A Survey of Freely Available Polish Stemmers and Evaluation of Their Applicability in Information Retrieval

Dawid Weiss

Poznań University of Technology
Piotrowo 3a, 60-965 Poznań, Poland
dawid.weiss@cs.put.poznan.pl

Abstract

Stemmers are computer programs for transforming all inflected forms of a word into a token representing its broader meaning. There are many openly available stemmers for English and other Indo-European languages, but until very recently stemmers for Polish were mostly commercial. In this paper we provide a survey of free and open source stemmers for Polish that came into view in the period of the last two years. We additionally present results from an experiment comparing these stemmers with two fully-fledged morphological analyzers.

1. Introduction

1.1. Background

In many languages, especially Indo-European, the sequence of letters forming a word may change without altering its primary meaning (due to inflection or other grammatical needs). We can talk about a *word* (or *lemma*) as a representative of a broader meaning expressed by many *word forms*.[1] This phenomenon causes significant difficulties for designers of computer algorithms where text is processed as a raw stream of symbols (word forms) and deeper contextual analysis of meaning is often prohibitive.

The process of finding a lemma (or many lemmas if there is ambiguity) for a given word form is called *lemmatization*. In many practical applications, however, full lemmatization is not necessary – it is enough that a unique token of any kind is found for all word forms of a single lemma; this method is commonly referred to as *stemming*.

Stemming is a very popular tool for quantitative text processing in Information Retrieval and Data Mining because it often brings an improvement to the quality of results and decrease in storage requirements of the semi-processed information.[2] The following things are expected from a stemming algorithm:

- for any given word form, find a unique sequence of characters, called a *stem*, representing the lemma this word form belongs to;
- stems should be unambiguous, that is one stem should represent one lemma.

Stemming algorithms are usually categorized between dictionary-based and rule-based. Dictionary-based stemmers always find correct lemmas for word forms present in their lookup dictionary (hence the name). Their strength is also their weakness as any word outside of their dictionary is not stemmed even if it follows a regular pattern of inflection. Rule-based stemmers are the contrary: they use *rules* (heuristic or manually created) to *transcode* the inflected form of a word into its stem. An example of such a rule in English could be to remove last three letters from words ending with a suffix *–ing* (e.g. *working→work*).

1.2. Stemmers for Polish

Polish is among the group of languages with highly developed system of inflectional rules, making word alternations very common. It was shown for example in (Stefanowski and Weiss, 2003) that applying even a simple stemming method to Polish yields significant improvement in an information retrieval application. Only a few stemmers dedicated to the Polish language were available until very recently; most of them commercially.

Probably the first free Polish morphological analyzer – SAM-95 – was created by Krzysztof Szafran (Szafran, 1996). SAM-95 is a hybrid stemmer that uses an a-tergo dictionary of suffixes of inflected forms, collected by Jan Tokarski. Potential stems are then generated and (and in an optional step) verified against a regular dictionary of known words.

Chronologically next is the Finite State Automaton (FSA) package (Daciuk, 1998), written by Jan Daciuk. FSA contains all the code needed to build an automaton-based stemmer (not only for Polish) and its author provides a small automaton sample built on a corpora of texts from a local newspaper.

Recently, in a relatively short period of time, a handful of open source or freely available heuristic and dictionary-based stemmers came into view: LAMETYZATOR (Weiss and Stefanowski, 2003), REG (Wąchnicka, 2004), STEMPEL (Białecki, 2004) and WASPELL (Płotnicki, 2003). We give a more detailed description of these stemmers in the next section.

A reader interested in the commercial stemmers of Polish will find their excellent survey in (Hajnicz and Kupść, 2001). We only focus on the freely available projects.

1.3. Goal and scope of this work

In this paper we provide a short survey of the freely available or open source stemmers for Polish along with

[1] A term *lexeme* is also often coined. Lexemes additionally include terms that form a distinct meaning only when grouped together, as in a phrasal verb 'take off' for instance.

