Journal of Universal Computer Science, vol. 16, no. 9 (2010), 1190-1214
submitted: 21/1/10, accepted: 28/4/10, appeared: 1/5/10 © J.UCS

LemmaGen: Multilingual Lemmatisation with Induced Ripple-Down Rules

Matjaž Juršič
(Jožef Stefan Institute, Ljubljana, Slovenia
matjaz.jursic@ijs.si)

Igor Mozetič
(Jožef Stefan Institute, Ljubljana, Slovenia
igor.mozetic@ijs.si)

Tomaž Erjavec
(Jožef Stefan Institute, Ljubljana, Slovenia
tomaz.erjavec@ijs.si)

Nada Lavrač
(Jožef Stefan Institute, Ljubljana, Slovenia and
University of Nova Gorica, Nova Gorica, Slovenia
nada.lavrac@ijs.si)

Abstract: Lemmatisation is the process of finding the normalised forms of words appearing in text. It is a useful preprocessing step for a number of language engineering and text mining tasks, and especially important for languages with rich inflectional morphology. This paper presents a new lemmatisation system, LemmaGen, which was trained to generate accurate and efficient lemmatisers for twelve different languages. Its evaluation on the corresponding lexicons shows that LemmaGen outperforms the lemmatisers generated by two alternative approaches, RDR and CST, both in terms of accuracy and efficiency. To our knowledge, LemmaGen is the most efficient publicly available lemmatiser trained on large lexicons of multiple languages, whose learning engine can be retrained to effectively generate lemmatisers of other languages.

Key Words: rule induction, ripple-down rules, natural language processing, lemmatisation
Category: I.2.6, I.2.7, E.1

1 Introduction

Lemmatisation is a valuable preprocessing step for a large number of language engineering and text mining tasks. It is the process of finding the normalised forms of wordforms, i.e. inflected words as they appear in text. For example, in English the lemma of wordforms *dogs* is *dog*, of *wolves* is *wolf*, of *sheep* is *sheep*, of *looking* is *look* and of *took* is *take*.

Traditionally, lemmatisation rules were hand-crafted. However, machine learning approaches to morphological analysis and lemmatisation became an increasingly interesting research subject. Machine learned lemmatisers are robust and can handle out-of-vocabulary words; they can be trained on large datasets and thus have large coverage; and it is easier and cheaper to acquire training datasets than rules. For example, a researcher in multilingual language processing who needs the functionality of lemmatisation can acquire lexicons for many languages via the European Language Resources Association ELRA or the U.S. based Linguistic Data Consortium LDC, train the learner on them, and use the resulting lemmatisers for text processing.

The goal of this work was to construct an efficient multilingual lemmatisation engine and to make it publicly available under the GNU open source license. To this end, we have developed a system, named LemmaGen, consisting of a learning algorithm for automatic generation of lemmatisation rules in the form of Ripple Down Rules (RDR) [Compton and Jansen 1990, Srinivasan et al. 1991] and an algorithm for efficient lemmatisation using the generated rules.

The proposed LemmaGen approach assumes the availability of a lexicon of lemmatised words. The main idea of our approach is: lemmatise a new—so far unseen—wordform in the same way as the most similar wordform in the lexicon was lemmatised. This can be achieved by *transforming* the problem of *lemmatisation* to the problem of *classification*: How to find the correct *class* (transformation) for the current wordform. Correspondingly, the lemmatisation problem is translated to a problem of finding the most appropriate class, i.e. , the one of the most similar wordform in the lexicon from which the lemmatiser is trained.

What remains is to define an appropriate measure of similarity between two wordforms. As a measure of similarity we chose the length of the suffixes shared by the two wordforms. For example, the similarity of wordforms *computable* and *compute* is 1 (only the last *-e* is the same), while the similarity of wordforms *computable* and *permutable* is 6 since they both share the *-utable* suffix. This notion of similarity makes sense because our target languages (mostly) belong to the Indo-European language group where inflection is commonly expressed by suffixes. In these languages the words that have similar suffixes usually behave alike when inflected, and as a consequence, their wordforms are similar when considering suffixes.

The concept of lemmatisation by similarity can be extended to several additional strategies, which in general increase the accuracy. Here we state just the two included in our approach. First, one must resolve the ambiguity when an unknown wordform has equal similarity to more than one wordform from the lexicon. Considering there is no additional information, we choose the most frequent class from the set of all similar wordform classes. In other words, we complement the similarity measure of wordforms with the class frequency. Second, it can also happen that one wordform has equal similarity to two wordforms with the same class frequency. In this case we look for the second most similar wordform to decide which class to prefer. These are some of the modifications of the similarity measure which proved to improve the overall accuracy of our lemmatisation approach.

The main advantages of the developed learner and multilingual lemmatiser are the following. Firstly, LemmaGen implements a general, nonincremental

RDR algorithm, specially tuned to learn from examples with a string-like structure. Secondly, it is a language independent learning engine, at least for inflectional languages. Consequently, it has already been applied to generate lemmatisers for twelve different European languages. Next, the format of induced rules is human readable and can be executed very efficiently (in constant time) when used for lemmatisation of new words/texts. Finally, the learning algorithm and the induced lemmatisers have been made freely available under the GNU open source license (downloadable from http://kt.ijs.si/software/LemmaGen/v2/).

This paper presents the LemmaGen learning engine and lemmatisers for twelve European languages, induced from eight Multext-East [Erjavec 2004] and five Multext (Multilingual Text Tools and Corpora) lexicons [Ide and Véronis 1994] - English appears in both sets. [Section 2] introduces the definitions and the lemmatisation problem. In [Section 3] we describe the LemmaGen input and output formats, aimed at generating rules in an understandable decision structure which enables efficient lemmatisation. [Section 4] describes how this rule structure is automatically constructed from the lemmatisation examples by LemmaGen. [Section 5] shows how the lemmatisation using constructed rules is performed. In [Section 6] we describe the application of LemmaGen to 13 different datasets, compare the results with two other publicly available lemmatisation rule learners, and evaluate their performance in terms of lemmatisation accuracy, efficiency, and applicability of the approach to different languages. [Section 7] briefly discusses the implementation and availability of the algorithm and the results. We conclude in [Section 8] with some conclusions and perspectives for further research.

2 Background and Related Work

We first introduce some definitions. A *wordform* is the (inflected) form of the word as it appears in a running text, e.g., *wolves*. This wordform can be morphologically analysed into its stem *wolf-* and ending *-s*. As evident from the example, phonological and morphological factors can influence how the abstract stem and ending are combined to arrive at the wordform. These factors are especially complex in languages with heavy inflection, such as Slovene and other Slavic languages, where stems can combine with many different endings, in a many-to-many relation, and the selection of the appropriate ending for a given stem and how they combine into a wordform can depend on a whole range of factors, from phonological to semantic. Two methods are typically used to abstract away from the variability of wordforms in preprocessing of texts (e.g., text mining, search, information extraction and retrieval): stemming and lemmatisation. These two methods are introduced in this section, followed by an outline of the related work.

2.1 Stemming and Lemmatisation

The first method, *stemming*, is popular in information extraction and retrieval, and essentially reduces a wordform to an invariant stem that semantically identifies it. This method often collapses different word-classes (parts-of-speech) and does not, in general, produce a surface form. So, for example, the wordforms *computer, computing, computes, computable, computed* would all be typically stemmed to *comput*.

The second method, *lemmatisation*, transforms a wordform to its canonical (normalised) form, the lemma, which corresponds to a headword in a dictionary. This canonical form is a particular wordform which, by convention, serves to identify an abstract word. The distinction between stems and lemmas is not so important in English, where the stem is often identical to the lemma, but is much more obvious in e.g. Slavic languages. For example, in English the lemma of the wordform *dogs* is *dog*, of *wolves* is *wolf*, and of *sheep* is *sheep*. On the other hand, in Slovene, for example, the wordform *ovce* (genitive of *sheep*) has the stem *ovc-*, while the lemma form, by convention the nominative singular, is *ovca*. In contrast to stemming, lemmatisation is more selective (a single stem can have more than one lemma, e.g., verbal and nominal) and results in an intuitively understandable form of the word. It is also more difficult: not only does the word ending need to be removed from the wordform, but, in general, the correct ending corresponding to the lemma has to be added, as illustrated by the examples we used for stemming: the wordforms *computing, computes, computed* should be lemmatised to *compute*; *computer* to *computer*; and *computable* to *computable*.

