Slovak morphology analyzer based on Levenshtein edit operations

Radovan Garabík

Ľudovít Štúr Institute of Linguistics
Slovak Academy of Sciences
Bratislava, Slovakia
korpus@juls.savba.sk
http://korpus.juls.savba.sk/

Abstract. Levenshtein edit operation is a basic string operation – insertion, deletion or substitution of a character in a string. Sequence of edit operations can be used to transform basic word form (lemma) into an inflected form, and the same sequence can can be used to transform lemmata belonging to the same inflectional paradigm. Presented system contains inflection paradigms of over 56000 lemmata from Short Dictionary of Slovak Language and from the most frequent word forms in the Slovak National Corpus, together with detailed grammar information about each generated word form.

1 Levenshtein distance and some definitions

Levenshtein distance[1] is a metric defined on the space of strings as a minimum number of Levenshtein edit operations needed to transform one string into the other, where by a Levenshtein edit operation we understand insertion, deletion or a substitution of a character.

A Levenshtein edit operation e can be formally described as $e = (o, s, d)$ – a triple of operation type o, position in the source string s and position in the destination string d, where operation type o is one of *replace*, *insert* or *delete*. For *replace* or *insert*, the replacement/new character is taken from the destination string.

Sequence of edit operations $q = (e_1, e_2, e_3, ...)$, together with the destination string D, when applied to a string $S \in \mathbb{S}$ defines a mapping function $f : \mathbb{S} \mapsto \mathbb{S}$, where $\mathbb{S}$ is a set of all strings.

To each word form $w \in \mathbb{W}$, where $\mathbb{W}$ is a set of all the words we can assign a set of *grammar categories* $G_w = \{g_1, g_2, g_3, ...\}$ represented by short mnemotechnical strings (called *morphological tags*).

Now for each tagged word form together with its morphological tag $(w_i, g_i) \in \mathbb{W} \times \mathbb{G}$ there exists a mapping function f_i consisting of Levenshtein edit operations such that $f_i(l) = w_i$, where l is a deliberately chosen word, called *lemma* and considered to be a basic word form for a given lexeme.

2 Technical implementation

Our system is really just a morphology generator – for each lemma known to it, it is able to generate all the forms, together with their respective tags. By putting all the forms and tags with information about lemma into a database[2], the system is able to work as a morphology analyzer – we just look up the analysed form in the database and find out corresponding morphological tag and lemma.

The system consists of two logically disjunct parts. One part is responsible for creating tables of paradigm templates and lists of mapping of all the lemmata into appropriate paradigm templates. This contains also helper programs used by linguists to create, evaluate and modify these tables and lists.

The second part is meant for end users queries and is nothing more than a simple wrapper around the database query library, to facilitate the lookup, with some simple logic implemented to account for creating superlatives out of comparatives (by adding the prefix *naj-*) and for creating verb negation (by adding the prefix *ne-*, with the exception of the verbs *íst'* and *byt'*),

The software is published under GNU General Public License[3] version 2 and can be obtained from the Slovak National Corpus WWW page[1].

3 General principles

The system responsible for creating, testing and editing paradigms is written completely in the Python programming language[4], paradigm editing and testing is done using a simple CLI interface.

All the texts, input and output in our system is unconditionally in UTF-8 encoding[5], and all the internal logic of the system uses Python unicode strings. Word forms in the tables are kept in UTF-8 encoding.

Since it is a suffix morphology we are interested in, we need to count the position for Levenshtein edit operations from the end of the words, so that words of different lengths but sharing the same suffix inflections can be declined by the same paradigm template, in order not to inflate unnecessarily the number of paradigm templates. This is easily realised by reversing the input strings before applying the edit operations, and by reversing the output obtained as the result – all done transparently to the users.

To keep the number of paradigm templates down we let our system work in NFD Unicode normalization[6] internally, normalizing user input into NFD before processing, and normalizing the output to NFC. This takes into account changes in orthographic palatalization of the last consonant *d, t, n* or *l*, represented only by adding or removing the final háček (combining diacritical character in the NFD normalization), but in the more usual NFC normalization it would have to be represented by changing the last character, and therefore requiring separate paradigms for each consonant.