[2] It must be honestly said that negative aspects of using stemming have also been reported.

description of methods used to create them. We then present outcomes of an experiment where we compared the results achieved by all free stemmers and two fully-fledged morphological analyzers – FORMAN from TiP Sp. z o.o. and MORFEUSZ from IPI PAN – on samples of real data.

It must be clearly stated here that we focus particularly on *quantitative* aspects of stemming, that is: which stemmer is better on the average. Full linguistic correctness is largely neglected. It was the experiment's primary focus and we we simply do not feel to be the experts in the domain.

2. A survey of free stemmers for Polish

2.1. FSA and SAM-95

Briefly outlined in Section 1.2., both FSA and SAM-95 suffer from certain limitations that make them quite difficult for an easy integration with other software. SAM-95 is provided as a precompiled binary and the batch processing mode it supports is unacceptable for on-line processing of large amounts of text. FSA, on the other hand, does not include an automaton created on a broad corpora so a fair amount of work is put on the shoulders of the final user of the package (stemming is not FSA's primary focus, we do not complain). We decided to omit the two packages in our experiment. The reality is, however, that traces of both projects can be found in the background of other stemmers. For example, LAMETYZATOR uses Jan Daciuk's FSA package to compile and traverse its data structure. SAM-95 in turn, was used to pre-analyze some words that served as a knowledge base for STEMPEL and REG.

2.2. LAMETYZATOR

Chronologically first, LAMETYZATOR (Weiss, 2003) is a dictionary-based stemmer written by Dawid Weiss and available in public domain (no license).Internally, LAMETYZATOR uses data generated from inflection rules and stems available in the *ispell-pl* project (Ispell-pl, 2002). All thus acquired pairs (inflected form – stem) are compressed into a finite state automaton (using FSA package) which makes the stemming process very fast and the dictionary relatively compact (about 1.6MB). LAMETYZATOR is a dictionary-based stemmer so for all word forms it has in its database, correct stems are returned (there can be more than one because ambiguities present in the input are preserved). To decrease the dictionary's size identical transformations (stem→stem) are not stored. By default, if no entry is found for a given word form, its verbatim copy is returned as its stem.

2.3. STEMPEL

STEMPEL is an example of a rule-based approach. It was written by Andrzej Białecki and is available under Apache-style license.[3] STEMPEL's rules for transforming a word form into its stem are stored as a dictionary of *patch commands*: sequences of atomic commands modifying the inflected word by addition, removal or substitution of letters until finally the stem is obtained. The patch commands are automatically learned from examples and form extrapolation rules hopefully applicable to word forms not seen during the learning process. The size of the learning set obviously affects the quality of stemming; one expects the-more-the-better relationship and this seems to be confirmed by our experiments.

Free version of STEMPEL comes with dictionaries generated by the author for tables up to 5000 inflected lemmas. Tables larger than that are available commercially, but we include them in this experiment for the sake of completeness.

It should also be pointed out that STEMPEL heavily relies on the previous work on patch-commands in stemming by Leo Galambos (Galambos, 2004) (done as part of the Egothor project).

2.4. REG

REG (Wąchnicka, 2004) is definitely among the most experimental stemmers. It is a master thesis written by Justyna Wąchnicka and again works on the principle of learning transformation rules from training data. In contrast with STEMPEL, the transformation rules are much simpler and specify the required suffix of a word and its entire replacement (number of truncated characters and a string to append). The rules were induced using an implementation of LEM2 rule induction algorithm (Grzymała-Busse, 1992). Unlike in STEMPEL where patch commands trie is optimized, rules in REG are stored in verbose, human-readable format.

Two sets of rules are available with the program (computed for two different training corpora). The stemmer works by matching attribute part of rules (in other words, suffixes) against the word form and choosing a rule of transformation that best matches the input.