Lemmatisation is also faced with the problem of ambiguity: a wordform, and especially that of an unknown word can have multiple possible lemmas. So, for example, the Slovene wordform *hotela* can be lemmatised as *hotel* (the noun *hotel*), or *hoteti* (the verb *to want*). Which is the correct lemma depends on the context that the wordform appears in. The task of morphosyntactic disambiguation (i.e. , determining if a certain wordform in the text is a noun or a verb, and also its other inflectional properties) is the domain of part-of-speech taggers, or, more accurately, morphosyntactic taggers. By using the information provided by such a tagger, a lemmatiser is in a much better position to correctly predict the lemma form. However, as shown in this paper, even without such information, a lemmatiser can still achieve a high lemmatisation accuracy. Moreover, since our approach does not need morphosyntactic information, it can be used also to lemmatise texts that are not part of complete sentences (contents of Short Message Service, web queries, etc.).

2.2 Related Work

The problem of stemming and lemmatisation was already addressed in the 1960's [Beth 1968]. Traditionally, hand-crafted morphological analysers (which, as a side-effect, could also produce lemmas of wordforms) have been developed for a number of languages. Due to the existence of high-coverage, precise and fast hand-crafted analysers, such as the well-known Porter stemmer [Porter 1990] which is considered a de facto standard for English, and its successor, the multilingual Snowball stemmer (http://snowball.tartarus.org/), stemming and lemmatisation were often taken as solved problems. However, these systems (often using various methods such as finite state automata or transducers to compress hand-crafted rules into a resulting language model) have several shortcomings: lemmatisers do not do well on out-of-vocabulary words, and they are expensive to construct. There are still languages without such an infrastructure, they are difficult to adapt to language varieties, and are quite often not publicly available, such as the stemmer for Slovene described in [Popovič and Willett 1990].

Traditional hand-coded rules in grammars of natural languages typically obey the Elsewhere condition [Kiparsky 1973]: "In cases where more than one rule is applicable, the most specific rule should apply". Hence, traditional hand-coded lemmatisation rules are ordered, with exceptions coming first, followed by more general rules. This principle has been followed also in most machine learning approaches to learning lemmatisation rules: a rule induction system ATRIS [Mladenić 1993, Mladenić 2002a], if-then classification rules and Naive Bayes [Mladenić 2002b], a first-order rule learning system CLOG [Erjavec and Džeroski 2004], the CST lemmatiser [Dalianis and Jongejan 2006], and the Ripple Down Rule (RDR) learning approach [Plisson et al. 2008].

The last three of these systems deserve special mention. The first-order decision list learner CLOG [Manandhar et al. 1998], described and evaluated in [Erjavec and Džeroski 2004, Džeroski and Erjavec 2000, Erjavec and Sárossy 2006], relies on having information from a part-of-speech (POS) tagger; at a cost of lower efficiency, POS tagging information allows CLOG to attain a high accuracy (note, however, that such a tagger is not available for all languages). The CST lemmatiser [Dalianis and Jongejan 2006] is one of the few trainable lemmatisers that is available for download (from http://www.cst.dk/online/lemmatiser/uk/) and we were therefore able to directly compare its results with the results obtained by our lemmatiser, LemmaGen. Another publicly available lemmatiser, RDR [Plisson et al. 2008], was—like LemmaGen—inspired by the Ripple Down Rule learning methodology for the GARVAN-ES1 expert system maintenance [Compton and Jansen 1988], where the idea was that new rules are incrementally added to the system when new examples of decisions are available. However, new examples might contradict already existing rules, therefore exceptions to the original rules have to be added as well. When executed, rules are 'fired' top-down until the most specific applicable rule fires, thus obeying the Elsewhere principle mentioned above.

To the best of our knowledge, the LemmaGen machine learning approach, proposed in this paper, results in the most efficient publicly available lemmatisers trained on large lexicons of several languages. Note also that the LemmaGen learning engine can be retrained on additional languages, and could, most likely, be generalised to deal with other ML problems involving string processing.

3 Knowledge Representation

This section describes the LemmaGen input (training examples) and output (lemmatisation rules) data structures.

3.1 Representation of Training Examples

The training data for lemmatisation is generally represented in the form of pairs *(Wordform, Lemma)*. However, for the sake of training by LemmaGen, training examples are actually represented as pairs *(Wordform, Class)* where the *Class* label is the transformation which replaces the wordform suffix by a suffix of the lemma [see Table 1]. It is worth pointing out that our method can be used also for stemming. One only has to replace the training pairs *(Wordform, Lemma)* with *(Wordform, Stem)*. Thus, even though we concentrate on lemmatisation, one can easily switch to stemming by changing the training data.

			Class
	Wordform	Lemma	(WordformSuffix --> LemmaSuffix)
English			
1	dogs	dog	("s" --> " ")
2	wolves	wolf	("ves" --> "f")
3	sheep	sheep	(" " --> " ")
4	looking	look	("ing" --> " ")
5	took	take	("ook" --> "ake")
Slovene			
1	brat	brati	(" " --> "i")
2	brala	brati	("la" --> "ti")
4	beri	brati	("eri" --> "rati")
4	berete	brati	("erete" --> "rati")

Table 1: Examples of class labels which are used to form training examples *(Wordform, Class)* for lemmatisation of English and Slovene words. Note a great variability of wordform endings for a single Slovene word *brati* (meaning *to read* in English).

[Table 1] illustrates the variability of suffixes of wordforms of Slovene, a language with high inflectional complexity. Note, however, that even in English there are many different suffixes of wordforms sharing the same stem. [Table 2] lists a selected set of wordforms from the Multext English lexicons (used in the experiments of [Section 6]) for lemmas starting with string *writ-*. The entries in the lexicon have the form of triplets *(Wordform, Lemma, MSD)*, where MSD stands for the morphosyntactic description of the wordform, i.e. a feature structure giving the part-of-speech and other morphosyntactic attributes of the wordform.

3.2 Representation of Lemmatisation Rules

This subsection describes the output of the learning algorithm. We first describe the standard Ripple Down Rule (RDR) structure and show how lemmatisation

Wordform	Lemma	MSD
write	write	`Vvb---`
write-off	write-off	`Ncns-`
write-offs	write-off	`Ncnp-`
writer	writer	`Ncfs-`
writer	writer	`Ncms-`
writers	writer	`Ncfp-`
writers	writer	`Ncmp-`
writes	write	`Vvfps3`
writhe	writhe	`Vvb---`
writhe	writhe	`Vvfpp-`
writhe	writhe	`Vvfps1`
writhe	writhe	`Vvfps2`
writhed	writhe	`Vvfs--`
writhed	writhe	`Vvps--`
writhes	writhe	`Vvfps3`
writhing	writhe	`Vvpp--`
writing	write	`Vvpp--`
writing	writing	`Ncns-`
writings	writing	`Ncnp-`
writs	writ	`Ncnp-`
written	write	`Vvps--`
wrote	write	`Vvfs--`

Table 2: Triplets of the form *(Wordform, Lemma, MSD)*, where MSD stands for the wordform morphosyntactic description. This is the set of all lemmas starting with *writ-*, as they appear in the Multext English lexicon (used in the experiments of [Section 6]).

rules can be expressed using it. Then, a description of our refinement of the RDR structure is given.

3.2.1 Standard RDR Format

Ripple Down Rules (RDRs) [Compton and Jansen 1990, Srinivasan et al. 1991] were originally used for *incremental* knowledge acquisition and maintenance of rule-based systems. Compared to standard if-then classification rules, RDRs resemble decision lists of the `IF-THEN-ELSE` form [Rivest 1987]. However, the main idea of RDRs is that the most general rules are constructed first, and later, as counter examples are encountered, exceptions to the rules are added (`EXCEPT` branches) in an iterative, incremental rule building process. Consequently, RDRs form a tree-like decision structure: rules and their exceptions are ordered, and the first condition that is satisfied and has no exceptions, fires the corresponding consequent. [Figure 1] shows a simple Ripple Down Rule describing flying properties of birds and objects.

```
IF bird THEN flies EXCEPT
    IF young bird THEN doesn't fly
    ELSE IF penguin THEN doesn't fly EXCEPT
       IF penguin in airplane THEN flies
ELSE IF airplane THEN flies
```

Figure 1: A simple Ripple Down Rule structure.

