Heuristical Coding of String Transformations

Tibor Répási, László Kovács

Department of Information Technology, University of Miskolc, Hungary
{repasi,kovacs}@iit.uni-miskolc.hu

Abstract: As a part of a study on statistical grammar learning, the word inflection is investigated in this article. The word inflection is used to create different grammatical instances of the word. In this paper, the different coding alternatives to describe the strings and the string transformations are investigated. The proposed methods are tested on a natural language, the Hungarian language.

Keywords: string transformation, word inflection

1 Problem Statement

The scope of our study is to build a statistical method to learn the rules of word inflection. The pre-requirement regarding the language is that the language uses words which are sequences of characters. In our language model, the words are built up from characters. During the inflection, a word is mapped to a word (a different word or the same word). The initial word is called stem. The grammar rule describes the generation of the inflected form from the stem.

It can be assumed that the transformation rules depend on the stem form of the concepts. There are distinct and relative few rules in the languages, i.e. the number of rules is much more less than the number of stems. In our approach, the rule assignment task is considered as a classification method, each stem is assigned to a rule class. We understand a rule class as a set of independent string transformations describing a particular grammar rule. In our training set we use to extract the rule class used to build the objective of nouns.

This paper focuses on the extraction and description of string transformations representing word inflections.

1.1 Requirements

One of the main requirements on the method is the efficiency. As the set of words W can be very large, the learning algorithm should have a polynomial cost

function of low grade. One of the key factors in efficiency is the selection of appropriate word and rule representation form. The main requirements on the representation form can be summarized in the followings:

- efficient management,
- generalization feature,
- position independently representation,
- compact description,
- flexibility.

2 Analysis of Base Methods

Regarding the string representation forms, the formula should represent a group of arbitrary strings. The intuitive way of group representation is to describe the common characteristics of the member strings. As different substrings may be common in the member strings, the representation should include all of the substrings. One of the key factors is whether the representation can preserve the ordering relation or not. If the ordering is not given, the representation form leads to over-generalization. On the other hand, if ordering is given, the generalized form should be described with a graph, that makes the execution more costly.

Another approach is the application of regular expressions. Regular expressions are meant to do pattern matching. An expression describes a rule of how to construct a string. Consisting of a sequence of atoms, each one describing a string feature, the atoms can be printable characters, which are substituted by themselves, and expressions metacharacters. These metacharacter expressions can describe generalized character substitutions, grouping of expressions, multiplicity of the preceding atom or logical relationships between atoms. Although the regular expression is powerful enough to describe general string patterns, the generalization of different regular expressions is not an unambiguous task. The resulted expressions tend to be over-generalized.

Considering the string transformation, the main solution found in the literature is the usage of edit transformation steps. The formalism using edit transformation uses three basic operation steps:

- replacement of a character with another,
- insertion of a new character,
- deletion of an existing character.

The chain of transformation steps for a (w,w') pair is calculated with a dynamic programming method. The main drawbacks of the algorithm has a relative high O(m^2) cost value. The other major drawback is the missing generalization feature. The transformation chain is based on rigid positions of the characters.

2.1 Additional Assumptions

To create the most suitable description of string transformations we use two well known assumptions of the target language:

a) the language is a suffixing language, which means that word inflections are built by adding suffixes to the stem, while the stem stays mainly unchanged,

b) the only kind of changes to the stem are targeted on the last vowal which can be prolonged, trimmed or dropped.

3 Subset-based Representation

A character is the atomic structure of the language. The alphabet $A = \{a\}$ is the set of characters in the language. There is a special separator character in the alphabet denoted with ε. A base word is a sequence of characters: $w = a^*$, where $a \in A$. The * symbol denotes the Klee-star operator (closure on the concatenation). The set of words is denoted by W. The ε symbol can occur only at the first and the last positions of the world.

The n-gram denotes a word of length n. The sub-word relation (≤) is defined as a subsequence:

$$w,v \in W,\ w \le v : \exists\ i,\ \forall\ j:\ w_j = v_{i+j}$$

In order to describe the common parts of words, a new concept, the generalized word is introduced. A generalized word means here a set of words. The consideration behind this concept is the following. Each normal word can be taken as the set of its sub-words. Thus the common part of two words is equal to the intersection of these sets. In order to describe the set in a compact form, the concept of generator word is introduced. A word is a generator in a set if all of its sub-words are contained in the set. The compact form of the set is given with the set of the generator words. So, the generalized form of a normal word can be given with the word itself, as it is the generator of the corresponding set. The main benefit of this generalization concept is that it enables to build a lattice of generalized words and thus also the classification methods based on the concept-lattice can be applied to solve the grammar induction problem.