The implementation of this stemmer is among the weakest and is very slow (full scan through all rules for each stem lookup). The author claims, however, that the performance shown on unknown words, especially proper names, beats other known free stemming algorithms for Polish.

REG is available by request from the author, its license could not be determined.

2.5. WASPELL

WASPELL (Płotnicki, 2003) is a project and master thesis written by Zbigniew Płotnicki (under a different supervisor than REG, although within the same institution). The stemming algorithm is again dictionary-based and uses ispell's data, enriched with morphological information acquired from parsing the input with SAM-95. The stemming pairs are compiled into a self-designed, compact graph structure (acyclic finite state automaton) which looks vaguely similar to Jan Daciuk's FSA, but involve many low-level optimizations at data structure level. The resulting data structure has an impressive compression ratio and includes morphological and stemming information.

WASPELL is available under LGPL license but, at the time of writing, only as a precompiled library.

[3] `http://getopt.org/stempel/`

2.6. STEMPELATOR

This is an ad-hoc stemmer created by the author of this paper for the needs of the experiment by merging LAMETYZATOR and STEMPEL. Since the source code to both is freely available we thought it would be interesting to see how a 'hybrid' approach of dictionary and rule-based stemmers would perform. The combination is quite simple: if a stem for a given word is not found in LAMETYZATOR, it is created using STEMPEL.

2.7. Other stemmers

We decided to include two fully-fledged morphological analyzers in our experiment. They are by no means free, but they give us a point of reference to compare to. FORMAN is a morphological analyzer from TiP Sp. z o.o. It is widely used because it is embedded in a popular word processor. MORFEUSZ, from Instytut Podstaw Informatyki PAN, is a morphological analyzer that is supposedly still at an early stage of development, but already established its strong position among Polish computational linguists.

3. Experimental comparison of stemmers

3.1. Evaluation procedure

The most important aspect of a good stemmer for Information Retrieval is its ability to isolate a unique stem for all word forms of a lemma. We collected groups of inflected forms that we expected to be conflated to a single stem and processed them using all of the available stemmers. Because STEMPEL heavily relies on the dictionary it uses, we repeated the evaluation procedure for tables of size 1000, 5000, 10000, 15000 and 20000. The same was done for STEMPELATOR since it uses STEMPEL internally. There are two versions of REG in the experiment (for the two provided rule sets). LAMETYZATOR-C is an adjustment made to raw results from LAMETYZATOR, which replaces unknown stems with verbatim copies of the input word (to restore part of the intentionally stripped information, see comment in Section 2.2.).

We then compared the expected output to the results produced by stemmers seeking answers to the following questions:

- How good a given stemmer is at creating unique stems for word forms?
- How often are stems identical to gramatically correct lemmas?

Finally, we also measured the speed of processing as an essential aspect for most practical applications. All tests were carried out on commodity hardware – a Pentium III, 850 Mhz with 512 MB RAM and a fast UATA hard drive. For programs written in Java, SUN's JRE 1.4 series was used.

3.2. Test data

There was a total of 11 test sets. Each set contained many lemmas with associated inflected word forms. Five test sets were created as a random sample of 5000 lemmas from author's own resources. In the remaining 6 test sets we made an attempt to simulate 'real life' data and extracted lemmas and inflected forms from samples of tagged Polish texts available as part of the IPI PAN corpora (Przepiórkowski, 2004). Names of test sets and their sizes are illustrated in Table 1. Please note that test sets extracted from tagged samples have a much lower number of inflected forms – this is a normal side-effect of Zipf's law (lemmas in random samples were created artificially by applying ispell's inflection rules).

Name	Lemmas	Inflected
a-publi	11050	22584
b-prasa	13070	24707
c-popul	14224	27678
d-proza	16034	29827
e-dramat	11373	21711
all.lemmas	35269	80483
rnd.sample.5000.1	5000	100309
rnd.sample.5000.2	5000	101138
rnd.sample.5000.3	5000	101028
rnd.sample.5000.4	5000	100334
rnd.sample.5000.5	5000	99047

Table 1: Test data sets used for the experiment.

