University of Padua

Department of Information Engineering
Department of Mathematics

Information Retrieval

Class Project

Marco Romanelli, [1106706]
Federico Ghirardelli, [1130426]

February 12, 2017

Chapter 1: Introduction

1.1 Purpose of the project

This project is composed by two main goals:

- Develop a statistical stemmer following the proposed paper;
- Evaluate the stemmer using the CLEF Ad-Hoc experimental collection for Italian, German and Russian.

The remainder of this paper proceeds as follows. Chapter 2 provides a short description of the GRAS stemmer and continue with the implementation, complexity and some issues found. Chapter 3 provides the results of the evaluation runs. Chapter 4 highlights future works and team's conclusions. References section enumerates bibliographic references and the code source links.

Chapter 2: GRAS

2.1 Overview

The GRAS stemmer [Paik et al., 2011] is a graph-based, lexicon analysis-based stemmer and is intended to be retrieval effective, general (language independent) and with low computational cost. Given a set of distinct words the goal is to partition them into classes where each class represents a set of morphological relates words.

The proposed solution consists of two main algorithms:

1. *Suffix pair identification.* The input words (lexicon) are partitioned into groups such that each pair of words has a longest common prefix of length at least a given threshold value l. Then, for each word class the suffix pair are derived from each pair of words and the frequencies of suffix pair are computed. Finally, the set of α-frequent suffix pair is constructed.
2. *Class formation.* The algorithm looks for pairs of words that are morphological related, that is, if they share a non-empty common prefix and the suffix pair is a valid candidate suffix pair. These words relationships is modeled using a graph where nodes represent words and edges are used to connect related words. The node which is connected by edges to a large number of other nodes, called pivot node, is identified. A word that is connected to a pivot is put in the same class as the pivot if it shares many common neighbors with the pivot. Once such words classes are formed, stemming is done by mapping all the words in a class to the pivot for that class.

2.2 Implementation

Initially, we choose to use Python language with the python-igraph [igraph core team, 2017] library. We opt for python mostly because it's easy and quick to write. The graph library was fast enough to deal even with thousand of nodes (and edges).

Issues Issues arose during the suffix pair identification algorithm and specifically while looping throw pair of words. Clearly, this peace of code theoretically takes quadratic time over the class dimensions. Moreover, we have to redo this step in order to create the graph edges, since two nodes are connect if there's an α-suffix pair induced by the word pair.

So first we tried to use some extra data structures and backup values in the first piece of code in order to speed-up the second part. However, we noticed that the python code do not scaled up well and therefore the memory required for by process increased a lot, up to a few gigabytes.

At this point, we switched to the idea that a higher running time was more desirable than an higher memory consumption. So we tried this new approach optimizing the python code to use less memory (like using itertools module and partial frequencies counting). Then we run the algorithm but we saw that it takes more time than we expected, way too much, even hours for a moderate lexicon.

Solution Hence we sadly decide to switch to another programming language and we choose Java[1]. We implemented the solution step-by-step, without any code optimization or extra data structures. Even in this conditions the code performed well, stemming the Italian and Russian lexicons in about ten-twenty minutes. Problems presented again stemming the German lexicon, the bigger one, that contains more than one million words. Since we didn't have much time for further optimizations, we decided to stem a smaller portion of the lexicon. Hence, we run GRAS only over words extracted from documents in SDA1994.

The code (both Java and Python) can be found in the public online repository of the project [Romanelli and Ghirardelli, 2017].

2.3 Parameter setting

The parameter setting is an important point of the proposed stemmer. It provide a way to tune the stemming procedure based on which language we are working on. Different languages may require different values of the three parameters in the algorithm. For the first parameters, l, we followed the suggestions of the paper's authors and we set it to the average word length, computing it for each language while parsing the lexicon. For the other two, which are α and δ, we run several experiments with different values for each language, like the authors of GRAS did in their paper, and we found that the best value for $alpha$ was 2. However, such a low value brings an high time computational cost so we decided, as a balancing act, to use the second-best value, which was 8 for both German and Russian and 6 for Italian. The changes of the δ value seemed to bring no significant improvements so we followed the paper and set it to 0.8.

The table 2.1 in the next page sums up those values:

2.4 Performances

Below we present time and memory consumptions for compute the lookup list (word-stem file) using the Java version of GRAS. These values were obtained setting the algorithm with parameters setting discussed before. Different values will cause a very different results.

[1]In the java version we used the JGraphT library[Naveh and Contributors, 2017]

Table 2.1: Parameter setting

	Italian	German	Russian
l	8	11	10
α	6	8	8
δ	0,8	0,8	0,8

Decreasing the l-value will generate less word classes with greater dimensions. Since the suffix pair identification algorithm is quadratic over the size of those classes, this will increase the time spent dramatically. A lower α-value will increase the α-frequent suffix pair set and therefore the number of edges in the graph.