A small - but realistic - RDR for wordform lemmatisation, constructed incrementally from examples in [Table 2], is shown in [Figure 2]. The individual rules in the RDR structure are ordered and need to be interpreted sequentially. A valuable feature of RDR rules is also that one can attach examples to individual clauses. For instance, in a rule such as

```
IF suffix("ote") THEN transform("ote" -->"ite") EG wrote
```

the additional `EG` keyword lists an example from the training set that caused the creation of the decision branch. This feature of RDRs turns out to be helpful for better understanding of complex rules.

```
1      IF suffix("") THEN transform(""-->"") EXCEPT
1.1        IF suffix("ote") THEN transform("ote"-->"ite")
1.2        ELSE IF suffix("ten") THEN transform("ten"-->"e")
1.3        ELSE IF suffix("s") THEN transform("s"-->"")
1.4        ELSE IF suffix("g") THEN transform(""-->"") EXCEPT
1.4.1          IF suffix("hing") THEN transform("ing"-->"e")
1.5        ELSE IF suffix("d") THEN transform("d"-->"")
1.6        ELSE IF suffix("e") THEN transform(""-->"")
1.7        ELSE IF suffix("r") THEN transform(""-->"")
1.8        ELSE IF suffix("f") THEN transform(""-->"")
1.9        ELSE IF suffix("t") THEN transform(""-->"")
```

Figure 2: A RDR structure, constructed by incremental learning algorithm RDR [Plisson et al. 2008] from examples in [Table 2].

3.2.2 Refined RDR Format

In the case of learning lemmatisation rules, training examples in the form of large lexicons are readily available. Therefore, there is no need for incrementally adding exceptions while maintaining the initial, general rules. A compact RDR structure, including all the exceptions, can be computed by a non-incremental algorithm. In this subsection we show how the original RDR representation can be refined in order to improve the readability and efficiency of RDRs for lemmatisation.

Focusing on a single if-then rule inside the rule structure in [Figure 2] we can see that it has the following form:

– A rule condition is the suffix of a wordform which fires the rule, e.g. *ote* in Rule 1.1.

– A rule consequent is a class, i.e. , a transformation of the form *Wordform-Suffix* `-->` *LemmaSuffix*, e.g. *ote* `-->` *ite* in Rule 1.1.

Utilising the original RDR structure for lemmatisation has some disadvantages considering our two main objectives, efficiency and readability. Firstly, efficiency suffers because one is not able to directly detect which exception applies to the wordform that is currently being lemmatised. Therefore, at each level of the tree one must test conditions of all the exceptions until either the one that applies is found or the end of the list is encountered. These lists of exceptions can be long (i.e. , up to 200 for real lexicons) and their processing can take a relatively large amount of time. The readability also suffers due to the long lists of exceptions and sequential triggering of rules. For example, in [Figure 2], Rule 1.6 is more general then Rule 1.1 and has to be checked *after* Rule 1.1 for correct interpretation.

To solve the problems of efficiency and readability we have refined the original RDR structure, illustrated in [Figure 3]. Note that, from now on, in the `IF-THEN-ELSE` rule format we will ommit the `ELSE` keyword in order to increase the readability of rules.

The refined RDR structure was achieved by imposing equal-similarity constraint that solves both problems and can be elegantly expressed using the similarity measure defined in [Section 1]: the similarity among conditions (suffixes) inside one exception list must be the same for all possible pairs. For instance, in [Figure 3], the similarity among conditions of the exception list of Rule 1 (*-d*, *-ote*, *-ing*, *-ten*, *-s*) is zero. On the other hand, if we focus on the exception list of Rule 1 in [Figure 2] we can notice that in the RDR structure the equal-similarity constraint is violated: similarities among different suffixes are not the same: while the similarity between suffixes 1.1 *-ote* and 1.2 *-ten* is 0, the similarity between 1.1 *-ote* and 1.6 *-e* is 1.

```
1      IF suffix("") THEN transform(""-->"") EXCEPT
1.1        IF suffix("d") THEN transform("d"-->"")
1.2        IF suffix("ote") THEN transform("ote"-->"ite")
1.3        IF suffix("ing") THEN transform("ing"-->"e") EXCEPT
1.3.1         IF suffix("ting") THEN transform(""-->"")
1.4        IF suffix("ten") THEN transform("ten"-->"e")
1.5        IF suffix("s") THEN transform("s"-->"")
```

Figure 3: A refined RDR tree structure, constructed by LemmaGen from English words in [Table 2].

Assertion.

Conditions in the same exception list of refined RDR structure satisfy the following constraints:

1. All the suffixes share the same ending ($k-1$ characters).
2. The first character that is different among suffixes (the k-th character from the right) is different for all the suffixes, therefore it disjunctively separates all the suffixes.

One can always choose such k that the first statement holds, e.g., if the suffixes share no common ending then k is set to 1. Subsequently, when k is identified we must only prove that the second statement holds; namely, that there are no two suffixes having the same character at the position k from the end. This assertion is proved by contradiction; if two such suffixes exist, the similarity between them is greater (at least k) than between the others ($k - 1$). This violates the equal-similarity constraint, therefore no such two suffixes exist. With this we have proved that the proposed assertion holds if the equal-similarity constraint is obeyed.

Using the above assertion we can derive the following properties of the refined RDR structure.

1. It can always be decided which sub-rule fires on a specific wordform by examining just one character (the k-th) of a wordform suffix. In combination with a proper implementation (e.g., a hash table), the RDR structure can be traversed very efficiently.

2. Exceptions are disjunctive. Consequently, there is always at most one sub-rule that fires for a specific wordform. Moreover, all the rules dealing with similar suffixes are grouped together; therefore the readability is considerably improved.

3. The maximal number of exceptions of one rule is limited by the number of characters in the alphabet. As conditions are disjunctive with respect to the character at the k-th position, the number of disjunctive conditions cannot be greater than the number of characters in an alphabet. For majority of European languages, this number is less than 30 (note that in the original RDR structure constructed by the incremental RDR learner there can be several hundreds of exceptions in constructed rules). This also contributes to improved efficiency and readability.

To summarise, the proposed refined RDR structure overcomes two weaknesses of original RDRs, improving their readability and enabling efficient execution in lemmatisation tasks.

4 The LemmaGen Learning Algorithm

This section describes the LemmaGen learning algorithm which learns lemmatisation rules in the form of a refined RDR structure. The learning algorithm is efficient and language independent since it can be used for training on new lexicons of lemmatised wordforms.

There are two main properties of the learning algorithm. Firstly, in contrast to most of the RDR algorithms it is not an incremental learner. Secondly, it is very efficient and has a low time complexity as compared to the original RDR learners. These two properties—non-incrementality and efficiency—are the core improvements of the LemmaGen learning algorithm.

The pseudocode of the algorithm is given in [Figures 4 and 5]: the top level function and recursive learning of a rule, respectively. To describe the algorithms, we use an object oriented representation of the recursive RDR structure. Each rule has three components: a condition, a class, and a (possibly empty) list of exceptions:

RDR ::= `IF` *rule.condition* `THEN` *rule.class* `EXCEPT` *rule.exceptions*

where

- *rule.condition* is a wordform suffix,
- *rule.class* is a transformation to be applied to a wordform, and
- *rule.exceptions* ::= nil | RDR+ (list of RDRs).

The input to learning is a list of training examples, and the output is a RDR structure. The top level function, *LearnRDR* [Figure 4], just sorts the examples by their wordforms with character strings reversed (line 1.2), and invokes recursive rule learning (function *LearnRecursive*) on the sorted list of examples (line 1.3).

```
1.1 function LearnRDR(examplesList)
1.2     sortedExamplesList = Sort(examplesList, 'reverse dictionary sort')
1.3     entireRDR = LearnRecursive(sortedExamplesList)
1.4     return entireRDR
```

Figure 4: Top level function of the LemmaGen learning algorithm.

