A Rule based Approach to Word Lemmatization

Joël Plisson, Nada Lavrac, Dunja Mladenic
Department of Knowledge Technologies
Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
e-mail: joel.plisson@ijs.si

ABSTRACT

Lemmatization is the process of finding the normalized form of a word. It is the same as looking for a transformation to apply on a word to get its normalized form. The approach presented in this paper focuses on word endings: what word suffix should be removed and/or added to get the normalized form. This paper compares the results of two word lemmatization algorithms, one based on if-then rules and the other based on ripple down rules induction algorithms. It presents the problem of lemmatization of words from Slovene free text and explains why the Ripple Down Rules (RDR) approach is very well suited for the task. When learning from a corpus of lemmatized Slovene words the RDR approach results in easy to understand rules of improved classification accuracy compared to the results of rule learning achieved in previous work.

1 INTRODUCTION

Lemmatization is an important preprocessing step for many applications of text mining. It is also used in natural language processing and many other fields that deal with linguistics in general. It also provides a productive way to generate generic keywords for search engines or labels for concept maps.

Lemmatization is similar to word stemming but it does not require to produce a stem of the word but to replace the suffix of a word, appearing in free text, with a (typically) different word suffix to get the normalized word form. For instance, the suffixes of words *working*, *works*, *worked* would change to get the normalized form *work* standing for the infinitive: *work*; in this case, both the normalized word form and the word stem are equal. Sometimes the normalized form may be different than the stem of the word. For example, the words *computes*, *computing*, *computed* would be stemmed to *comput*, but their normalized form is the infinitive of the verb: *compute*.

A lot of work has been carried out in word lemmatization and stemming, particularly for English, e.g., the well-known Porter stemmer [1]. But except for few research prototypes [2, 3] no algorithm is readily available for lemmatization of Slovene words. The main difficulty of word lemmatization of Slovene is that Slovene is a highly inflected natural language, having up to 30 different word forms for the same normalized word.

The paper proposes a novel word lemmatization algorithm, inspired by well known Ripple Down Rule (RDR) induction algorithms [4, 5]. By focusing on word endings the induced rules determine which word suffix should be removed and/or added to get the normalized form of a word.

The RDR learning algorithm can be applied to a lexicon of words in which normalized forms have been defined. The algorithm is not biased to Slovene; it could be applied to lexicons of English words or words in other languages without modifications. Once the lemmatization rules have been learned, the rules can be used as a classifier for unseen words for which a normalized form needs to be generated. Moreover, the rules can be applied to lemmatizing words in a free text, after having transformed the text into a list of words.

The paper presents the problem of lemmatization of words from Slovene free text and explains why the Ripple Down Rules (RDR) approach is very well suited for the task, showing that on the task of learning from a corpus of lemmatized Slovene words the RDR based approach results in easy to understand rules of improved classification accuracy compared to the results of rule learning achieved in previous work. Section 2 defines the problem of lemmatization of words from Slovene free text. Section 3 explains why Ripple Down Rules are appropriate for solving this problem. The data and the experimental setting are presented in Section 4 and the results of experiments in Section 4.3. The paper concludes with a discussion and plans for further work.

2 DESCRIPTION OF THE PROBLEM

To understand how lemmatization works, one should know how different forms of a word are created. Most of the words change when they are used in different grammatical

forms. The ending of the word is replaced by a grammatical ending and this leads to a new form of the initial word.

When we refer to a verb in general, we usually use its normalized form as in the verb *walk*. We then add inflections to the normalized form as required. These inflections indicate the tense. The "s" inflection indicates the present tense, and the "ed" inflection indicates the past tense. Thus there would be one rule for each tense covering most of the cases and then one could add exceptions as it is usually done in the grammar of all natural languages.

Lemmatization will be the inverse transformation: replacement of the grammatical ending by the initial suffix. Consequently, we define lemmatization in the same way as defined in [2], as a replacement of a suffix (the grammatical ending) by another suffix (the ending of the normalized word).

