IST 2016

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 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, you should develop a Python program (i.e., create a file named `exercise-number.py` with the necessary instructions) 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 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. 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 an approach for keyphrase extraction where:

- Candidates are selected by considering all words and word bi-grams (i.e., sequences of two consecutive words), after removing stop words and punctuation.
- Candidates are scored according to the TF-IDF heuristic, and the top-5 candidates are selected as a document's keyphrases.

You 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.

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` 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/zelandiya/keyword-extraction-datasets`.

In this case, the IDF scores should be estimated with basis on the entire collection of documents, in the dataset of your choice, instead of relying on the 20 newsgroups collection.

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. Your program should also print the mean average precision.

3 Improving the simple approach

Write a Python program that improves on the keyphrase extraction method from Exercise 1, 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 longer word n-grams (where $1 \leq n \leq 3$), afterwards filtering the candidates according to a particular regular pattern involving parts-of-speech tags. 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 tagger from the `nltk` package, which was introduced in the lab classes.

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:

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

$$\text{IDF}(t) = \log \frac{N - n(t) + 0.5}{n(t) + 0.5}$$

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.

The parameters k_1 and b are free parameters, usually set to $k_1 = 1.2$ and $b = 0.75$.

In the IDF 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.

4 A more sophisticated approach

In a paper entitled *A language model approach to keyphrase extraction*[1], Tomokiyo and Hurst proposed an approach for keyphrase extraction (i.e., an approach for ranking candidates) that leverages probabilistic language models, similar to the naive Bayes method and to the probabilistic retrieval models that were introduced in the classes. You should now develop a Python program that implements the method advanced by Tomokiyo and Hurst.

The method ranks candidates with basis on a combination of two scores, namely **phraseness** (i.e., the degree to which a given word sequence is considered to be a real common phrase, instead of a random sequence of words) and **informativeness** (i.e., how well a phrase captures or illustrates the key ideas in a document). Both phraseness and informativeness can be estimated with probabilistic language models.

In general, a probabilistic language model assigns a probability value to every sequence of words $W =< w_1, w_2, \ldots, w_m >$. Assuming each word w_i depends on the previous n words, n-gram language models are commonly used. In the general case, the probability computation for an n-gram language model is as follows:

$$\begin{aligned} P(W) &= \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \\ &\approx \prod_{i=1}^{m} P(w_i \mid w_{i-(N-1)}, \ldots, w_{i-1}) \\ &\approx \frac{\text{count}(w_{i-(N-1)}, \ldots, w_{i-1}, w_i)}{\text{count}(w_{i-(N-1)}, \ldots, w_{i-1})} \\ &\approx \frac{\text{count}(w_{i-(N-1)}^{i-1}, w_i)}{\text{count}(w_{i-(N-1)}^{i-1})}. \end{aligned} \tag{1}$$

The simplest case is a unigram language model, where each word is independent of others.

$$P(W) \approx \prod_{i=1}^{m} P(w_i). \tag{2}$$

[1] `http://dl.acm.org/citation.cfm?id=1119287`

Suppose we have access to a foreground document, from where we want to extract keyphrases, and a background corpus which provides a general description for the domain under analysis. The backgroud corpus can be used to measure phraseness, and the relationship between the foreground and background can be used to formalize the notion of informativeness.

Tomokiyo and Hurst proposed to use the pointwise Kullback-Leibler divergence $\delta_W(P||Q)$, between probabilities $P(W)$ and $Q(W)$ given by language models, to compute phraseness and informativeness:

$$\delta_W(P||Q) = P(W) \times \log\left(\frac{P(W)}{Q(W)}\right). \tag{3}$$

Let LM^1_{bg} denote a unigram language model computed from the background corpus, LM^1_{fg} denote a unigram language model computed from the foreground document, LM^n_{bg} denote an n-gram language model computed from the background corpus, and LM^n_{fg} denote an n-gram language model computed from the foreground document. With these language models, the phraseness of a keyphrase W can be computed as follows:

$$\delta_W(LM^n_{bg}||LM^1_{bg}). \tag{4}$$

The informativeness of a keyphrase W can, in turn, be computed as follows:

$$\delta_W(LM^1_{fg}||LM^1_{bg}). \tag{5}$$

Leveraging the aforementioned ideas, your program should select n-grams as candidate keyphrases, and then rank keyphrases according to a linear combination of phraseness and informativeness (e.g., computed by adding both scores). Your program should apply this new keyphrase extraction procedure to one of the document collections from Exercise 2. Recall that the probabilities associated to the language models (e.g., word bi-gram language models) can be estimated from document occurrences, together with a smoothing technique.