MultiScience - XXX. microCAD International Multidisciplinary Scientific Conference
University of Miskolc, Hungary, 21-22 April 2016, ISBN 978-963-358-113-1

EDIT DISTANCE BASED GRAMMATICAL RULE GENERATION USING AN IMPROVED COST FUNCTION

Gábor Szabó
PhD student
Department of Information Technology, University of Miskolc

ABSTRACT

Natural language processing is a very popular research area that poses many challenges especially for highly agglutinative languages. Although there are some existing solutions that use different methods and models, applying the concept of Levenshtein distance is very rare among them. In this paper a novel Levenshtein distance based grammatical rule generation method is proposed that uses a uniquely improved cost function for finding the optimal rule for each word pair in the training data set. The base of the method is to extend every character in the words with their phonetical attributes to reduce the number of candidate rules, thus helping to choose one of them. A simple test experiment proves that the novel cost function can indeed improve the quality of the generated rules.

1. INTRODUCTION

The original research area that I've been working on was the theory of ontologies that are the ultimate tools of knowledge engineering [2]. Usually the input data of ontologization is huge amounts of free text, that's why natural language processing came into the picture. NLP's lowest level is morphology and grammatical rule generation that helps us to understand the relationships between words and their roles in the text. Being a native Hungarian and knowing that highly agglutinative languages pose many challenges in this area, the Hungarian language is used extensively in my researches.

In the literature there are some very promising results of morphology learning and rule generation. [4] presents a novel Machine Learning approach to the acquisition of stochastic string transductions based on Pair Hidden Markov Models. PHMMs are mainly used in computational biology, however they can be applied in other machine learning problems as well, like morphology learning. The evaluation of the method shows a near-perfect result for English past tense and a somewhat worse but still good hit rate for German and Arabic words in plural form. [5] tested Hidden Markov Models against another agglutinative language, Finnish with great results.

A very easy method for learning word transformations is TASR [6] that operates on a simple tree structure that contains the suffix transformations of the training data. After building this tree structure, the input words are matched to the tree nodes from bottom to top and the first matching node's transformation is applied. The method is restricted to handle suffixes, although it shows exceptional results and great runtime speed [8]. The other side of the spectrum is Ostia [7] that uses finite state transducers to learn string transformations. The mathematical model of Ostia is very

DOI: 10.26649/musci.2016.059

sophisticated, but as such, building the transducer takes more time and effort. Although Ostia can model any kinds of transformations at any position in the source words, practice shows that it has an additional time cost in real life examples [8].

In this paper, a somewhat unusual approach is taken for learning string transformations by deeming every word as a simple string and induce the rules based on the Levenshtein distance. The Levenshtein or edit distance was first published in [1] and as the Markov model and its variations, it has many application areas as well. [9] applies this concept into security and cryptography where an asymmetric two-party protocol is introduced to find the minimal edit distance of two strings under privacy constraints. As the core of the Levenshtein distance is to measure the similarities among different strings, it can be well used in finding plagiarism, pointed out by [10]. The base concept is that many people modifies only small portion of copied text, and this way the similarities can be recognized by calculating edit distances.

If we search for the grammatical applications of the Levenshtein method, we can find some results in literature, but these are not so frequent. [11] for instance uses the shortest edit script as a substep in a proposed algorithm that solves the problem of morphological tagging and lemmatization. As opposed to Morfette, this paper presents a unique model for handling inflection learning with the Levenshtein distance in its central point.

After the literary outline, the paper's next sections are structured in the following way:

- Section 2 introduces the theory of Levenshtein distance to build the solid ground of the results of the paper.
- Section 3 presents an improved cost function that can be used during the training algorithm to improve the quality of the generated grammatical rules.
- Section 4 analyzes the quality of the resulting rules to see how much improvement we can reach with the new cost function.
- Section 5 proposes further questions and development plans to build a useable, performant system by generalizing characters into n-dimensional vectors.

2. THE LEVENSHTEIN DISTANCE

The Levenshtein or edit distance [1] is a simple measurement of a string pair that shows how much the two strings are similar to each other by giving us the number of simple transformation steps that are required to get the target string from the source string. These transformation steps can fall into four categories: addition of a character, deletion of a character, replacing a character with another character and leaving a character as is.

To formalize our model, let's introduce the following concepts. Let's introduce our alphabet like:

$$c \in C^* = C \cup \{ \epsilon \} \tag{1}$$

where c is a character, ϵ is the empty character and C is the set of non-empty characters.