3.3. The results

3.4. Ability to find a unique stem

We took into account lemmas (entries in the test set) with at least two inflected forms and measured the number of lemmas for which all word forms were conflated to a unique stem.

We made two versions of the above calculations: one considering only the first stem returned for any inflected form (a realistic approach if no context analysis is performed), and second by looking for any stem present in stems generated for all inflected forms.

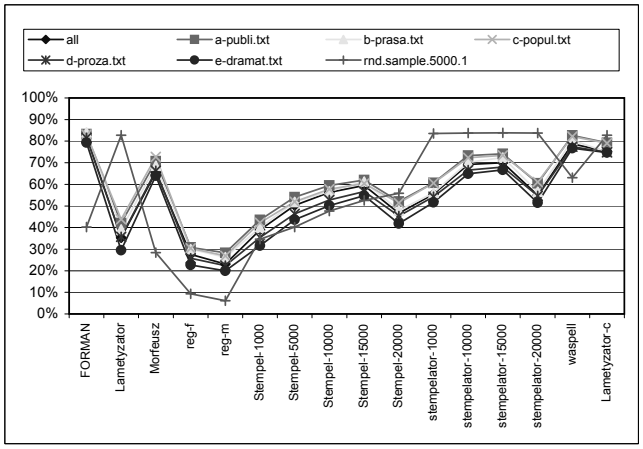
In the former case, as seen in Figure 1(a), even the results returned by morphological analyzers did not exceed 80% (because it is impossible to tell which lemma should be ordered first).

LAMETYZATOR very low score (around 40%) on previously unknown data is a result of omitted lemmas in its dictionary – when an adjustment is made (LAMETYZATOR-C), the score goes up to around 80%.

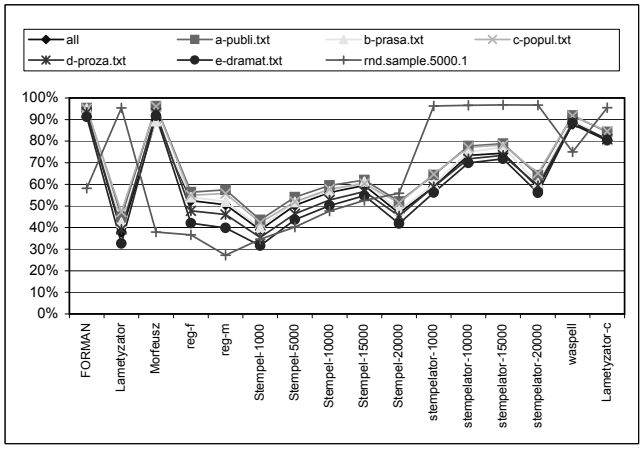
Poor performance of REG suggests serious integrity issues with its rules database. Manual inspection of the results showed that in many cases, REG returns correct lemmas for individual inflected forms, but is not consistent for all inflected variations of a lemma.

STEMPEL observes a steady increase of quality with the size of the dictionary until it reaches 15000 lemmas. For dictionary 20000, there is a sudden drop of quality of results. This seems to be only an unfortunate incident with the dictionary used in the experiment. In a separate test with 20 different dictionaries of identical size, the correlation of dictionary size and quality was evident with the point of saturation at about 75% and table size 25000.

The hybrid approach used in STEMPELATOR turned out to improve the results achieved by stand-alone LAMETYZATOR and STEMPEL. This can be further improved



(a) Identical first stem for all inflected forms



(b) Identical stem exists at any position in stems returned for inflected forms

Figure 1: Percentage of all lemmas with at least two inflected forms for which stemmers found a unique stem.

if we add all lemmas to the dictionary of LAMETYZATOR (as in LAMETYZATOR-C) and only applied STEMPEL to the remaining unknown words. We wrote such a hybrid stemmer recently (after this paper had been accepted) and it turned out to support our intuition (Weiss, 2005).