[1] `http://korpus.juls.savba.sk`

4 API

System contains three constant database tables:

- `form2lemma.cdb` – table containing word forms as keys and corresponding lemmata as values
- `form2taglemma.cdb` – table containing word forms as keys and morphological tags and lemmata as values
- `lemma2tagforms.cdb` – table containing lemmata as keys and morphological tags and word forms as values

The constant database tables can be accessed directly, from any programming language supporting tdb database, or converted into a convenient form. However, the databases do not contain neither verb negation, comparatives nor superlatives. Preferred higher level API written in the Python programming language is contained in the module `mlv_skling`. The module has one public class, `Morphology`, that takes during instantiation as an argument path to the directory containing all the necessary morphology tables. Methods available are `form2taglemma`, `lemma2tagforms`, `get_stems` and `get_stem`. `form2taglemma` and `lemma2tagforms` analyze given word form or lemma and return tuple of morphological tag and lemma, or morphological tag and form. `get_stems` applies simple stemming algorithm to the word and return a list of all possible stems. `get_stem` returns just the first stem found.

5 Stemming algorithm

Stemming is the process of finding the stem (base or root form) of inflected words, regardless of the existence of the stem alone – it is sufficient if the word forms of the same lexeme map to the same stem, in order to facilitate full text indexing and search. Usually, for full text query purposes, we are not interested in the grammar analysis, and we want to maximize recall at the expense of precision. Our stemming algorithm uses lemma as the basic form, stripping vowels following the rightmost consonant in the lemma, and in case of verbs, stripping the infinitive suffix *-t'* and then stripping the vowels. This collapses many near homonyms to the same form, directly usable as the word stem.

6 Language coverage

At the time of writing, the database contains all the words from the 3^{rd} edition of the Short dictionary of Slovak language, with several thousand additional most frequent words present in the Slovak national corpus, adding up to 56269 different lemmata (54315 unique lemmata, accounting for homonymy). These lemmata are inflected by 1365 different paradigms, giving 601 253 unique word forms and 1 616 379 different pairs of morphological tags and word forms.

An average tokenized fiction text contains 19 % of punctuation and other nonword elements. On average, the analyzer covers 91 % of the remaining tokens, where 45 % of tokens are unambiguously assigned their morphology categories and lemmata, 61% of tokens have unambiguously assigned lemmata, but not morphological tags.

Stemming is markedly better, only 6.6 % of tokens cannot be stemmed unambiguously (and this is mostly due to rather frequent ambiguous words *je*, lemma *byt'* or *jest'* and *si*, lemma *byt'* or *si*) – in an information retrieval system, these words would be probably included in the list of stopwords, further improving unambiguity of the stemming.

7 Conclusion and future work

At the time of writing, vocabulary of the presented system is still being improved. The next task will be to add most frequent acronyms, abbreviations and proper names (toponyms and anthroponyms). Numerals will be taken care with the help of additional module, exploiting their regular formation. To improve the percentage of unmarked words in the analyzed texts, a "guesser" module will be implemented, trying to find out the nearest appropriate morphologic tag for words not found in the dictionary, based on suffix similarity with existing words.

The analyzer is able to obtain lemmata and grammar categories for a broad range of most frequent Slovak words, including punctuation and digits, and is successfully used in the Slovak National Corpus database.

References

1. Левенштейн, В. И.: Двоичные коды с исправлением выпадений, вставок и замещений символов, Докл. АН СССР, 163, 4, (1965) 845–848.
2. http://cr.yp.to/cdb.html
3. Free Software Foundation, Inc. (1989, 1991)
4. http://www.python.org/
5. The Unicode Consortium. The Unicode Standard, Version 4.0 Boston, MA, Addison-Wesley Developers Press, ISBN 0-321-18578-1 (2003)
6. The Unicode Consortium. Unicode Technical Report #15: Unicode Normalization Forms. http://www.unicode.org/unicode/reports/tr15/