The *LearnRecursive* function [Figure 5] is the core of the learning algorithm. The function assumes that the examples are sorted by their wordforms, and that each recursive invocation deals with example wordforms which share increasingly longer suffixes. The recursion stops when all the remaining wordforms in the *currentExamplesList* are equal or when there remains just a single example in this list. The *LearnRecursive* function is executed as follows:

- It first finds the common suffix of all examples from the *currentExampleList. commonSuffix* can be determined by just comparing the suffixes of the first and last wordforms (line 2.2) because the *currentExamplesList* is sorted.
- Next, it initialises the main variable *currentRule* to an empty RDR structure (line 2.4), assigns to its condition the *commonSuffix* (line 2.5), and to its class the most frequent class of the current set of examples (line 2.6). When determining the most frequent class (function *MostFreqClass*), it ignores such classes `X-->Y` in which `X` is more specific than the current *commonSuffix.*
- The list of exceptions (*currentRule.exceptions*) is constructed in the part of the pseudocode between lines 2.7 and 2.19.
    - Line 2.8 sets the index to start grouping of examples into subsets with longer common suffixes (longer than current *commonSuffix*). The subsets are identified iteratively in the **for** loop (line 2.9). These subsets will later be used to form new exceptions of the current rule.

- First, two adjacent wordforms are extracted from *currentExamplesList* (lines 2.10 and 2.11). Then the character which distinguishes the suffixes of the two wordforms needs to be determined. It is the first character left of the common suffix of the two wordforms; in the pseudocode, the position of this "distinguishing" character is determined by subtracting the length of the common suffix from the wordlength of each of the two wordforms (lines 2.12 and 2.13).

- Once the subgroup of examples with a longer common suffix is determined (through identifying the two wordforms with different "distinguishing" character - i.e. when the condition 2.14 is no longer satisfied), the exception to the current rule is constructed (2.17) by recursively calling the *LearnRecursive* procedure for the current group of examples (a subset of *currentExamplesList* from position *start* to current position *i*). Finally, the created exception is added to the list of exceptions of the *currentRule* (line 2.18).

The pseudocode is an abstraction of the actual implementation and does not deal with the details of boundary conditions. It also omits some heuristics which optimise the resulting RDR structure by taking into account subsumption between the rules.

```
2.1   function LearnRecursive(currentExamplesList)
2.2       commonSuffix = EqualSuffix(currentExamplesList.First().wordform,
                                  currentExamplesList.Last().wordform)
2.3       lengthCommonSuffix = StringLength(commonSuffix)
2.4       currentRule = RDR()
2.5       currentRule.condition = commonSuffix
2.6       currentRule.class = MostFreqClass(currentExamplesList)
2.7       currentRule.exceptions = nil
2.8       start = 1;
2.9       for i = 1 to Length(currentExamplesList)-1
2.10          wf1 = currentExamplesList[i].wordform
2.11          wf2 = currentExamplesList[i+1].wordform
2.12          charPosition1 = StringLength(wf1) - lengthCommonSuffix
2.13          charPosition2 = StringLength(wf2) - lengthCommonSuffix
2.14          if (GetChar(wf1,charPosition1) == GetChar(wf2,charPosition2))
2.15              continue
2.16          else
2.17              exceptionRule = LearnRecursive(currentExamplesList[start..i])
2.18              currentRule.exceptions.Append(exceptionRule)
2.19              start = i+1
2.20      return currentRule
```

Figure 5: Recursive function for learning a RDR structure (top level rule and all exception subrules) from a current list of examples.

Theoretical time complexity of the algorithm is O$(N \cdot M)$ where N is the number of all training examples in *examplesList* and M is the length of the longest wordform in the examples. There are two main parts of the algorithm which need to be considered for time complexity calculation. Firstly, there is lexicographic sorting of the examples (line 1.2 in [Figure 4]); the time complexity of sorting is in general O($N \cdot logN$), however, since we are dealing with strings of limited length we can use radix sort which has the time complexity of O($N \cdot M$). The second part refers to learning of a RDR (line 1.3), specifically the function *LearnRecursive(examplesList)*. The worst-case time complexity here is again O($N \cdot M$) - the algorithm tests each word in the list of examples (N) and each time progresses by at least one character, therefore at most M such repetitions can happen for a word. Thus, we conclude that also the overall time complexity is O($N \cdot M$). Furthermore, as M is usually fixed for a given language, it can be considered a constant and the actual time complexity of the learning is O(N), i.e. it is linear with respect to the number of examples.

5 The LemmaGen Lemmatisation Algorithm

This section describes the LemmaGen lemmatisation algorithm, which uses the refined RDR structure to assign lemmas to words of a given language. The efficiency of this algorithm is due to the design and compactness of the refined RDR structure.

In the following we present how the learned RDR structure is applied to the classification of new words during lemmatisation. [Figure 6] shows the pseudocode of the lemmatisation algorithm. The code is simple, although one point needs some elaboration: how to (effectively) choose the next exception (if it exists):

– First, the position of a character (*keyChar*) which disjunctively separates the exceptions inside the wordform is retrieved (line 1.3). *charPosition* is calculated by subtracting the length of the current rule suffix from the length of the wordform (line 1.2). In such a way we get the k-th character of the wordform (as defined in [Section 3.2.2]).

– Using the k-th character as the key, the exception is directly retrieved from the hash table *rootRule.exceptions* if it exists (lines 1.4 and 1.5). The key in the hash table is just one (distinguishing) character, while the value is the exception-sub-rule that corresponds to the given character.

– With this procedure one retrieves exception rules along the tree path until there are no more exceptions. When the last one (i.e. the most specific one) is found, it is used to lemmatise the wordform by applying the assigned transformation to it (line 1.8).

The worst-case time complexity of the lemmatising algorithm is O(M), where M is depth of the RDR tree, i.e. in the worst case the length of the longest wordform from the examples. However, since the longest training wordform is limited, lemmatisation can be performed in constant time O(1).

```
function Lemmatise(wordform, rootRule)
    charPosition = StringLength(wordform) - StringLength(rootRule.condition)
    keyChar = GetChar(wordform,charPosition)
    if (rootRule.exceptions.Exist(keyChar))
        exceptionRule = rootRule.exceptions.Get(keyChar)
        return Lemmatise(wordform, exceptionRule)
    else
        lemma = rootRule.class.Transform(wordform)
        return lemma
```

Figure 6: The lemmatisation function implements a recursive descent through the RDR structure until there is no exception to the current rule.

6 Learning Multi-lingual Lemmatisation Rules: LemmaGen Application to Multext-East and Multext data

This section describes how the LemmaGen learner was used to induce lemmatisation rules for all the languages for which training lexicons are available through EU projects Multext [Ide and Véronis 1994] (Multilingual Text Tools and Corpora) and Multext-East [Erjavec 2004] (Multext for Central and Eastern European Languages). The lexicons were used to automatically learn lemmatisers for different languages. The accurate and efficient publicly available lemmatisers for 12 European languages represent an important contribution of this paper.

6.1 Multext-East and Multext Lexicons

There were altogether 13 lexicons for 12 European languages available for learning (there are two different training sets available for English). The sizes and properties of these training sets are listed in [Table 3]. Some observations can be made from this table:

- The sizes of the lexicons differ very much across languages, and the expectation would be that the larger the lexicon (training set), the better the achieved lemmatisation accuracy.
- Each lexicon contains records in the form of triples *(Wordform, Lemma, MSD)*, where MSD stands for the wordform morphosyntactic description.
- The number of distinct MSDs for a language may seem to indicate its inflectional complexity; however, it should be noted that the differences in their numbers are also due to different MSD design principles employed for various languages. So, for example, Slovene has significantly more MSDs than Serbian or Czech, although the languages are inflectionally of comparable complexity. The main reason for this is a very detailed pronoun typology for Slovene, which leads to over one thousand MSDs for pronouns alone.
- The ratio of the number of wordforms against the number of distinct lemmas should indicate the size of inflectional paradigms of various languages, and also give an indication of their inflectional complexity. However, this is not

overall the case, as different principles were adopted in constructing the lexica: some languages (i.e. Multext languages, Slovene, Romanian) include the complete paradigms for each lemma, while the others include only entries of wordforms actually attested in a reference corpus.

