Academic Year 2020/2021

Prospeção e Análise de Dados

Project 2

Bacchiocchi Francesco: f.bacchiocchi@campus.fct.unl.pt
Castelli Daniele: d.castelli@campus.fct.unl.pt

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

June 17, 2021

The Algorithm:

The Local Max algorithm considers the whole corpus and produces as output a set of Relevant expressions. In particular, the algorithm defines and extracts the relevant expressions in the following way:

- An antecedent of a hole free n-gram ($w_1 \ldots w_n$) is a hole free (n-1)-gram of the kind: ($w_1 \ldots w_{n-1}$) or ($w_2 \ldots w_n$).
- A successor of a hole free n-gram ($w_1 \ldots w_n$) is an antecedent of a (n+1)-gram of containing ($w_1 \ldots w_n$)

W= ($w_1 \ldots w_n$) is a RE if:

g(W)>g(ant(W)) and g(W)>g(succ(W)) for all ant(W) and succ(W) (if W's size>2)

g(W)>g(succ(W)) for all succ(W) (if W's size=2)

where g(.) is a glue function.

Glues Functions:

To define the cohesion between two different words we can introduce a set of glues functions. We will start presenting them for two different words and then we will extend them to a general n-grams relying on the so-called pseudo-bigram transformation.

- The **DICE** function between two unigram x, y is defined as:

$$Dice(x,y) = \frac{2f(x_, y)}{f(x)+f(y)}$$

  Where f(x), f(y) and f(x, y) are the frequencies of the unigrams x, y and the bigram (x, y) respectively. The generalization for n-grams is defined as:

$$Dice\_f(w_1 \ldots w_n) = \frac{f(w_1 \ldots w_n)}{F}$$

$$F = \frac{1}{n-1}\sum_{i=1}^{n-1} f(w_1 \cdots w_i) + f(w_{i+1} \ldots w_n)$$

- The **SCP** (Symmetrical Conditional Probability) function between two unigram x, y is defined as defined as:

$$SCP(x,y) = p(x|y) = p(y|x) = \frac{p(x_, y)}{p(x)\,p(y)}\frac{p(x_, y)}{p(x)p(y)}$$

  Where p(x), p(y) and p(x, y) are the probabilities of the unigrams x, y and of the bigram (x, y) respectively. The generalization for n-grams is defined as:

$$SCP\_f((w_1 \ldots w_n)) = \frac{p((w_1 \ldots w_n))}{F}$$

$$F = \frac{1}{n-1}\sum_{i=1}^{n-1} p(w_1 \cdots w_i)p(w_{i+1} \ldots w_n)$$

- The **MI** function between two unigram x, y is defined as defined as:

$$Mi(x,y) = log \frac{p(x,y)}{p(x)\,p(y)}$$

Where p(x), p(y) and p(x, y) are the probabilities of the unigrams x, y and of the bigram (x, y) respectively. The generalization for n-grams is defined as:

$$MI\_f((w_1 \ldots w_n)) = \log \frac{p((w_1 \ldots w_n))}{F}$$

$$F = \frac{1}{n-1} \sum_{i=1}^{n-1} p(w_1 \cdots w_i) p(w_{i+1} \ldots w_n)$$

Note that in the literature we can find many other examples of glues function, we have chosen to present these three since they are the ones we have implemented in our extractor of REs.

Results:

We have analysed two different corpora having 2 million words each. We have applied to each one the Local Max algorithm filtering the obtained relevant expressions following three criteria:

- All the REs of two words containing stopping words such as: "in", "as", "for", "to", "it", "if", "away", "behind", "by" have been deleted like: “Such as”, “for it”, “to do”, “if it” etc…
- All the REs of two words containing stopping characters such as: “,”, “.”, “;”, “:”, “!”, “?”, “^”, etc…
- All REs with frequency less than two

The results on the first corpus are given by:

GLUE	PRECISION	RECALL	F1-SCORE	NUMBER OF RES (FILTERED)
DICE	49%	46%	47%	35177
SCP	72%	42%	53%	16995
MI	64%	36%	46%	13299

For each glue function, to compute the precision, we have randomly selected 100 REs and we have evaluated the percentage of actual REs. To compute the recall, we have selected 100 random REs from the documents, and we have computed how many of these were present in the REs predicted by the algorithm. Since it was infeasible to check the filters manually, we looked for the RE’s in the unfiltered list. Finally, to compute the F1 score we relied on the following formula:

$$\mathrm{F1-score} = \frac{2 \cdot \mathrm{Precison} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

The results on the second corpus are given by:

GLUE	PRECISION	NUMBER OF RES (FILTERED)
DICE	61%	51224
SCP	76%	38910
MI	66%	30361

After observing unexpected poor results in the estimate of the recall for the first corpus we guessed that a possible problem could be the distinction between lower- and upper-case characters. For this reason, we did not compute the recall for the second corpus. A possible improvement could be to modify the original corpus to have only lower-case characters.