A generalized word w^+ is defined as a set of extended words:

$$w^+ = \{(w,\upsilon)\}$$

where w denotes a word and υ is position descriptor. In our approach the position descriptor is a multi-valued attribute. The υ is usually a set of linguistic variables like front, end, near end, middle or middle-end. These variables describe the approximated position of the substrings. The set of generalized words is denoted by W^+. A generator word of w^+ is a word, whose all sub-words are contained in the set of w^+. The set of generators related to a w^+ is denoted by $g(w^+)$.

$$g(w^+) = \{ w \mid w \in w^+, \forall w' \leq w : w' \in w^+, ! \exists w'' \in w^+ : w \leq w'' \}$$

In the case of generalized word, the position description of an element in $g(w^+)$ is equal to the union of the contained position descriptors of the member words. If the minimal length of words in a generalized word is constrained by n, an n-level generalized word is defined:

$$\forall w \in w^+ : |w| \geq n$$

Based on the subset relationship, an ordering can be introduced on the set of generalized words. For any $w_1^+, w_2^+ \in W^+$ words, the $\leq$ relation is defined as

$$w_1^+ \leq w_2^+ : \forall w \in w_2^+ \Rightarrow w \in w_1^+$$

If $w_1^+ \leq w_2^+$ is met, then the word w_2^+ is called a generalization of the word w_1^+.

The intersection of the generalized words can be computed as the intersection of the sets w_1^+ and w_2^+. It results in a generalized word. This word is the lowest common generalization of the two words.

The proposed word model provided a good theoretical foundation to manage the generalization of words using Formal Concept Analysis methodology.

4 RE-based Representation

Regular expressions are well known and widely used, due to their flexibility leads to taking over the concept of metacharacter expressions for describing higher level operations. Unfortunately, regular expressions are not suitable for any kind of transformation of strings.

As the basic string edition operations are too fine-grained and not general enough in editing only the necessary part of the word. On the other hand these operations are not specialized on assumption b), since they simply not aware of vowals at all, neighter of short and long vowal pairs.

Both, regular expressions and string edition is used to parse a string in the left-to-right order. According to assumption a) the changing part of the word – suffix – is

at the end, while the constant part – the stem – is at the beginning, the proposed coding should use the right-to-left order in parsing the words.

Let a transformation rule *T*, be a sequence of atoms. Atoms are parsed one-by-one, but there are 3 types of atoms which has to follow in the order of types:

- extensions: described by printable characters, symbolizing characters which has to be suffixed to the stem,
- modifications: describes a character transformation to be done on the current character of the stem:
  - apostrophe ('): the vowal has to be prolonged,
  - headmark (^): the vowal has to be trimmed,
  - dot (**.**) : the character is copied unchanged,
  - tilde (**~**): the character is dropped.
- copy: the asterisk (*****), which appears at the end of the transformation signs that the remaining part of the stem has to be copied without any change.

4.1 Examples

As example we use the three nouns “alma”, “szamár” and “selyem”, the subjective forms of which are in order “almát”, “szamarat” and “selymet”. The transformation code for the first word is “t'*” (t, apostrophe, asterisk), that means we take the stem “alma” and suffix it with a “t”, than we modify the last character to a prolonged vowal, finally copy the remaining part of the stem unchanged. In the case of the second word the transformation is “ta.^*”, that is we suffix the stam with “t” first, than with “a”, so the ending will be “at”. Than we copy the last character of the stam, and trim the next one, finally simply copy the remaining part. The resulting “szamarat” is the subjective form of “szamár”. For the third example the transformation is “te.~*”, which suffixes the stem with “t” than with “e”, the last character remains unchanged, the next is dropped, remaining part is copied. The result is “selymet”.

4.2 Testing

As a testrun we created the transformations for ~4500 hungrian nouns which were prepared and added the subjective form for each. Within the testrun each pair was scanned to create the transformation, then the transformation has been done on the stem, to test if the original and the transformed word are equal. This procedure was done on an ordinary PC in about 20 seconds.

References

[1] Satta G., Henderson, John C. “String Transformation Learning” Proceedings of the Eighth Conference on European Chapter of the Association for Computational Linguistics, 1997

[2] Crochemore M., Rytter W. “Text Algorithms”, Oxford University Press, 1994

[3] Jeffrey E. F. Friedl “Mastering Regular Expressions”, O'Reilly, 1997

[4] Levenshtein V. I. “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”, Russion Problemy Peredache Informatsii, 1965

[5] Manning C., Schütze H.: Foundations of Statistical Natural Language Processing, MIT Press, 1999

[6] Jurafsky D., Martin J. H.: An Introduction to Speech Recognition, Computational Linguistics and Natural Language Processing, 2006

[7] Krenn, B., Samuelsson C.: The Linguistic's Guide to Statistics, 1997