- The number of lemmas per wordform and per wordform-MSD pair indicate the upper bound on lemmatisation accuracy. With the ratio greater than 1, a lemmatiser will be, even theoretically, unable to always generate the correct lemma given a wordform or wordform-MSD pair. As can be seen, this problem can be quite severe if the lemmatiser operates only on wordforms; in the worst case (Estonian) the ambiguity is over 15%. If the MSD is taken into account, some ambiguity still remains, but is, even in the worst case (Hungarian), under 1.4%.
- The percentage of entries where the wordform is identical to the lemma (WF=Lemma in [Table 3]) gives the accuracy of a baseline lemmatiser, which would simply assume that the lemma is always identical to the wordform. As can be seen, this approach would already give about 60% accuracy for English. However, with other languages the situation corresponds less well with intuitions, again due to the differences in design criteria discussed above. Lexicons including complete inflectional paradigms have a much lower percentage, as lemma-identical wordforms will tend to occur much more frequently in corpora than in paradigms. Especially surprising is the very low number for Spanish - inspection of the lexicon reveals that the lexicon contains a large number of verbs, which, in Spanish, have a large number of different wordforms, leading to under 3% of lemma-identical wordforms.

In summary, the statistics over the lexicons have as much to do with the decisions made in the design of the MSD sets and what wordforms to include in the lexica, as with the morphological complexity of the various languages.

6.2 Learners Used in the Experimental Evaluation of LemmaGen

We have compared the performance of LemmaGen to the performance of other available lemmatisation rule learners: CST [Dalianis and Jongejan 2006] and RDR [Plisson et al. 2008].

CST [Dalianis and Jongejan 2006] is one of the few trainable lemmatisers that is available for download, and therefore we were able to directly compare its results with the results obtained by our lemmatiser, LemmaGen. The other publicly available lemmatiser, RDR [Plisson et al. 2008] was—like LemmaGen—inspired by the Ripple Down Rule learning methodology [Compton and Jansen 1988], implementing the idea of iterative ruleset construction: new rules are incrementally added to the system when new examples of decisions are made available.

Language	Records	Wordforms	Lemmas	MSDs	Lemmas per WF	Lemmas per WF-MSD	WF= Lemma
Multext-East							
Slovene	557,970	198,083	16,352	2,083	1.0444	1.0008	5.75%
Serbian	20,294	16,809	8,355	906	1.0288	1.0025	25.08%
Bulgarian	55,200	40,708	22,790	338	1.1017	1.0058	28.92%
Czech	184,628	56,795	23,030	1,428	1.0464	1.0062	39.36%
English	71,784	48,309	27,343	135	1.0208	1.0007	**61.08%**
Estonian	135,094	89,128	46,747	642	**1.1540**	1.0079	24.88%
Hungarian	64,042	50,908	27,991	619	1.1219	**1.0138**	29.45%
Romanian	428,194	352,003	39,275	616	1.0453	1.0010	11.77%
Multext							
English	66,216	43,273	22,794	133	1.0184	1.0006	58.17%
French	306,795	231,734	29,319	380	1.0166	1.0014	11.71%
German	233,858	50,085	10,485	227	1.0319	1.0024	22.11%
Italian	145,530	115,614	8,877	247	1.0636	1.0042	6.47%
Spanish	510,709	474,150	13,232	264	1.0069	1.0010	**2.73%**

Table 3: Sizes and properties of the Multext-East and Multext lexicons, in terms of numbers of records, different wordforms, lemmas and MSDs, as well as average numbers of lemmas per wordform, lemmas per pair *(Wordform, MSD)*, and the percentage of records where the wordform is identical to the lemma.

6.3 Experimental Settings

For training and testing experiments we used 5-fold cross validation repeated 20 times. Five-fold cross validation is adequate for our experiments as it is estimated that on the average, for all the twelve languages, approximately 80% of all the words of a given language are present in the lexicon. The procedure is repeated 20 times to ensure the stability of statistical evaluation measures (e.g. average accuracy, standard deviation).

The three learning algorithm were tested in two different experimental settings. Recall that in the original lexicons the records are formed of triples *(Wordform, Lemma, Morphosyntactic description)*. The two settings differ in terms of whether they use the morphosyntactic descriptions (MSD) assigned to the pairs wordform-lemma or not.

– In the first experimental setting the MSDs were not used. This is the more difficult experimental setting, mimicking the situation in applications working on raw text data (e.g., web page analysis, information retrieval, document clustering and classification, etc.), where stemming and lemmatisation are standardly used in the preprocessing of text documents and in which data is usually not (manually or automatically) annotated by morphosyntactic tags. This realistic setting does not need, apart from tokenisation and case normalisation, any additional processing. However, the achieved accuracies are lower than those achieved in the second setting.

– In the second experimental setting, MSD information was used in the following way: for every language the available training set was split into separate training sets, one per MSD. For Slovene this amounts to 2,083 training sets, for Serbian 906 training sets, etc. (c.f. MSDs column of [Table 3]). While being more complex for experiments, due to the separation into numerous separate training sets, the learning task in each training sets is simpler. The main reason is that this separation by itself eliminates many ambiguities: the same wordform string may have several lemmas, but if these are separated into different training sets for different MSDs, the ambiguities are automatically resolved. In this experimental setting the overall classification accuracy is expected to be higher than in the first one. In practice, using MSD information means that, in order to lemmatise a text, it needs to be first tagged with such MSDs. We do not address this issue here, but see the experiments with CLOG [Erjavec and Džeroski 2004].

6.4 Experimental Results Without Using MSDs

In these experiments, only pairs *(Wordform,Lemma)* from the lexicons were used. LemmaGen was compared to RDR and CST.

6.4.1 Accuracy Testing

	Training Set			Test Set		
Language	RDR	CST	LemmaGen	RDR	CST	LemmaGen
Mult-East						
Slovene	95.5 ±0.04	97.7 ±0.02	97.7 ±0.02	78.4 ±0.21	78.9 ±0.21	**79.8** ±0.22
Serbian	94.3 ±0.14	97.8 ±0.08	**97.9** ±0.08	63.8 ±0.80	64.0 ±0.82	**65.3** ±0.77
Bulgarian	91.1 ±0.11	93.5 ±0.07	**93.7** ±0.07	68.7 ±0.52	69.3 ±0.51	**70.4** ±0.49
Czech	96.7 ±0.06	97.9 ±0.04	97.9 ±0.03	75.0 ±0.56	76.9 ±0.50	**78.3** ±0.51
English	97.7 ±0.05	98.8 ±0.03	98.8 ±0.03	89.1 ±0.36	90.4 ±0.34	**90.8** ±0.32
Estonian	87.1 ±0.08	89.6 ±0.07	**89.7** ±0.07	62.7 ±0.35	62.3 ±0.33	**63.3** ±0.32
Hungarian	90.2 ±0.07	91.8 ±0.06	**91.9** ±0.06	72.0 ±0.39	71.7 ±0.37	**72.3** ±0.37
Romanian	94.9 ±0.03	96.8 ±0.02	96.8 ±0.02	72.6 ±0.16	72.7 ±0.16	**73.1** ±0.16
Multext						
English	98.1 ±0.05	99.0 ±0.03	99.0 ±0.03	90.7 ±0.34	92.0 ±0.31	**92.4** ±0.29
French	96.7 ±0.04	98.8 ±0.02	98.8 ±0.02	86.4 ±0.22	86.4 ±0.22	**87.5** ±0.20
German	94.6 ±0.08	98.0 ±0.05	98.0 ±0.04	77.1 ±0.43	81.2 ±0.42	**82.7** ±0.43
Italian	93.8 ±0.05	95.6 ±0.04	**95.7** ±0.04	**80.6** ±0.27	79.8 ±0.26	80.5 ±0.25
Spanish	99.1 ±0.01	99.4 ±0.01	99.4 ±0.01	94.3 ±0.07	95.2 ±0.06	**95.4** ±0.06

Table 4: Comparison of accuracies achieved by RDR, CST and LemmaGen in the first experimental setting (without using the MSD information).

