IOP Conference Series: Materials Science and Engineering

PAPER • **OPEN ACCESS**

Method of lemmatizer selections in multiplexing lemmatization

To cite this article: O A Sychev and N A Penskoy 2019 *IOP Conf. Ser.: Mater. Sci. Eng.* **483** 012091

View the article online for updates and enhancements.

This content was downloaded from IP address 163.125.19.252 on 06/11/2019 at 08:03

IOP Conf. Series: Materials Science and Engineering **483** (2019) 012091 doi:10.1088/1757-899X/483/1/012091

Method of lemmatizer selections in multiplexing lemmatization

O A Sychev, N A Penskoy

Volgograd State Technical University, 28 Lenina Avenue, Volgograd, 400005, Russia

E-mail: oasychev@gmail.com

Abstract. The article describes the method of selecting a particular lemmatizer while multiplexing lemmatization using word's part of speech and suffix. The article also describes the experiment to determine the best suffix (or prefix) length for multiplexing lemmatization. Lemmatizers, that are used to build the multiplexing lemmatizer, show worse results, than the resulting lemmatizer. The developed lemmatizer is compared with the best theoretically possible multiplexing lemmatizer The practical problems of creating multiplexing lemmatizer are discussed.

1. Introduction

Getting a word's lemma is necessary for solving various problems of natural language processing such as extracting knowledge from the natural language texts, texts classification and clustering, information retrieval, text abstracting and annotating and natural language translation, especially from a synthetic language to an analytic language [1]. Lemmatization is a problem of creating a binary relation between word forms and their lemmas, that represent the basic form of words. The problem is made worse because of word homonymy – one word can serve for different word forms of different lemmas (for example a word "lay" can be both past and past participle form of the verb "lie" and nominal form of the verb "lay").

The problem of mapping words to their lemmas in English can be reduced to transforming word's prefix and suffix [2].

There were developed several methods of lemmatization and part-of-speech determining.

The first method uses a dictionary of all word forms where each word form is mapped to its lemma and, possibly, part of speech. That method requires a lot of resources to create and verify the dictionary and big amount of memory to store it. The Prague Dependency Treebank is an example of that method [3]. That method also can be used approximately, while calculating editing distance, such as Levenshtein distance [4] or Damerau-Levenshtein distance. However, such approach could yield wrong results for languages with short words.

The second method is the rule-based approach. It uses a set of grammar rules to form lemmas. Some rules are applicable to certain words (like irregular verbs), others are applicable to thousands of words that are inflected using the same grammar rules [5].

The third method of lemmatization, which is another form of rule-based approach, is based on generating a graph in which nodes are states and edges change word during transitions from one state to others [6, 7].

Different researchers used different methods to compare the lemmatizer efficiency: some use information retrieval on a set of sentences [1, 8], others use a database of words with defined word forms and lemmas [9, 10]. However, the problem of using a particular dictionary for testing lemmatizers is that even if lemmatizer, which uses dictionaries (the first method), shows the best efficiency, that doesn't guarantee it works well with other words, that don't belong to dictionary.

The results of experiments in [10] show that lemmatization efficiency can be increased by multiplexing [11] several lemmatizers, while using each one for those words it performs with best. Multiplexing implies selecting different lemmatizers for different words by some algorithm.

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

2. Methods

The method of multiplexing lemmatization requires using several lemmatizers and choosing one of them by some criteria to lemmatize a particular word. In [10] it was shown, that generally PHPMorphy, which uses a dictionary-based approach, is the best lemmatizer available. However, it has its flaws and deficiencies and can be improved by multiplexing it with other lemmatizers. To determine the best lemmatizers for multiplexing the authors compared potential gains in the best possible scenario.

Table 1 shows a ratio of words that were correctly lemmatized by different tools while being incorrectly lemmatized by PHPMorphy to all words that were incorrectly lemmatized by PHPMorphy. That is the maximal gain in efficiency that can be obtained by multiplexing PHPMorphy with that tool. English Penn Treebank tagset is used to designate parts of speech.