Table 2.2: Time and memory consumptions

	Italian	German (SDA1994)	Russian
Time	16 : 42.40	21 : 25.82	13 : 00.88
Memory	537′500	512′552	525′556
$\|V\|$	357′904	373′156	377′166
$\|E\|$	516′066	119′812	204′885

We run GRAS on a 64-bit linux machine (Ubuntu 16.04 LTS) with 4 GiB of RAM, a 3.3 GHz i5-2500 Intel Core and we observed the following parameters:

- time: elapsed real (wall clock) time used by the process [minutes:seconds.];
- memory: maximum resident set (RSS) size of the process during its lifetime [Kilobytes];
- $|V|$: number of vertexes created by the algorithm, egual to the number of unique words in the corpus;
- $|E|$: number of edges for the graph.

Chapter 3: Evaluation

3.1 Collections

The collections provided are part of the CLEF initiative for Ad-Hoc task for Italian, German and Russian languages. Below we summarize some information:

Table 3.1: Statistics on Test Corpora

	Italian	German (SDA1994)	Russian
No. of documents	157′558	225′371	16′716
No. of unique words	357′904	1′026′017(373′156)	377′166
No. of queries	51	56	28

3.2 Runs

We now present the retrieval performance of the proposed solution compared with a language-dependent version of Porter stemmer [Porter, 2001], called Snowball. For each language, we present two visual results: one table showing MAP value, prec@10 and prec@30 and a recall-precision curve.

We used several retrieval strategies in order two compare GRAS with Snowball:

- Snowball: ItalianSnowball, GermanSnowball and RussianSnowball with/without stop word removal;
- GRAS: the proposed with/without stop word removal.

Besides, we used two different stoplist in order two compare their effectives in different scenarios.

Italian We run the first set of experiments on Italian because we have more familiarity with the language. GRAS with the second stoplist performed better than the other methods both in term of MAP and precision.

Table 3.2: Retrieval results for Italian

Pipeline	RUN	MAP	P@10	P@30
stoplist, Snowball	1	$0,0795$	$0,049$	$0,0418$
stoplist2, Snowball	2	$0,0792$	$0,0451$	$0,0399$
stoplist, GRAS	3	$0,2987$	$0,2314$	$0,1431$
stoplist2, GRAS	4	$\mathbf{0,3079}$	$\mathbf{0,2373}$	$\mathbf{0,1516}$
Snowball	5	$0,0823$	$0,049$	$0,0444$
GRAS	6	$0,3075$	$0,2529$	$0,1503$

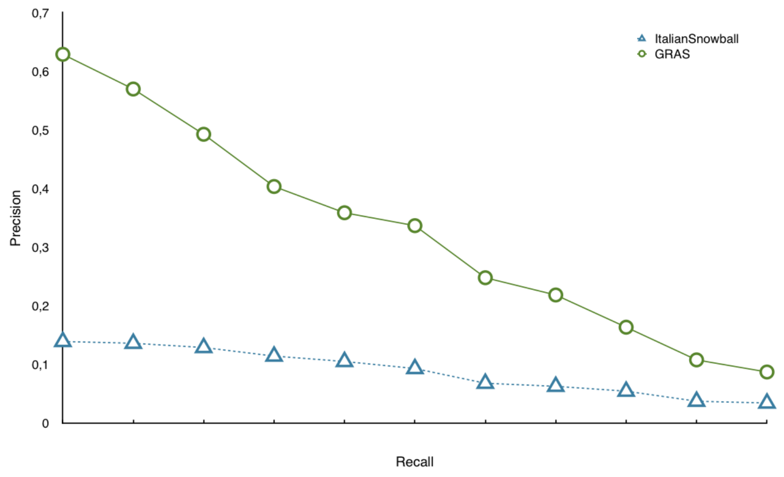

Figure 3.1: Precision recall curve for Italian

Russian Once again, GRASS is the top performer both in MAP and prec@10, but there's a difference between the run with the two stoplist. The run with the second stoplist performed better at prec@30 rather then using the first stoplist.

Table 3.3: Retrieval results for Russian

Pipeline	RUN	MAP	P@10	P@30
stoplist, Snowball	1	0, 1381	0, 0857	0, 0405
stoplist2, Snowball	2	0, 1362	0, 0893	0, 0405
stoplist, GRAS	3	**0, 3549**	**0, 2071**	0, 100
stoplist2, GRAS	4	0, 3515	0, 2071	**0, 1024**
Snowball	5	0, 1386	0, 0857	0, 0381
GRAS	6	0, 3424	0, 2036	0, 0976