Note a low score achieved by MORFEUSZ in Figure 1(a). This was caused mainly by the assumption of taking the first stem from the result returned for each word form. When we looked for any stem spanning all inflected forms of a lemma (Figure 1(b)), MORFEUSZ was significantly better. Interestingly, in practical applications in Information Retrieval a unique stem is the most attractive option as it simplifies word comparisons – a fundamental element of more complex algorithms. A stemmer returning more than one stem for a word form is actually more problematic than helpful if one wants to avoid further linguistic analysis (even shallow).

WASPELL also achieved a very high score in this test – slightly better than adjusted LAMETYZATOR and comparable with full morphological analyzers. It is a pity source code to this stemmer is not yet public.

In both charts in Figure 1 we may observe significant difference between test data from IPI PAN corpora and our random samples of lemmas from a custom data set. We could not come up with any other explanation of this fact other than that our data set had grammatically incorrect lemmas (a conclusion drawn after looking at MORFEUSZ and FORMAN's scores in Figure 1(b)). Yet, the rule-based stemmers trained on this data turned out to be quite effective on IPI PAN texts, so we belive their stemming capabilities are credible.

3.5. Ability to find the correct lemma

We assumed a single 'correct' lemma to be the symbol an inflected form should conflate to. We calculated the ratio of all correctly stemmed forms to all forms present in the input (this is slightly different to what was presented in Section 3.4. – we consider individual inflected forms, not their sets).

If only the first stem returned for an inflected form was considered then LAMETYZATOR-C and WASPELL ranked at around 75% and STEMPELATOR at around 70% of correct lemmas. MORFEUSZ and FORMAN ranked at around 80–85%. This score is for the IPI PAN data set, the same score for our random samples was much higher for STEMPELATOR, WASPELL and LAMETYZATOR – around 90–95%. Scores for MORFEUSZ and FORMAN remained at approximately the same level of 80–85%. If any stem returned for a single word form was considered a correct answer then LAMETYZATOR, STEMPELATOR and WASPELL hit almost a 100% accuracy. Unfortunately results of MORFEUSZ and FORMAN increased only marginally to 85–90%. Manual inspection of the results revealed certain inconsistencies in the learning data set. For instance, lemma for *chwilowo* in the ispell-generated dictionaries was *chwilowy*. Similar errors were also encountered with negations: lemmas in i-spell were also negations as in *niewyraźne→niewyraźny*. A qualitative training set would surely improve the results of algorithmic and dictionary stemmers.

As a final conclusion, please note that REG turned out to be much better in this test (between 60 and 70%), confirming that it has the capability of transforming words back into their stems, but is not consistent among word forms.

3.6. Efficiency

As we mentioned in the introduction, it was expected that REG would be the slowest stemmer. The scale the difference was quite unexpected though – it took REG 84 and 109 minutes (depending on the rules file) to process the test sets, whereas other stemmers completed the task under 90 *seconds* (see Table 2). REG may be excused as a purely research initiative, but quite amazingly, also commercial FORMAN was almost ten times slower than the rest of the competition. STEMPEL leads the speed race with times under 40 seconds even for the largest dictionary size (or about 20000 words per second). LAMETYZATOR

and WASPELL were a little behind with 15500 and 11000 words per second. Please note that all the winning stemmers are Java based and the timing in our results includes the overhead of multiple spawns of the Java Virtual Machine (one for each test set).