The ratio is measured separately for each part of speech. If we get the same lemma while lemmatizing two forms of each word, the word is considered correctly lemmatized. Tests for noun lemmatization include infinitive form (NN) and plural form (NNS); tests for adverbs include infinitive (RB), comparative (RBR) and superlative forms (RBS); tests for adjectives include infinitive (JJ), comparative (JJR) and superlative forms (JJS). Verbs are used in their base form (VB), past tense form (VBD), present participle form (VBG), past participle form (VBN), non-3rd person singular present form (VBP), 3rd person singular present form (VBZ).

In table 1 it can be seen, that generally, word_operators lemmatizer showed the best results for multiplexing with PHPMorphy. Word_operatrors lemmatizer was developed using the rule-based approach [10], which efficiently complements PHPMorphy in areas where it performs worse. The best case is the lemmatization of adjectives: word_operators lemmatizer correctly lemmatized 100% of words. However, it can't absolutely replace PHPMorphy. Analysis of noun lemmatization shows that word_operators lemmatizer is better for about 3 thousand words, but there can be seen no clear pattern to use for rule-based multiplexing PHPMorphy with word_operators.

Verb analysis (VBD/VBG, VBN/VBG, VBP/VBZ cases) shows a large group of verbs with suffix "is" that are poorly lemmatized by PHPMorphy. But the analysis of all verbs with that suffix doesn't show any significant difference between PHPMorphy and word_operators. So, the researchers formed a hypothesis that PHPMorphy problems in verb lemmatizing were mainly caused by its incomplete dictionary, not by a word pattern that can be identified and used for multiplexing it with other lemmatizers.

Table 1. The relative improvement of PHPMorphy efficiency by multiplexing it with other lemmatizers (ideal multiplexing).

type/tool	NLTK lemmatizer	SimpleNLG	ruby lemmatizer	skyeng	word operators
JJ/JJR	0.353211	0.117737	0.166667	0.472477	0.467890
JJ/JJS	0.248603	0.069832	0.167598	0.365922	0.332402
JJR/JJS	0.720000	0.272000	0.188000	0.788000	1.000000
NN/NNS	0.204370	0.001490	0.147256	0.207350	0.766327
VBD/VBG	0.531250	0.010417	0.156250	0.531250	0.744792
VBD/VBN	0.409091	0.181818	0.227273	0.409091	0.590909
VBD/VBZ	0.530000	0.010000	0.140000	0.550000	0.440000
VBN/VBG	0.553191	0.010638	0.228723	0.553191	0.750000
VBN/VBZ	0.545455	0.010101	0.116162	0.565657	0.424242
VBP/VBD	0.476923	0.000000	0.174359	0.528205	0.487179
VBP/VBG	0.520408	0.010204	0.204082	0.551020	0.673469
VBP/VBN	0.497382	0.000000	0.120419	0.549738	0.471204
VBP/VBZ	0.685393	0.033708	0.179775	0.685393	0.797753
VBZ/VBG	0.580952	0.023810	0.176190	0.609524	0.547619
Total	0.293603	0.029917	0.154679	0.320178	0.688256

Considering that most English words change their suffix during inflection, it is possible that grouping words by their suffix (of some fixed length) and determining the best lemmatizer for each group will improve the resulting lemmatization efficiency. Since attempts to find specific clusters of words, on which particular lemmatizers perform best, failed, a database with mapping each possible suffix to the relevant lemmatizer is necessary.

As an alternative hypothesis, using prefixes instead of suffixes must be considered.In English language the first letters of a word are often the root of the word, so using fixed length prefixes we can explore how word roots affect words' lemmatization by modern software.

To do that we have to group all words by their parts of speech and inside these groups we have to group words by their prefix or suffix. The best lemmatizer will be determined for each group. That data were stored in a file. We will evaluate the efficiency of a multiplexing lemmatizer using different lengths of prefixes and suffixes to find the best implementation.

A multiplexer lemmatizer should determine the part of speech for the word, find its prefix or suffix of necessary length and find the lemmatizer to use that data and get lemma.