Results of the comparison are given in Table 4]. Some further explanations are required before analysing the results table.

- Accuracy: Lemmatisation assigns a transformation (*class*) to a wordform. If there are P correctly lemmatised wordforms, and N is the total number of wordforms, then $Acc = \frac{P}{N}$.

- Accuracy was averaged separately on training and test sets over 100 runs (20 times 5-fold cross validation). When constructing a test set we made sure that no two words with the same wordform and lemma appeared in both the training and test set in the same validation step.
- The pairwise differences in the accuracy were tested for statistical significance by a paired Wilcoxon signed-ranks test [Wilcoxon 1945]. When the highest accuracy is significantly different from the rest (for a p-value = 0.05) this is indicated by a bold value in the table.

6.4.2 Algorithm Ranking in Terms of Accuracy

The goal of our experimental evaluation is to verify a hypothesis that LemmaGen performs significantly better in comparison to RDR and CST, over multiple data sets. The widely used t-test is usually inappropriate and statistically unsafe for such a comparison [Demšar 2006].

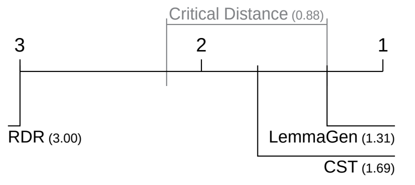
To compare two classifiers over multiple data sets, the Wilcoxon signed-ranks test [Wilcoxon 1945] is recommended. However, in our case we want to compare three classifiers in terms of their accuracies. To test the significance of differences between multiple means, a common statistical method is the well-known ANOVA [Fisher 1959] and its non-parametric counterpart, the Friedman test [Friedman 1940].

We applied the Friedman test and its corresponding Bonferroni-Dunn [Dunn 1961] post-hoc test. The Friedman test ranks the algorithms for each data set separately, the best performing algorithm getting the rank of 1, the second best rank 2, etc. In the case of ties (we used accuracies computed to a precision of just one decimal point), average ranks are assigned. The Friedman test then compares the average ranks of the algorithms. The null-hypothesis states that all the algorithms are equivalent and so their ranks should be equal. If the null-hypothesis is rejected, we can proceed with a post-hoc test.

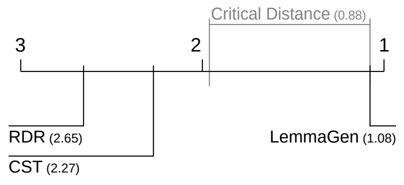
- The Nemenyi test [Nemenyi 1963] should be used if all the classifiers are to be compared to each other.
- In our case, however, we want to compare other classifiers (RDR and CST) to our control classifier (LemmaGen). Consequently, the Bonferroni-Dunn [Dunn 1961] test is used since it has a much greater power when all classifiers are compared only to a control classifier and not between themselves [Demšar 2006].

The Bonferroni-Dunn test computes the critical distance (in our case 0.88 for given p-value 0.05) between a classifier and the control classifier, and concludes that the accuracy of the two classifiers is significantly different if the corresponding average ranks differ by at least the critical distance.

The results of the Bonferroni-Dunn post-hoc tests are graphically represented by a simple diagram. [Figure 7] shows the two results of the analysis of the



(a) Training Set



(b) Test Set

Figure 7: Visualisation of Bonferroni-Dunn post-hoc tests for the first experiment (without MSDs), using average ranks of algorithms on data from [Table 4].

accuracies from [Table 4]. On the axis of each diagram we plot the average rank of the algorithms. The lowest (best) ranks are to the right. We also show the critical distance on the top, and connect the algorithms that are not significantly different. From the results we can draw two conclusion. On the training sets, LemmaGen is significantly better then RDR, but not significantly different from CST. However, on the test sets, LemmaGen is in terms of accuracy significantly better than both CST and RDR.

6.5 Experimental Results When Using MSDs

In these experiments, MSD information was used, and datasets were separated by language and by MSD. LemmaGen was again compared to RDR and CST.

6.5.1 Accuracy Testing

	Training Set			Test Set		
Language	RDR	CST	LemmaGen	RDR	CST	LemmaGen
Mult-East						
Slovene	99.8 ±0.01	99.9 ±0.00	99.9 ±0.00	93.2 ±0.16	91.1 ±0.18	**93.4** ±0.16
Serbian	99.7 ±0.03	99.7 ±0.03	**99.8** ±0.02	85.2 ±0.60	83.0 ±0.63	**86.1** ±0.61
Bulgarian	99.4 ±0.02	99.5 ±0.02	99.5 ±0.02	93.7 ±0.22	84.0 ±0.40	**94.1** ±0.21
Czech	99.4 ±0.03	99.4 ±0.03	**99.5** ±0.03	90.0 ±0.35	89.3 ±0.36	**90.6** ±0.35
English	99.7 ±0.02	99.9 ±0.01	99.9 ±0.01	97.5 ±0.12	95.4 ±0.23	**97.7** ±0.17
Estonian	99.1 ±0.02	99.2 ±0.02	**99.3** ±0.02	90.3 ±0.22	80.4 ±0.29	**90.8** ±0.21
Hungarian	98.7 ±0.03	98.8 ±0.04	**98.9** ±0.03	92.1 ±0.25	79.9 ±0.34	**92.3** ±0.24
Romanian	99.7 ±0.00	99.9 ±0.00	99.9 ±0.00	88.3 ±0.10	83.0 ±0.14	**88.6** ±0.11
Multext						
English	99.8 ±0.01	99.9 ±0.01	99.9 ±0.01	97.8 ±0.16	96.0 ±0.19	**98.0** ±0.13
French	99.6 ±0.01	99.8 ±0.01	99.8 ±0.01	96.8 ±0.09	94.8 ±0.13	**97.1** ±0.08
German	99.7 ±0.01	99.7 ±0.01	99.8 ±0.01	96.2 ±0.17	95.4 ±0.20	**96.3** ±0.16
Italian	99.3 ±0.02	99.6 ±0.01	99.6 ±0.01	95.4 ±0.14	88.3 ±0.23	**95.7** ±0.14
Spanish	99.8 ±0.00	99.9 ±0.00	99.9 ±0.00	98.1 ±0.05	97.5 ±0.05	**98.4** ±0.04

Table 5: Comparison of accuracies achieved by RDR, CST and LemmaGen in the second experimental setting (using the MSD information).

The methodology of testing for second experimental setting is the same as in case of first experimental setting, therefore all the results from [Table 5] are directly comparable to [Table 4]. We notice an overall increase of accuracies in this setting as expected, since MSD tags can be used by algorithms to disambiguate between ambiguous wordforms.

6.5.2 Algorithms Ranking in Terms of Accuracy

Methodology of ranking is again the same as in the first experimental setting and the conclusions are also similar. In terms of accuracy, LemmaGen is, on training sets, significantly better then RDR, but not significantly different from CST. However, on the test sets, LemmaGen is significantly better than CST and RDR, as shown in [Figure 8].

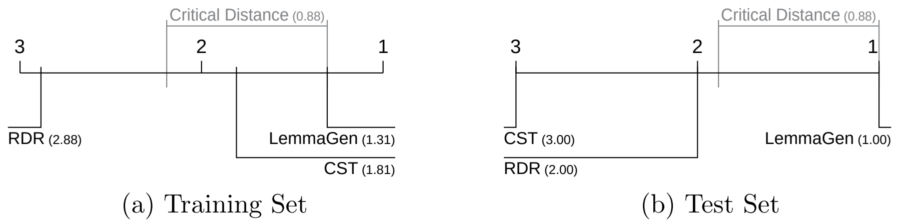


Figure 8: Visualisation of Bonferroni-Dunn post-hoc tests for the second experiment (with MSDs), using average ranks of algorithms on data from [Table 5].

