision@5 evaluation metric, and the mean average precision.

3 Improving the simple approach

Write a Python program that improves on the baseline keyphrase extraction method from Exercise 1 (e.g., in terms of the mean average precision), considering the following two aspects, **as well as other aspects that you deem to be relevant** (i.e., the evaluation for this exercise will value creative solutions, proposed to improve the results for keyphrase extraction).

3.1 Candidate selection

The candidate selection phase should now consider filtering the candidates according to a particular regular pattern involving parts-of-speech tags in connection to the word tokens. The idea is to limit candidates to noun phrases matching a parts-of-speech regular expression pattern like `{(<JJ>* <NN.*>+ <IN>)? <JJ>* <NN.*>+}+`. Parts-of-speech tags for the words in the textual documents can be obtained by the taggers from the `nltk` or `spaCy` packages, or they can be obtained from the XML files in the evaluation collection from the previous exercise.

3.2 Candidate scoring

The candidate scoring phase should now rank candidates according to the BM25 term weighting heuristic, instead of TF-IDF. Recall the BM25 term weighting formula:

$$\text{score}(D,t) = \log\left(\frac{N - n(t) + 0.5}{n(t) + 0.5}\right) \cdot \frac{f(t,D)\cdot(k_1+1)}{f(t,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

In the previous equations, $f(t, D)$ is the frequency for term t in document D, $|D|$ is the length of document D in terms of the number of words, and $avgdl$ is the average document length in a background text collection. Both k_1 and b are free parameters, usually set to $k_1 = 1.2$ and $b = 0.75$. In the IDF part of the formula, N is the total number of documents in a background collection, and $n(t)$ is the number of documents, from this background, containing the term t.

NOTE: The results of the improved approach should be comparatively evaluated against the baseline method from the first two exercises. Other ideas that you can consider testing involve (i) experiment with BM25F instead of BM25, using the first sentences as a separate field, (ii) combining BM25 with the length of the candidate keyphrases, or (iii) using named entity recognition or other IE techniques to improve the candidate selection.

4 A supervised approach

Keyphrase extraction can also be formulated as a supervised learning problem. In this case, candidates for a given document can be represented through a set of descriptive features (e.g., through different ranking scores), and training data can be used to learn an optimal combination of the features for the purpose of selecting relevant keyphrases.

For this exercise, you should implement one such supervised approach for keyphrase extraction, leveraging a classification algorithm such as the Perceptron (i.e., the classifier should assign a binary decision to each candidate, depending on whether it corresponds to a keyphrase of interest). The following features can be considered in the ranking model:

- Position of the candidate in the document;
- Length of the candidate.
- Term frequency, inverse document frequency, TF-IDF, or BM25 scores for the candidate;

After training, the inferred model can be used to score candidates according to the probability/confidence assigned to the positive class, and the highest scoring candidates should be returned as the keyphrases.

You should use the same dataset from the previous exercise for evaluating the quality of the obtained results (e.g., through a metric such as the mean average precision), and your implementation can rely on packages such as `scikit-learn`.

The evaluation for this exercise will also value creative solutions for improving the results (e.g., solutions involving different sets of features, or involving different classification methods).