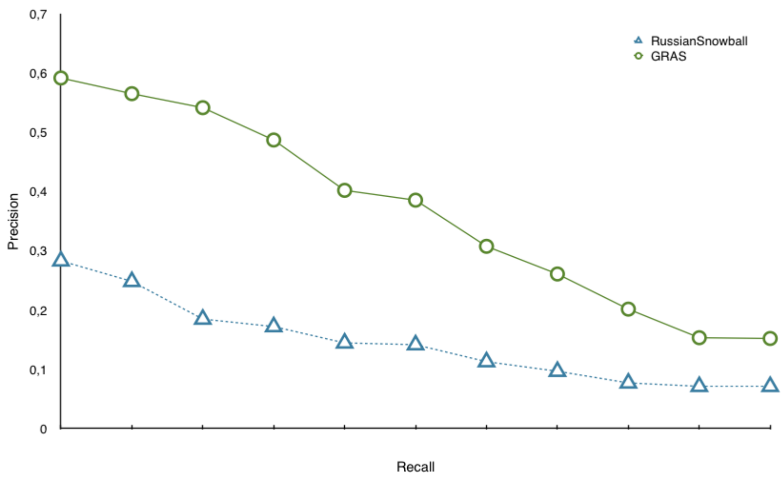

Figure 3.2: Precision recall curve for Russian

German As explained earlier, due to the large amount of words in the german corpus we weren't able to stem the entire lexicon. The lexicon taken for indexing is only a portion of the total. Like in the runs with Italian, the best performance is found with GRASS with the second stoplist. In this case, however, the precision and recall follow a similar trend for both stemmers concerned.

Table 3.4: Retrieval results for German

Pipeline	RUN	MAP	P@10	P@30
stoplist, Snowball	1	0, 1383	0, 2107	0, 1375
stoplist2, Snowball	2	0, 1358	0, 2089	0, 1381
stoplist, GRAS	3	0, 1902	0, 2857	0, 1762
stoplist2, GRAS	4	**0, 1923**	**0, 2929**	**0, 1863**
Snowball	5	0, 1349	0, 2054	0, 1357
GRAS	6	0, 1903	0, 2964	0, 1863

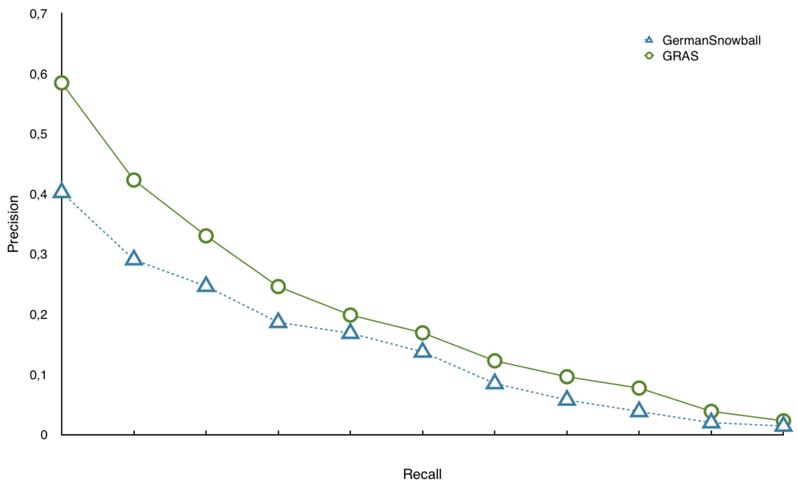

Figure 3.3: Precision recall curve for German

Chapter 4: Conclusion

4.1 Future works

One weak points of the whole procedure is the grouping of words that sharing common prefix. To improve the complexity, Trie data structure can be used to reduce the execution time. Tries, or radix-tree, are kind of search tree that allows words with similar character prefixes to use the same prefix data and store the rest of the word in the form of tails or child nodes as a separate data . One character of the string is stored at each level of the tree, with first character of the string stored at the root.

4.2 Closure

In conclusion, we've seen the challenging behind the developing of a statistical stemmer and the differences between the theoretical approach and the actual implementation (the development of the code). We've better understood how the evaluation of a retrieval system works and its complexity; how different component interact with each others and how this interaction affects results. This project drove us into retrieval systems word and helped us to understand the course subject and its goals.

References

[igraph core team, 2017] igraph core team, T. (2015 (accessed February 8, 2017)). python-igraph library. http://igraph.org.

[Naveh and Contributors, 2017] Naveh, B. and Contributors (2017). JGraphT library. http://jgrapht.org.

[Paik et al., 2011] Paik, J. H., Mitra, M., Parui, S. K., and Järvelin, K. (2011). GRAS: an effective and efficient stemming algorithm for information retrieval. *ACM Trans. Inf. Syst.*, 29(4):19:1–19:24.

[Porter, 2001] Porter, M. F. (2001). Snowball: A language for stemming algorithms. Published online. Accessed 11.03.2008, 15.00h.

[Romanelli and Ghirardelli, 2017] Romanelli, M. and Ghirardelli, F. (2017). Information retrieval project repository. https://bitbucket.org/stem-ir-2016-2017/ir-romanelli-ghirardelli.