Stemmer	Total time (mm:ss)	Words per second
FormAN	11:10	1053
Lametyzator	00:45	15683
Morfeusz	00:50	14115
reg-f	84:13	140
reg-m	109:41	107
Stempel-1000	00:27	26138
Stempel-5000	00:29	24335
Stempel-10000	00:28	25205
Stempel-15000	00:32	22054
Stempel-20000	00:40	17643
Stempelator-1000	00:48	14703
Stempelator-5000	00:49	14403
Stempelator-10000	00:53	13316
Stempelator-15000	00:54	13069
Stempelator-20000	00:56	12602
Waspell	01:03	11202

Table 2: Speed of processing (a sum of processing times for all test sets).

4. Summary and conclusions

In this paper we provide a short survey of freely available and open source stemmers for the Polish language. We also conduct an experiment comparing those stemmers in an attempt to simulate the requirements of an information retrieval application.

The results seem to indicate that the quality of dictionary-based and rule-based stemmers are very good, in many cases comparable with commercial morphological analyzers. Occasional failures can be blamed on inaccuracies in the training data sets.

Most of the introduced products also release their source code, so there is a great opportunity for further development. We showed one potential opportunity by hybridizing a dictionary and rule-based stemmers for the needs of this paper, later extending this concept in a follow-up technical report (Weiss, 2005).

Acknowledgement The author would like to thank Andrzej Białecki for his invaluable help and providing resources that made this work possible.

This research has been supported by grant: KBN 3 T11C 050 26.

5. References

Białecki, Andrzej, 2004. *Stempel* – Algorithmic Stemmer for the Polish Language. [on-line] `http://getopt.org/stempel/`.

Daciuk, Jan, 1998. *Incremental Construction of Finite-State Automata and Transducers, and their Use in the Natural Language Processing*. Ph.D. thesis, Technical University of Gdańsk, Poland.

Galambos, Leo, 2004. Semi-automatic stemmer evaluation. In *Proceedings of the International IIS: Intelligent Information Processing and Web Mining Conference*, Advances in Soft Computing. Zakopane, Poland.

Grzymała-Busse, Jerzy W., 1992. LERS – A System for Learning from Examples Based on Rough Sets. In Roman Słowinski (ed.), *Intelligent Decision Support Handbook of Applications and Advances of the Rough Sets Theory*. Kluwer Academic Publishers.

Hajnicz, Elżbieta and Anna Kupść, 2001. Przegląd analizatorow morfologicznych dla języka polskiego. Technical Report 937, IPI PAN.

Ispell-pl, 2002. Polish dictionary for ispell. [on-line] `http://sourceforge.net/projects/ispell-pl`.

Płotnicki, Zbigniew, 2003. *Słownik morfologiczny języka polskiego na licencji LGPL*. Master's thesis, Poznań University of Technology.

Przepiórkowski, Adam, 2004. The ipi pan corpus: Preliminary version. [on-line] `http://www.korpus.pl`.

Stefanowski, Jerzy and Dawid Weiss, 2003. Carrot^2 and language properties in web search results clustering. In *Proceedings of AWIC-2003, First International Atlantic Web Intelligence Conference*, Lecture Notes in Computer Science. Madrid, Spain: Springer.

Szafran, Krzasztof, 1996. Analizator morfologiczny SAM-95. Technical Report TR 96 226, Faculty of Mathematics, Informatics and Mechanics. Warsaw University.

Wąchnicka, Justyna, 2004. *Odkrywanie reguł lematyzacji dla języka polskiego w oparciu o słownik* ispell-pl. Master's thesis, Poznań University of Technology.

Weiss, Dawid, 2003. Lametyzator. [on-line] `http://www.cs.put.poznan.pl/dweiss/xml/projects/lametyzator/index.xml`.

Weiss, Dawid, 2005. Stempelator: A Hybrid Stemmer for the Polish Language. Technical Report RA-002/05, Institute of Computing Science, Poznań University of Technology, Poland.

Weiss, Dawid and Jerzy Stefanowski, 2003. Web search results clustering in Polish: Experimental evaluation of Carrot. In *Proceedings of the International IIS: Intelligent Information Processing and Web Mining Conference*, Advances in Soft Computing. Zakopane, Poland.