2.1 Lemmatization as substitution of suffixes

The grammatical ending usually depends on the word ending. It means that two words with different endings will not have the same grammatical ending, even if they are used in the same grammatical form.

For example, the words *property* and *train* will not receive the same grammatical ending even if they are both used in their plural form. *Property* will lose its "y" and receive a suffix "ies" to become *properties* while *train* will receive the suffix "s" and change to *trains*. And this is also true when we try to recover the normalized form. The grammatical ending will inform us about the ending of the normalized form. The suffix "ies" of the word *properties* indicates that the normalized form ends with a "y". In the proposed approach to word lemmatization, the idea is to create rules to recover the initial suffix from the grammatical ending.

2.2 Lemmatization as a machine learning problem

As proposed in [2], each word is labeled by a class label, that represents the transformation that should be applied to get the normalized form of the word. To determine this class, a stem should be found first. It is the part the two words (the word and its normalized form) have in common. The words *property* and *properties* have both the stem "propert" in common. Or in Slovene, the words "BRESKEV" and "BRESKVAH" have "BRESK" in common. Then we can see that we should remove the suffix "VAH" from "BRESKVAH" and add the suffix "EV" to get the normalized form "BRESKEV". Thus we assign the class label "VAHtoEV" to the word "BRESKVAH". Our training set is made of words with their suffixes and class labels.

Let us now explain the selected class label form. The class label assigned to a word is defined by giving the transformation to be applied on the word in order to get the normalized form. The transformation is given in the form of map S1 to S2 (S1toS2), where S1 is a suffix of the word and S2 is a suffix of the normalized word. The class label "_to_" means that the word does not change.

As the grammatical endings for words may have different lengths, in our approach words are divided into suffixes of up to *n* letters, where *n* is the size of the word. For instance, the word "BRESKEV" is represented by the following sample: "V" "EV" "KEV" "SKEV" "ESKEV" "RESKEV" with the class value: "_to_".

3 RIPPLE DOWN RULE APPROACH

Initially, Ripple Down Rules (RDRs, [4, 5]) have been developed for knowledge acquisition and maintenance of rule-based systems. In knowledge acquisition and incremental rule learning, it is often hard to add new rules and certify that the adding of a rule will not cause the inconsistency of the rule base, causing the existing rules to perform badly in new classification tasks.

As opposed to standard classification rules, induced by using a covering algorithm for ruleset construction (such as AQ [6], CN2 [7] and ATRIS [8]), Ripple Down Rules create exceptions to existing rules, so that the changes are confined to the context of the rule and will not affect other rules. Ripple Down Rules resemble decision lists [9] which induce rules of the form "if-then-else", as new RDR rules are added by creating **except** or **else** branches to the existing rules. If a rule fires but produces an incorrect conclusion then an **except** branch is created for the new rule. If no rule fires then an **else** branch is created for the new rule. Take a simple Ripple Down Rule:

if a ^ b **then** c
    except if d **then** e
else if f ^ g **then** h

The rule is interpreted as "if a and b are true then we conclude c unless d is true. In that case we conclude e. If a and b are not true then we continue with the other rule and conclude h, if f and g are true." This rule form fits very well the problem of lemmatization.

To create an exception to a rule, the algorithm should first recover the word that induced the rule that fired. Then the differences between the two words are calculated. The conditions of the exception rule will correspond to these differences. Suppose that the current rules that the algorithm has constructed are the following:

if A **then** _to_ **because of** MINA
else if H **then** VAHtoEV **because of** BRESKVAH
endif

and that the new word "BRESKVAMA" is presented to the algorithm, then a new rule will be added as follows:

if A **then** _to_ **because of** MINA
    except if VAMA **then** VAMAtoEV **because of** BRESKVAMA
    end except
else if VAH **then** VAHtoEV **because of** BRESKVAH
endif

The grammar of a language is made of rules that cover almost all examples and some exceptions to these rules. For example, in English, there would be a rule that says: the suffix "ed" is added to every verb when used in the past tense and some exception rules would be: unless the verb ends with a "y", the suffix "ied" is added. All the auxiliary verbs are exceptions to the rule of the "ed" suffix as well. Therefore we need a system that first creates the most general possible rules and then adds exceptions for more specific cases.