Explicit Keywords:

The explicit keywords can be considered as the most "representative" multiword describing a document. To find them for each document we have ranked separately the filtered REs (obtained with each one of the previous presented glues) and the unigrams by the Tf-Idf metric defined for multigram as:

$$Tf - Idf\left(Re\ d_j\right) = \frac{f\left(Re\ d_j\right)}{Size(dj)} \cdot log\left(\frac{|\ D\ |}{|d \in D: f(Re, d) > 0|}\right)$$

While for unigrams as:

$$Tf - Idf\left(W\ d_j\right) = \frac{f\left(W\ d_j\right)}{Size(dj)} \cdot log\left(\frac{|\ D\ |}{|d \in D: f(W, d) > 0|}\right)$$

Where we have defined:

- $Size(dj)$: number of words in document j
- $f\left(Re\ d_j\right)$: frequencies of Re in document j
- |D|: number of documents of the corpus

We have then kept the first five ranked unigrams and the first ten ranked REs for each document and considered them as explicit keywords.

Results:

We report some examples of Explicit Keywords (obtained with SCP as glue) for a few of the analysed documents in the first corpus. The correspondence between the numeration of documents here reported and the names of the text files can be found in the text file doc order.

Doc6:	*Kiriella, election, candidate, total vote, Ceylonese politician, 20 september 1947*
Doc57:	*Gerusalem, archeological, west bound, Holy city*
Doc214:	*Mascots, Asian, tournament, Afc asian cup, 2007 edition*
Doc316:	*President, Vice-President, Controversies, Succession, Pro tempore*

We now report the analogous table using the MI glue on the second corpus:

Doc1:	*Montenegro, Podgorica, government of Montenegro, Motorwale, route description*
Doc6:	*Adelaide, Windsor, Cottage, Re-erected, furnishings*
Doc167:	*Michelangelo, national Gallery, canvas painting*
Doc357:	*pollution, chemistry, center for atmospheric science, volatile organic compounds*

Implicit Keywords:

Finally, we have detected the Implicit keywords, which are those relevant expression not necessarily present in a certain document but related to its contest. Before presenting the algorithm to find them we must introduce the concept of correlation between two RE A and B:

$$cor(A,B) = \frac{cov(A,B)}{\sqrt{cov(A,A)}\sqrt{cov(B,B)}}$$

Where:

$cov(A,B) = \frac{1}{|D|-1}\sum_{i=1}^{|D|}(p(A,d_i) - p(A,.))\,(p(B,d_i) - p(B,.))$ and $p(A,.) = \frac{1}{|D|}\sum_{i=1}^{|D|} p(A,d_i)$

Some examples correlations of RE in the first corpus using SCP as glue in the extractor are given by:

- Cor (“Phoenix Suns”, “NBA Finals”) = 0.88
- Cor (“literary criticism”, “Garden of Eden”) = 0.44
- Cor (“Washington DC”, “Monetary Fund”) = 0.57
- Cor (“memorial plaque”, “King Charles III”) = 0.71

As we can see positive correlated RE present a semantic similarity. Finally, to compute the implicit keywords we have computed for each document d and for each RE not presented in d the following score:

$$Score(RE,d_i) = \sum_{i=1}^{|D|} cor(RE,k_i)$$

Where k_i is the i-th ranked explicit keyword of d. In this way for each document, we were able to rank the REs having a higher score and defining the first ten as implicit keywords.

Results:

We report some examples of Implicit Keywords (obtained with SCP as glue) for a few of the analysed documents in the first corpus. The correspondence between the numeration of documents here reported and the names of the text files can be found in the text file doc order.

doc85:	*golden ratio, volumetric capacity*
doc86:	*bronze medal at the 1987 Pan, Racial Discrimination, Legislation Amendment*
doc158:	*school's motto, surrounding areas, makeup of the city*
doc181:	*American football linebacker, Recreation Area, Jacksonville Jaguars*

As we experienced the algorithm we implemented is not very efficient to find Implicit Keywords and is also very slow (several hours). A possible improvement would have been to use the SemiProx index (Which we implemented and left as a comment in the code) but it would have required too much time to run, being the correlation alone already very costly.