The authors performed the experiment to find the optimal suffix (prefix) length for that algorithm. As shown in [13], in the English language the average word length is 5 characters, so suffixes and prefixes with a length of up to 5 characters were considered. The use of a large prefix (or suffix) brings multiplexing lemmatization close to creating a dictionary containing the best lemmatizer for each word. Too long suffix or prefix will make the multiplexing lemmatizer less efficient when encountering words that weren't used for its training. But the use of too short suffixes doesn't allow enough distinctions between differently inflected words.

The suffix size also affects the number of rules we must record in the file. Not every letter combination of that length will be used (because not all the letter combinations actually occur in words), but the number of possible entries will be growing exponentially. It will affect the lemmatizer performance.

There was used the English dictionary from the Language Tool project to perform the experiment. That dictionary contains about 350 thousand of entries that consist of a lemma, a word form, and a part-of-speech tag. The lemmatizer efficiency with a given suffix (prefix) length was measured in two ways. The first one was for measuring the efficiency of multiplexing lemmatizer that was built using all words in the dictionary for given suffix length. The second lemmatizer was built using 80% of words; the words were chosen randomly from the dictionary; the efficiency of this lemmatizer was measured too. The second measurement allows us to evaluate how efficiently the multiplexing lemmatizer will behave for words that were not used for its building; decrease of that efficiency would signal us that increasing suffix length more isn't effective.

Five lemmatizers were used to build the multiplexing lemmatizer: PHPMorphy, NLTKLemmatizer, SimpleNLG, rubylemmatizer, skyeng, and word_operators lemmatizer.

3. Results

Table 2 shows the number of rules in the multiplexing file, the lemmatization efficiency and the lemmatization efficiency when training lemmatizer on partial data set, depending on suffix length.

Table 2 shows that the optimal suffix length is 3. Suffix lengths' value over 3 letters considerably increases the multiplexing file while decreasing efficiency for words not present in the original dictionary (cf the third and the fourth columns). The efficiency for lemmatizers built using the full dictionary rises insignificantly.

After performing a similar experiment using prefixes instead of suffixes, it was found that the optimal prefix length is 2. Lemmatization efficiency for lemmatizers trained on 80% of dictionary drops faster with prefix length than with suffix length.

Table 3 shows the comparison of the best prefix-based and the best suffix-based multiplexing lemmatizers for different parts of speech. The suffix-based lemmatizer performs better than the prefix-based one; the difference is the most evident for comparative adjectives (JJR) and verbs in past tense form.

Table 2. The number of rules and lemmatization efficiency depending on suffix length.

Suffix length	Number of rules	Lemmatization efficiency	Lemmatization efficiency for the training of lemmatizer using 80% of the dictionary
1	42	0.97488	0.97484
2	569	0.98090	0.98060
3	4230	0.98507	0.98276
4	18765	0.98702	0.97622
5	49947	0.98905	0.95127

Table 3. The efficiency of the prefix-based and the suffix-based multiplexing lemmatizers for different parts of speech.

Part of speech	prefix_tool	suffix_tool
JJ	0.998663	0.998736
JJR	0.605851	0.856156
JJS	0.804255	0.876123
NN	0.999924	0.999975
NNS	0.941226	0.948157
VBD	0.653851	0.974248
VBG	0.966683	0.981184
VBN	0.967657	0.975084
VBP	0.999250	0.999417
VBZ	0.981024	0.981690
Total	0.969263	0.979621

4. Discussion

Table 4 allows us to compare the developed multiplexing lemmatizer (multiplexing_tool) with the two best existing lemmatizers (PHPMorphy and word_operators) and the best theoretically possible multiplexing lemmatizer (i.e. the lemmatizer that performs correct lemmatization for every word for which at least one of the included lemmatizers does it correctly). The multiplexing lemmatizer using 3-letter suffixes performs better than any existing lemmatizer; it performs slightly worse than the theoretically ideal case. It's especially evident for lemmatizing comparative adjectives, where multiplexing lemmatizers have a big advantage over two best individual lemmatizers.