A transformation step can be formalized as

$$s \subseteq C^* x C^* \tag{2}$$

where we transform one character to another one. Of course the category of the transformation step can be denoted with a subscript symbol:

$$\begin{gathered}(\epsilon, c) \in s_+ \\ (c, \epsilon) \in s_- \\ (c, c) \in s_= \\ (c, c') \in s_{\neq}\end{gathered} \tag{3}$$

These simple transformation steps can be organized into a transformation path that shows how to transform the source string to into the target. If we imagine the so-called Levenshtein matrix, then this path leads us from the top left corner of the matrix to the bottom right corner. See Fig. 1 for a graphical example of the matrix of word pair *(alma, almát)* that is the base and accusative form of *apple* in Hungarian.

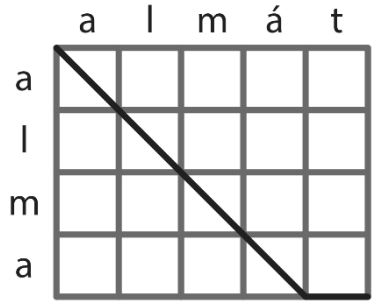

Fig. 1
An optimal Levenshtein path for the Hungarian word pair *(alma, almát)*

The path in the matrix can be denoted with a sequence of transformation steps:

$$p = (s_1, \dots, s_n) \tag{4}$$

Each transformation step on the path has a cost. According to the original method, invariant replacements have 0 cost, while every other transformation costs 1:

$$\begin{gathered}cost(s_=) = 0 \\ cost(s_+) = cost(s_-) = cost(s_{\neq}) = 1\end{gathered} \tag{5}$$

The edit distance δ equals the total cost of an optimal path $\hat{p}$ which has minimal total cost:

$$\delta = \min_{\hat{p}} \sum_{s \in \hat{p}} cost(s) \tag{6}$$

After we've chosen a $\hat{p}$ path, we can drop those parts that are invariant. This way we can get an $r = (t_1, \dots, t_m)$ rule for each word pair where $t_i = (s_{i_1}, \dots, s_{i_k})$ is a coherent slice of $\hat{p}$ that satisfies the $s_{i_j}(c) \neq c$ condition. $R = \{ r \}$ will be the set of extracted rules.

However, the path is not unique, there might be multiple paths in the Levenshtein matrix with minimal cost. This is acceptable for most general applications of this method, but for grammatical rule generation it influences the rule we generate. Fig. 2 gives another minimal path of the previous example.

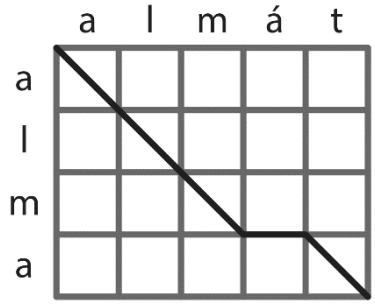

Fig. 2

Another optimal Levenshtein path for the Hungarian word pair *(alma, almát)*

Humans who speak Hungarian know that the first figure's rule (replacing *a* with *á* and adding *t*) is more intuitive and thus more accepted than the rule of second figure (adding *á* and replacing *a* with *t*), so the chosen method should be able to distinguish between them and successfully choose the first, better rule.

In the next section, an improved cost function is introduced to tackle this problem.

3. IMPROVED COST FUNCTION

The intuition behind the improved cost function is that historically the Hungarian language's grammatical rules had been formed according to phonetic properties, i.e. those rules are more usual that don't require us to pronounce sounds that are phonetically far from each other. Of course in Hungarian there are several exceptions, but these exceptions are not part of the investigation at this stage.

In order to bring the phonetic properties into the model, a new alphabet is created by extending the ordinary characters with their phonetic attributes defined as:

$$a: C \to R_a \tag{7}$$

The Hungarian language has 4 main phonetical attributes for vowels and 4 for consonants, as well as a categorization attribute (vowel or consonant). So our attribute universe A will contain 9 attribute functions. The next tables show the used attributes.

The first step of the new method is to map each character to the new extended character introduced above. After that, the route searching algorithm in the Levenshtein matrix is also slightly modified to include a new, improved cost function which is the total number of different attribute values along the Levenshtein path:

$$cost(s) = \sum_{\substack{a \in A \\ (c,c') \in s \\ a(c) \neq a(c')}} 1 \tag{8}$$

To summarize the calculation, for each transformation step we count those attributes that have different values for the two characters. If either character is empty,

the other character's total attribute count is used as the difference. It can be easily seen that for invariant replacements the new cost function still yields 0.