6.6 Accuracy Comparison by Language

In this section we briefly compare and discuss the accuracy results achieved by LemmaGen on the test sets according to language, with the results summarised in [Table 6].

The accuracy rank of a language corresponds much better to the traditional notion of inflection complexity of languages and language families than do most statistics over the lexicons, with weakly inflecting West-European languages coming first, followed by heavily inflecting Slavic languages. The two non-Indo-European (and non-inflecting, but rather agglutinating) languages, Hungarian and Estonian are grouped together and appear among the Slavic languages.

There are, however, some outliers. Spanish should appear close to Italian, and the fact that it is first seems especially surprising as only about 3% of the wordforms in the lexicon are identical to their lemmas (c.f. [Table 3]), meaning that some morphological operation needs to be performed on practically every wordform. The explanation seems to be, as discussed in Section 6.1 that Spanish contains full paradigms for a large number of verbs, but while these paradigms contain many wordforms, the relation of the wordforms to the lemmas is quite regular and simple.

The first Slavic language is, unsurprisingly, Bulgarian, as it has lost almost all its nominal inflection, and is by far the least inflecting among the Slavic languages covered. It is interesting to note that Bulgarian, while being ranked seventh in the second experiment, was only the eleventh in the first - meaning that the lemma of its wordforms is very difficult to predict without recourse to the MSD. Next comes Slovene, which is also, to a certain extent, an outlier, as it would be expected to be quite close to Czech and Serbian. This is, as mentioned in Section 6.1, largely due to the different principles in constructing the lexica -

Slovene contains full inflectional paradigms, which Czech and Serbian have only the wordforms attested in the reference corpus used to construct the lexicon. They thus cover fewer wordforms per lemma, making for a sparser – and more irregular – dataset.

Next come the two agglutinating languages, Hungarian and Estonian, which have significantly different properties than inflectional ones - but, it would seem from the results, are still simpler than Slavic ones when the construction principles of the lexicons (taking only corpus wordforms and not full paradigms) are the same. After Czech comes, somewhat surprisingly, Romanian, which should intuitively be simpler than Slavic languages, although more complex than other Romance ones. An inspection of the lexicon reveals that at least a part of the reason is in their treatment of abbreviations and bound wordforms. The first are often expanded (e.g. "ADN" has the lemma "acid_dezoxiribonucleic"), while the second exhibit changes also in the first part of the word, rather than only in the suffix (e.g. "-mbarcasem" has the lemma "mbarca" and "amorul-propriu" the lemma "amor-propriu"). Such cases cannot be adequately covered by a lemmatiser that operates on word suffixes.

The worst ranked is Serbian – but this is likely due to the much smaller lexicon (training set) than is available for the other languages.

Language	MSDs	Experiment 1		Experiment 2	
		Accuracy	Rank	Accuracy	Rank
Spanish	264	95.4	1	98.4	1
English MTE	133	92.4	2	98.0	2
English MT	135	90.8	3	97.7	3
French	380	87.5	4	97.1	4
German	227	82.7	5	96.3	5
Italian	247	80.5	6	95.7	6
Bulgarian	338	70.4	11	94.1	7
Slovene	2,083	79.8	7	93.4	8
Hungarian	619	72.3	10	92.3	9
Estonian	643	63.3	13	90.8	10
Czech	1,428	78.3	8	90.6	11
Romanian	616	73.1	9	88.6	12
Serbian	906	65.3	12	86.1	13

Table 6: Comparison of LemmaGen accuracy by language. The first column gives the language, the second the number of different MSDs in the lexicon, the third and fourth columns the accuracy and rank for the first experiment with LemmaGen, and the fifth and sixth the accuracy and rank for the second experiment. The rows are sorted according to the rank of the second experiment (when MSDs were used).

6.7 Efficiency Testing

[Table 7] shows the results of efficiency testing. Numbers in the table represent the average time (in µs) needed to learn (Training time) and lemmatise (Lemmatisation time) one wordform. Hence, the table values should be read as: The

	Training time per word [µs]			Lemmatisation time per word [µs]		
Language	RDR	CST	LemmaGen	RDR	CST	LemmaGen
Multext-East						
Slovene	14.3	87.8	**3.9**	3.2	4.6	**0.6**
Serbian	9.7	58.6	**5.8**	2.5	5.2	**0.7**
Bulgarian	22.2	72.5	**4.6**	5.5	5.3	**0.7**
Czech	10.8	63.1	**3.4**	2.3	4.2	**0.6**
English	8.2	57.4	**3.2**	2.0	4.8	**0.6**
Estonian	14.3	118.7	**4.8**	3.0	6.0	**0.7**
Hungarian	10.5	79.2	**4.5**	2.2	6.0	**0.7**
Romanian	120.7	210.2	**6.7**	46.2	7.2	**0.7**
Multext						
English	7.6	52.1	**3.1**	1.8	4.7	**0.6**
French	16.6	92.5	**4.6**	3.7	6.2	**0.7**
German	12.0	74.9	**3.1**	2.7	4.2	**0.7**
Italian	9.2	86.9	**4.6**	2.0	6.1	**0.7**
Spanish	17.4	92.4	**4.7**	4.0	7.6	**0.7**

Table 7: Efficiency Testing: Comparison of the learning and lemmatisation efficiency between RDR, CST and LemmaGen.

lower the better. We do not provide ranking tests for efficiency results, since it is obvious that LemmaGen is by far more efficient than the other two algorithms.

7 Implementation and Availability

The LemmaGen learning algorithm (implemented in C++) and the 13 induced lemmatisers (lemmatisation rules in the refined RDR form, together with the reasoner which assigns a lemmatised form to a new word to be lemmatised) are freely available under the GNU open source license and downloadable from http://kt.ijs.si/software/LemmaGen/v2/. The original implementation is in the form of a set of command line utilities. However, we also provide much nicer web user interface (http://lemmagen.ijs.si/), were users can directly lemmatise individual texts or entire files in any described language without the need to download, install or compile the software locally. Our lemmatisers have already been used in practice: lemmatisers for 12 different languages have been incorporated into the Orange data mining toolkit [Orange 2009].

The Multext-East lexicons are also freely available from our website at http://nl.ijs.si/ME/, and the Multext lexicons are available from ELRA (http://www.elra.info).

8 Summary and Further Work

We have developed LemmaGen, a learning algorithm for automatic generation of lemmatisation rules in the form of a refined RDR tree structure. The algorithm has O(N) learning time complexity, is very efficient in lemmatisation, and can produce accurate lemmatisers from sufficiently large lexicons. The main contributions of this paper are the following:

– Improved compactness and readability of rule representation, by refining the RDR structure, resulting in condensed rules according to the similarity between conditions (suffixes) inside one exception list [see Section 3].

– Design of a generic, non-incremental RDR learning algorithm with low worst-case time complexity [see Section 4].

– Efficiency of the LemmaGen lemmatisation algorithm, empirically validated against RDR and CST, which uses the refined RDR lemmatisation rules to assign lemmas to words of a given language. Improved efficiency is achieved by exploiting the design and compactness of the refined RDR structure [see Section 5].

– Lemmatisation rules, induced from Multext-East and Multext training lexicons, which improve the accuracy and efficiency of lemmatisation for most of the 12 European languages covered [see Section 6].

– Finally, the availability of the LemmaGen software, the language models, as well as the public data from which the results were achieved, enabling the reproducibility of the reported experimental results, and direct use of the developed software and models for further research and applications [see Section 7].

This work leaves ample room for further research. It would be worthwhile to perform linguistic analysis of the LemmaGen rules, especially those obtained in the second experimental setting in which the data was separated into subsets according to different MSDs. Next, it is our plan to incorporate LemmaGen into other text analytics toolboxes, in the first instance into Ontogen [Fortuna et al. 2006].

Testing LemmaGen on other languages is also on our future agenda; the ultimate test will be the application of LemmaGen to lexicons of radically different languages, such as Arabic, which will most likely show the limitations of wordform suffix analysis approach used in our work.

In further work it would be also interesting to test whether the developed non-incremental RDR learning algorithm can be adapted to learning from other examples with a string-like structure, such as aminoacids in the case of proteins, and nucleotides in the case of genes.