4 EXPERIMENTS

4.1 The data

Five datasets taken from [2], were obtained as random samples of different size taken from a large hand-constructed lexicon MULText-EAST [10]. The whole lexicon contains about 20 000 different normalized words with listed different forms for each of them resulting in about 500 000 different entries (potential learning examples). For each word its normalized form is provided and additionally, the word is annotated by some information about the word form such as singular/plural, noun/verb/adjective, etc. Since our motivation is to develop an approach that can be used on any text in the selected natural language and this additional information is not readily available in the usual texts, we are not using it here. However, one can argue that if a language tagger is available and the text is in the appropriate form to apply it, this additional information could be very valuable.

4.2 The experimental setting

We have done different experiments to expose the ability of Ripple Down Rules to perform lemmatization. We used the datasets presented before to provide a comparison with classification rules induced by the ATRIS rule induction algorithm [8, 2]. The five original datasets, used in [2], contain 160, 920, 1890, 3820, 5720 samples randomly taken from the lexicon, respectively.

For each of the five datasets, we report results of classification accuracy achieved in 5-fold cross-validation, where we used exactly the same examples. We used subsets of the whole labeled data set split randomly into 5 folds. The whole training-testing loop was thus repeated five times, with each of the five subsets being excluded from training and used as testing exactly once (cross-validation).

4.3 Results of experiments

The results of all experiments reported in this section are shown in Figure 1 and Table1.

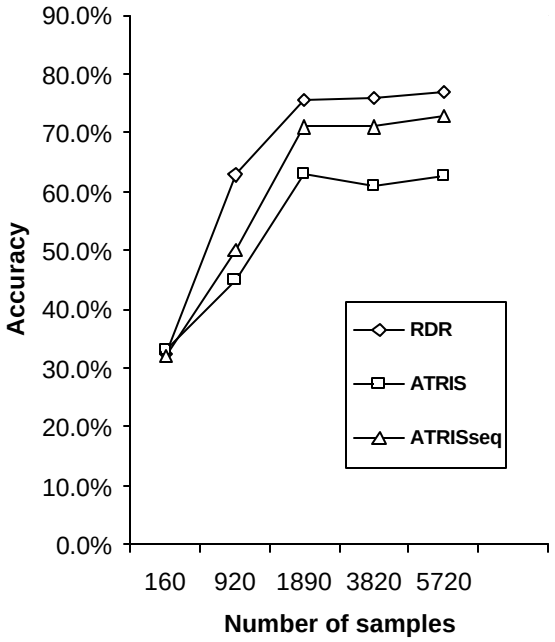

Figure 1: *Results of 5-fold cross-valdation for each of the five datasets having different number of samples.*

	Original approach	Sequential modeling
ATRIS	62.6 ± 0.07	72.8 ± 0.7
RDR	77.0 ± 0.6	77.0 ± 0.6

Table 1: *Classification accuracy with 5-fold cross-validation on dataset with 5720 samples.*

The samples were presented to the algorithm in a random order. As in previous work using ATRIS, we can see (Figure 1) that the variation in the performance on different sample sizes was small, after having included at least 2000 examples. For the same maximal number of samples presented, RDR achieved 77.0% accuracy. We can attribute this improvement to the construction of the general rules first and then of more specific rules. It seems that we can cover more examples with less training. That is why we get better accuracy even with fewer examples presented. The improvement may be also attributed to the construction of RDR where we keep all the cases covered by the rules in order to build reliable exceptions.