Table 1
Phonetic attributes of Hungarian vowels

		Horizontal Tongue Position							
		Front				Back			
Vertical Tongue Position	Close	ü	ű	i	í	u	ú		
	Middle	ö	ő		é	o	ó		
	Semi-Open			e		a			
	Open								á
Length		Short	Long	Short	Long	Short	Long	Short	Long
Lip Shape		Rounded		Unrounded		Rounded		Unrounded	

Table 2
Phonetic attributes of Hungarian consonants

		Way of Sound Production								
		Plosive			Fricative		Lateral-Fricative		Lateral-Approximative	Trill
Voice		Voiced		Unvoiced	Voiced	Unvoiced	Voiced	Unvoiced	Voiced	Voiced
Place of Sound Production	Bilabial	m	b	p						
	Labio-Dental				v	f				
	Dental-Alveolar	n	d	t	z	sz	dz	c	l	r
	Dental-Postalveolar				zs	s	dzs	cs		
	Palatal	ny	gy	ty	j					
	Velar		g	k						
	Glottal					h				
Uvula Position		Nasal	Oral							

Returning to the example of the previous figures, it seems logical to make a little modification and move *á* and *é* to the right side of their short vowel pairs *a* and *e* so that every vowel pairs' number of changing attributes is the same. To handle special

consonants like *ly*, *q*, *w*, *x* and *y*, we introduce an artificial discriminator attribute that has six attributes: one for each special consonant and a *'normal'* flag.

4. QUALITY ANALYSIS OF THE GENERATED RULES

To compare the quality of the generated rules in case of the original unit and the novel attribute based cost function, first a training data set was prepared. For this the results of a previous work [8] was used with 48 347 word pairs of accusative case.

The intuition behind the test execution is that if we determine which the most common rules in practice are, those rules are good candidates for rule generation. This means if we cannot decide which rule to choose during the generation, the better choice according to the intuition is the rule that occurs more often in the training set.

To gather statistics from the 48 thousand word pairs, the following steps were executed:

1. Generate reference rules with the unit cost function based calculation, using the best 5 paths for each word pair, and calculate the relative rule frequencies.
2. Take the best rule for the word pairs using the unit cost function.
3. Generate the best rule using the improved attribute based cost function.

After the discussion about the two calculations, it is not surprising that in step 2, the best rule is often ambiguous, but the improved cost function – while maintaining the minimal value of 0 in case of invariant replacements – offers greater granularity among the candidates, and this way the optimal rule is usually easier to choose.

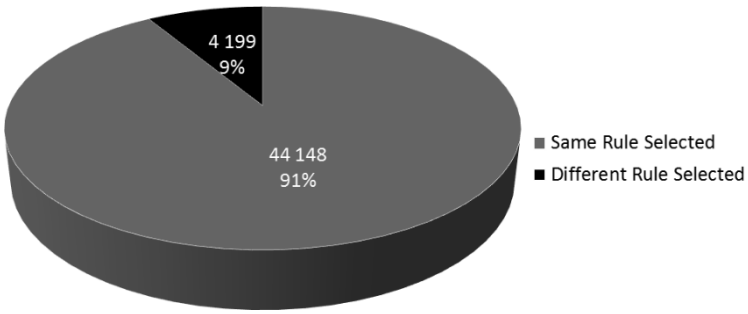

Fig. 3
Results of the unit cost function

This is verified by Fig. 3 and Fig. 4 as in case of the unit cost function, for 91% of the training set the rule with highest relative frequency was chosen, while in case of the improved attribute based cost function, this is an even higher 99%.

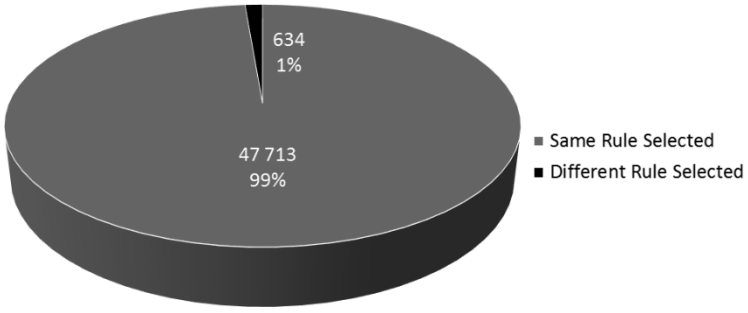

Fig. 4
Results of the improved attribute based cost function

5. FUTURE WORK

With the introduced novel attribute based cost function, the problem of choosing the optimal rule is eliminated almost entirely, but the production mode of the framework is still a work in progress. Currently the best idea that is worth evaluating is the construction of a concept lattice [3] from the generated rules, reducing the required time to search among the rules for matching. However, there are many questions that need to be answered in order to find the best possible model:

- How to intersect grammatical rules?
- When is a grammatical rule the subset of another one?
- How should a node look like to fully support searching in the lattice?
- How to optimize the process of building the lattice, which intersections can be dropped early?

Currently, the first three questions have been answered and a working prototype system exists that demonstrates that the concept is working. However, the performance of the system is not perfect yet, so the fourth question is being investigated by the time of writing this paper.

The current model of nodes contains one or more transformation objects that have the following properties:

- prefix: the prefix of the transformation, i.e. the characters before the changing characters in the word
- transformation: the transformation steps that need to be applied
- suffix: the suffix of the transformation
- front position: the index of the changing occurrence of the context (prefix + transformation base + suffix) in the source word from its beginning, i.e. 0 if the transformation needs to be applied for the first occurrence
- back position: same as front position, but from the back of the word

Intersection and the subset operator are defined for these objects using the attributed characters, which means that characters are considered attribute sets, and the operations are performed against these sets.

A further idea is that we should handle transformations differently. Instead of a simple sequence of transformation steps, they should be thought of as change vectors, i.e. the transformation step *(i, í)* would mean that the base character's length attribute is changed from *short* to *long*. This way a transformation would be more generic and the characters would be n-dimensional vectors where the projections are the attribute values in the i^{th} dimension.

6. CONCLUSION

This paper introduced a novel grammatical rule generation method based on the Levenshtein or edit distance. Besides the usage of Levenshtein distance in a morphological algorithm and treating words as simple character sequences, the new model also contains an improved cost function for finding the optimal path in the Levenshtein matrix. This cost function requires the characters of strings to be extended with their phonetical attributes, and it defines the cost of a transformation step to be the number of different attribute values between two characters. This

method is proved to generate better rules than the original unit cost function based algorithm, because the number of candidate rules with minimal cost is lower, thus choosing from them is less ambiguous. This research area has much potential, as in the future a concept lattice can be built from the generated rules to better support searching among the rules.

REFERENCES

[1] LEVENSHTEIN, V. I.: **Binary codes capable of correcting deletions, insertions, and reversals.** Soviet physics doklady. Vol. 10. No. 8. 1966.

[2] GRUBER, T. R.: **A translation approach to portable ontology specifications.** Knowledge acquisition. Vol. 5. No. 2. 1993.

[3] ANDREW, S.: **In-close, a fast algorithm for computing formal concepts.** International Conference on Conceptual Structures (ICCS), Moscow, 2009.

[4] CLARK, A.: **Learning morphology with pair hidden markov models.** ACL (Companion Volume). 2001.

[5] CREUTZ, M. and LAGUS, K.: **Induction of a simple morphology for highly-inflecting languages.** Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology. Association for Computational Linguistics, 2004.

[6] SHALONOVA, K. and FLACH, P.: **Morphology learning using tree of aligned suffix rules.** ICML Workshop: Challenges and Applications of Grammar Induction. 2007.

[7] DE LA HIGUERA, C.: **Grammatical inference: learning automata and grammars.** Cambridge University Press, 2010.

[8] SZABO, G. and KOVACS, L.: **Efficiency analysis of inflectional rule induction.** Carpathian Control Conference (ICCC), 2015 16th International. IEEE, 2015.

[9] RANE, S. and SUN, W.: **Privacy preserving string comparisons based on Levenshtein distance.** Information Forensics and Security (WIFS), 2010 IEEE International Workshop. IEEE, 2010.

[10] SU, Z., AHN, B. R., EOM, K. Y., KANG, M. K., KIM, J. P. and KIM, M. K.: **Plagiarism detection using the Levenshtein distance and Smith-Waterman algorithm.** Innovative Computing Information and Control, 2008. ICICIC'08. 3rd International Conference. IEEE, 2008.

[11] CHRUPALA, G., DINU, G. and VAN GENABITH, J.: **Learning morphology with morfette.** 2008.