Acknowledgements

This work was supported by the Slovenian Research Agency grant Knowledge Technologies and by the EU project FP7-211898 BISON. We thank Joël Plisson, Dunja Mladenić and Sašo Džeroski for previous collaboration in the development of lemmatisers for Slovene.

References

[Beth 1968] Beth, L.J. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistic* 11, pages 22–31, 1968.

[Brants 2000] Brants, T. TnT - A Statistical Part-of-Speech Tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000*, pages 224–231, Seattle, WA, 2000. http://www.coli.uni-sb.de/~thorsten/tnt/.

[Chrupala 2006] Chrupala, G. Simple data-driven context-sensitive lemmatisation. In *Proceedings of SEPLN-2006*, Zaragoza, Spain, 2006.

[Clark 2002] Clark, A. Memory-based learning of morphology with stochastic transducers. In *Proceedings of the Annual Meeting of the Association of Computational Linguistics*, pages 513–520, 2002.

[Clark and Niblett 1989] Clark, P. and Niblett, T. The CN2 induction algorithm. *Machine Learning*, pages 261–283, 1989.

[Compton and Jansen 1988] Compton, P. and Jansen, R. Knowledge in context: A strategy for expert system maintenance. In *Proceedings of the 2nd Australian Joint Artificial Intelligence Conference*, pages 292–306, 1988.

[Compton and Jansen 1990] Compton, P. J. and Jansen, R. A philosophical basis for knowledge acquisition. *Knowledge Acquisition*, 2:241–257, 1990.

[Dalianis and Jongejan 2006] Dalianis, H. and Jongejan, B. Hand-crafted versus machine-learned inflectional rules: The Euroling-SiteSeeker stemmer and CST's lemmatiser. In *Proceedings of LREC-2006*, pages 663–666, Genova 2006.

[Demšar 2006] Demšar, J. Statistical comparison of classifiers over multiple data sets. *Journal of Machine Learning Research* 7, pages 1–30, 2006.

[Dunn 1961] Dunn, O. J. Multiple comparisons among means. *Journal of the American Statistical Association*, 56:52–64, 1961.

[Džeroski and Erjavec 2000] Džeroski, S. and Erjavec, T. Learning to Lemmatise Slovene Words. In James Cussens and Sašo Džeroski, editors, *Learning Language in Logic*, number 1925 in Lecture notes in artificial intelligence, pages 69–88. Springer-Verlag, Berlin, 2000.

[Erjavec 2004] Erjavec, T. MULTEXT-East Version 3: Multilingual Morphosyntactic Specifications, Lexicons and Corpora. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC'04*, pages 1535–1538, Paris, 2004. http://nl.ijs.si/ME/V3/.

[Erjavec 2006] Erjavec, T. The English-Slovene ACQUIS corpus. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation, LREC'06*, pages 2138–2141, Paris, 2006. http://nl.ijs.si/svez/.

[Erjavec and Džeroski 2004] Erjavec, T. and Džeroski, S. Machine Learning of Language Structure: Lemmatising Unknown Slovene Words. *Applied Artificial Intelligence*, 18(1):17–41, 2004.

[Erjavec and Sárossy 2006] Erjavec, T. and Sárossy, B. Morphosyntactic tagging of Slovene legal language. *Informatica*, 30(4):483–488, 2006.

[Fisher 1959] Fisher, R. A. *Statistical methods and scientific inference (2nd ed).* Hafner Publishing Co., New York, 1959.

[Fortuna et al. 2006] Fortuna, B., Mladenić, D., and Grobelnik, M. Semi-automatic data-driven ontology construction system. *Proceedings of the 9th international multi-conference Information Society IS-2006* , A:212-220, 2006.

[Friedman 1940] Friedman, M. A comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics*, 11:86–92, 1940.

[Gaines 1991] Gaines, B. Induction and visualisation of rules with exceptions. In *6th AAAI Knowledge Acquisition for Knowledge Based Systems Workshop*, pages 7.1–7.17, 1991.

[Ide and Véronis 1994] Ide, N. and Véronis, J. MULTEXT: Multilingual Text Tools and Corpora. In *Proceedings of the 15th Conference on Computational Linguistics*, pages 588–592, 1994.

[Juršič 2007] Juršič, M. *Efficient Implementation of a system for Construction, Application and Evaluation of RDR Type Lemmatisers.* Faculty of Computer and Information Science, University of Ljubljana, Best Computer Science Thesis University Award, 2007.

[Kiparsky 1973] Kiparsky, P. "Elsewhere" in phonology. In Steven R. Anderson, editor, *Festschrift for Morris Halle*, pages 93–106. Holt, Rinehart and Winston, New York, 1973.

[Ling 1994] Ling, C. X.. Learning the past tense of English verbs: The symbolic pattern associator vs. connectionist models. *Journal of Artificial Intelligence Research*, 1:209–229, 1994.

[Manandhar et al. 1998] Manandhar, S., Džeroski, S. and Erjavec, T. Learning Multilingual Morphology with CLOG. In David Page, editor, *Inductive Logic Programming; 8th International Workshop ILP-98, Proceedings*, number 1446 in Lecture Notes in Artificial Intelligence, pages 135–144, Berlin, 1998.

[Mladenić 1993] Mladenić, D. Combinatorial optimization in inductive concept learning. In *Proceedings of the International Conference on Machine Learning, ICML*, pages 205–211, 1993.

[Mladenić 2002a] Mladenić, D. Learning word normalization using word suffix and context from unlabeled data. In *Proceedings of the International Conference on Machine Learning, ICML*, pages 427–434, 2002.

[Mladenić 2002b] Mladenić, D. Automatic word lemmatisation. In *Proceedings of the 5th International Multi-Conference Information Society, IS-2002 B*, pages 153–159, 2002.

[Mooney 1996] Mooney, R. J. Inductive logic programming for natural language processing. In *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pages 3–22, Berlin, 1997.

[Mooney and Califf 1995] Mooney, R. J. and Califf, M. E.. Induction of First-Order Decision Lists: Results on Learning the Past Tense of English Verbs. *Journal of Artificial Intelligence Research*, 3(1):1–24, 1995.

[Nakov 2003] Nakov, P. BulStem: Design and Evaluation of Inflectional Stemmer for Bulgarian. In *Proceeding of the Workshop on Balkan Language Resources and Tools*, Thessaloniki, 2003.

[Nemenyi 1963] Nemenyi, P. B. Distribution-free multiple comparisons. Ph.D. thesis, Princeton University, 1963.

[Orange 2009] Orange data mining software, http://www.ailab.si/orange.

[Plisson et al. 2008] Plisson, J., Lavrač, N., Mladenić, D. and Erjavec, T. Ripple Down Rule Learning for Automated Word Lemmatisation. *AI Communications* 21(1):15–26, 2008.

[Popovič and Willett 1990] Popovič M. and Willett. Processing of documents and queries in a Slovene language free text retrieval system. Literary and Linguistic Computing, 5, pages 182-190, 1990.

[Porter 1990] Porter, M.. An algorithm for suffix stripping. *Program* 14(3), pages 130-137, 1980.

[Rivest 1987] Rivest, R. L. Learning decision lists. *Machine Learning*, pages 229–246, 1987.

[Srinivasan et al. 1991] Srinivasan, A. Compton, P. J., Malor, R., Edwards, G., Sammut, C. and Lazarus, L. Knowledge acquisition in context for a complex domain. In *Proceedings of the European Knowledge Acquisition Workshop, Aberdeen*, 1991.

[Stroppa and Francois 2005] Stroppa, N. and Francois, Y. An analogical learner for morphological analysis. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 120–127. Association for Computational Linguistics, 2005.

[van den Bosch and Daelemans 1999] van den Bosch, A. and Daelemans, W. Memory-based morphological analysis. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 285–292, Morristown, New Jersey, 1999.

[Wilcoxon 1945] Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics*, 1:80-83, 1945.

[Yarowsky et al. 2001] Yarowsky, D. Ngai, G. and Wicentowski, R. Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of the first international conference on Human language technology research HLt-2001*, pages 1–8, Morristown, NJ, USA, 2001.