When looking at the results of previous work [2], we see that the best accuracy is obtained when using *Sequential*

modeling [11]. Sequential modeling is used to preformat the training data in order to get words grouped according to their last letter. Then rules are generated for each group of words with the same ending. Namely, instead of taking all the training examples with 486 different class values, a one level decision tree (decision stump) is generated that divides the examples into subsets depending on the last letter in the word. In this way the problem is decomposed into 20 independent subproblems, each having between 1 and 40 class values (most of the subproblems have less than 10 class values) instead of 486 class values.

We should note that the architecture of the rules induced by our algorithm already handles sequential modeling because rules are based on suffixes with increasing lengths, which means that the first rules are always based on the last letter only. Thus we get exactly the same results for the two experiments.

5 DISCUSSION

The experiments have confirmed that the architecture of Ripple Down Rules is appropriate for the problem of lemmatization. We believe that these results may be improved by using grammatical information about the words as is done in [3] whose results are better (92%) but they are using much more (grammatical) information and they are learning separate rules for verbs, nouns, adjectives, ... As the training and testing conditions are not the same, the results are not comparable and our goal was to improve the accuracy on free text without any grammatical information. Hence, we are only able to compare the results of RDR with the results previously achieved by the ATRIS rule learner, using the same experimental setting. But as we expect the creation or improvement of taggers for Slovene, grammatical information will become more and more available and valuable. Then the creation of a new system capable of creating more precise rules for Slovene will be possible. We also expect to obtain rules with a strong linguistic meaning, where the grammar will be presented clearly, just as the grammatical rules of a language usually are. The goal is to create a system for both lemmatization performance and linguistic knowledge extraction.

Acknowledgements

This work was supported by the PASCAL 6FP Network of Excellence on Pattern Analysis, Statistical Modelling and Computational Learning and AVLIS 6FP STREP project on Superpeer Semantic Search Engine. The authors are grateful to Tomaž Erjavec and Sašo Džeroski for the discussions on the topic of this paper. Tomaž Erjavec also provided the MULText-EAST lexicon used as a source of training examples in the experiments with RDR.

References

[1] M. Porter. An Algorithm for Suffix Stripping. *Proc. ACM SIGIR Conference on Conference on Research and Development in Information Retrieval*. 1980.

[2] D. Mladenic. Learning Word Normalization Using Word Suffix and Context from Unlabeled Data. *Proc. ICML*. 2002.

[3] S. Džeroski, T. Erjavec. Machine Learning of Morphosyntactic Structure: Lemmatising Unknown Slovene Words. *Applied Artificial Intelligence*. 2004.

[4] A. Srinivasan, P. Compton, R. Malor, G. Edwards, C. Sammut, L. Lazarus. Knowledge Acquisition in Context for a Complex Domain. *Proc. European Knowledge Acquisition Workshop*. Aberdeen. 1991.

[5] Y. Mansuri, J. G. Kim, P. Compton, C. Sammut. An evaluation of Ripple-Down Rules. *Proceedings of the IJCAI'91 Knowledge Acquisition Workshop* Pokolbin. 1991.

[6] R. S. Michalski, I. Mozetic, J. Hong, N. Lavrac. The multi-purpose incremental learning system aq15 and its testing application on three medical domains. *Proc. of the 5th National Conference on Artificial Intelligence*. 1986.

[7] P. Clark and T. Niblett. The cn2 Induction Algorithm. *Machine Learning*. 1989.

[8] D. Mladenic. Combinatorial Optimization in Inductive Concept Learning. *Proc. of ICML*. 1993.

[9] R. L. Rivest. Learning Decision Lists. *Machine Learning*. 1987.

[10] T. Erjavec. The multext-east Slovene Lexicon. *Proc. of the 7th Slovene Electrotechnical Conference ERK*. 1998.

[11] Y. Even-Zohar, D. Roth. A Sequential Model for Multi-class Classification. *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2001.