So multiplexing lemmatization is a useful way to efficiently improve lemmatization. The experiment shows us that the best multiplexing lemmatizers could be built using word suffixes with a length of 3 letters and parts of speech to determine which lemmatizer to use.

Practical concerns of using multiplexing lemmatizers should address a problem of getting all multiplexed lemmatizers to work on a particular platform. These lemmatizers are implemented in different programming languages. So, it's better not to just note the most effective lemmatizer for each suffix in the file, but several most effective ones to have a fallback if a particular lemmatizer is unavailable at a particular platform or in particular time. That allows the lemmatizing system to be more reliable to cases when external web services are going down.

Table 4. Comparison of multiplexing lemmatizer efficiency with the best lemmatizers and the theoretically ideal multiplexing lemmatizer.

word_operator/POS	PHPMorphy	word_operators	multiplexing_tool	ideal_multiplexing

JJ	0.793151	0.998718	0.998736	0.999982
JJR	0.709061	0.779358	0.856156	0.905323
JJS	0.803783	0.791962	0.876123	0.892671
NN	0.887252	0.999962	0.999975	1.000000
NNS	0.924569	0.931075	0.948157	0.991306
VBD	0.961792	0.545215	0.974248	0.994610
VBG	0.966683	0.549976	0.981184	0.994487
VBN	0.962746	0.544562	0.975084	0.994729
VBP	0.987286	0.999300	0.999417	0.999767
VBZ	0.948899	0.931067	0.981690	0.998803
Total	0.887812	0.928967	0.979621	0.995122

References

[1] Ozturkmenoglu O and Alpkocak A. 2012 Comparison of different lemmatization approaches for information retrieval on turkish text collection *Innovations in Intelligent Systems and Applications* (Trabzon: IEEE) 1-5

[2] Blevins J P 2006 English Inflection and Derivation The handbook of English linguistics ed Aarts B, McMahon A M (Wiley-Blackwell) pp 507-36

[3] Jan H, Panevová J, Hajičová E, Sgall P, Pajas P, Štěpánek J, Havelka J, Mikulová M, and Žabokrtský Z. 2006 *The Prague Dependency Treebank 2.0* (Philadelphia: Linguistic Data Consortium)

[4] Lyras D P, Sgarbas K N and Fakotakis N D 2007 Using the levenshtein edit distance for automatic lemmatization: a case study for modern greek and english *Tools with Artificial Intelligence* (Patras: IEEE) **2** 428-35

[5] Joel P, Nada L and Dunja M 2004 A rule based approach to word lemmatization 7th International Multi-Conference Information Society IS (Ljubljana: IEEE)

[6] Nicolai G, Cherry C and Kondrak G 2015 Inflection generation as discriminative string transduction *Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: human language technologies* (Denver: ACL Antology) 922-31

[7] Bhattacharyya P, Bahuguna A, Talukdar L, Phukan B 2014 Facilitating multi-lingual sense annotation: human mediated lemmatizer *Proceedings of the Seventh Global Wordnet Conference* 224-31

[8] Kanis J and Skorkovská L 2010 Comparison of different lemmatization approaches through the means of information retrieval performance International Conference on Text, Speech and Dialogue 93-100

[9] Eger S, Gleim R and Mehler A 2016 *International Conference on Language Resources and Evaluation* (Portorož: European Language Resources Association)

[10] Sychev O, Gurtovoy V and Penskoy N 2017 A study of efficiency of modern inflection and lemmatization software *Proceedings of the IV International research conference "Information Technologies in Science, Management, Social sphere and Medicine"* 308-15

[11] Bruneton E, Coupaye T and Stefani J 2002 Recursive and dynamic software composition with sharing *Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming*

[12] Marcus M P, Marcinkiewicz M A and Santorini B 1993 Building a large annotated corpus of english: the penn treebank *Computational linguistics 19* **2** 313-30

[13] Sigurd B, Olofsson M, Van Weijer J 2004 Word length, sentence length and frequency–zipf revisited Studia Linguistica 581 37-52