Identification of lemmatization errors using neural models

Attila Novák, Borbála Novák

Pázmány Péter Catholic University Faculty of Information Technology and Bionics
MTA-PPKE Hungarian Language Technology Research Group
Budapest, Práter u. 50/a.
{novak.attila, novak.borbala}@itk.ppke.hu

Abstract. Most research related to text processing focus on implementing algorithms solving complex tasks, considering simple preprocessing tools as acceptable together with their shortcomings. However, the performance of these low-level tools is sometimes far from being perfect, and errors introduced by them in a text processing chain propagate to higher level modules. In this research, our goal was to create an algorithm that can be used to improve the accuracy of a neglected, but important low-level tool, the lemmatizer. In order to achieve this goal, we experimented with a recurrent neural network classifier and an SVM-based classifier using a word embedding representation of word forms. Our system is able to predict with high accuracy (92.13%) whether a lemma candidate assigned to a word form is correct with the given part-of-speech.

Keywords: lemmatization, morphology, error correction, machine learning, SVM, RNN

1 Introduction

Most automatic text processing chains consist of a cascade of independent modules, where each module is applied to the output of the preceding one. In the case of such architectures, errors introduced at the beginning of the process, propagate through the entire chain with high-level tools producing erroneous output due to the erroneous input received from earlier modules. Thus it is of crucial importance that tools performing standard preprocessing tasks work as accurately as possible.

In this paper, we propose a method aiming at improving the quality of such a tool: we present the implementation of a method that is able to judge whether a lemma candidate produced automatically by a morphological analyzer/guesser algorithm together with the corresponding part-of-speech tag generated by the tool is correct. Using this method, the number of erroneous lemmata introduced into the annotation during automatic analysis can be decreased significantly.

In our research, we experimented with the lemmatizer algorithm of the PurePos part-of-speech tagger [8]. PurePos provides the possibility of integrating a morphological analyzer (MA), and the morphological analyzer provides lemmata

Published by Springer, Cham in:
A. Gelbukh (Ed.): CICLing 2018, LNCS 13396, pp. 399–407, 2023.
https://doi.org/10.1007/978-3-031-23793-5_32

and morphosyntactic tags for words that are included in its lexicon. For those words unknown to the analyzer, however, PurePos applies a simple lemmatization algorithm: a slightly modified version of the suffix guesser algorithm of the TnT tagger [1] is used that returns the operation to be applied to the word ending to obtain the lemma from the original word form in addition to the assigned morphosyntactic tag [8]. The accuracy of this algorithm is relatively low (89.01% measured on standard texts, i.e. it generates one erroneous lemma for every 10 words unknown to the MA[8]).

On the other hand, erroneous lemmata may not only be generated for word forms unknown to the morphological analyzer. In some cases, the analyzer itself overapplies some productive pattern generating an analysis, that may not make sense, with the lemma not being correct either.

The main source of erroneously lemmatized words is noisy or low quality texts. If the input contains a high number of misspelled or non-standard word forms, the morphological analyzer will not be able to cope with them, and, in absence of a morphological analysis, the less accurate suffix based guesser is applied. In the case of misspelled or non-standard words, however, the guesser often has no chance of algorithmically deriving a proper analysis. To make this possible, the text should be corrected prior to analysis.

The aim of our research was to create an algorithm that is able to identify incorrect lemmata, regardless of whether the guesser or the morphological analyzer produced them.

2 Method

We divided lemma candidates produced by PurePos into three classes:

1. Not lemma (`notlem`): the generated lemma candidate is not a lemma, but an inflected word form, e.g. *nyelvészek[N]* 'linguists[Noun]'. The PoS tag is otherwise correct.
2. Bad analysis (`badana`): the generated lemma candidate is not correct considering the assigned part-of-speech, e.g. *vár[Adj]* 'castle/wait[ADJ]'. The lemma candidate itself may or may not be correct. In the latter case, it may contain some orthographic error, or it may be a truncated word form.
3. Correct lemma (`lem`): the generated lemma candidate is correct together with the assigned part-of-speech tag, e.g. *vár[V]* 'wait[Verb]'.

We investigated the performance of two classification algorithms at the task of automatically assigning each lemma candidate to one of these three categories: a character-based recurrent neural network (RNN), and a Support Vector Machine classifier (SVM). The feature set for the SVM was a representation of the lemma candidates that consisted of the embedding vector of the original word form obtained from a raw tokenized corpus, the embedding vector of the lemma and the part-of-speech tag obtained from the morphologically tagged version of the same corpus, and a one-hot representation of the proposed PoS tag itself. In the following subsections, we present each step of the implementation and training of our lemma candidate classification systems as well as evaluation of their performance.

2.1 The Training Set

To create the training set, we used a 1.2-billion-token Hungarian webcorpus [2]. We analyzed the whole corpus using the PurePos tagger augmented with the Hungarian Humor morphological analyzer [5,9]. Being a webcorpus, it contains much noisy, non-standard text (social media contents, comments, blogs, etc.), thus there are many erroneously analyzed and lemmatized words in the output of the tagger/lemmatizer. we collected all the word forms occurring at least 5 times in the corpus together with their analyses. The analyses occurring in the resulting list were either created by the morphological analyzer, or by the guesser algorithm, but no information about the origin of the analysis was present in the output. We applied the morphological analyzer to the word forms appearing in the list and generated two classes of items based on the output of the analyzer. We considered base forms included in the lexicon of the morphological analyzer together with their part of speech as correct lemmata (`lem`). Words that were recognized by the analyzer as correct but inflected word forms rather than base forms were assigned the category (`notlem`).

In addition, we needed to identify analyses that contained incorrect lemmata either due to the form of the lemma or the assigned part-of-speech being incorrect. This type of analysis errors turned out to be especially characteristic of texts containing a high number of word forms with nonstandard spelling, such as unaccented social media texts, erroneously tokenized texts, historical documents etc. In order to collect such badly analyzed words, we first collected lemmata that frequently appeared in their uninflected base forms in the corpus with some part-of-speech, but had an alternative analysis with a different part-of-speech with which they did not appear in the corpus as a base form. For example, the ambiguous lemma *vár*, 'castle[Noun]' or 'wait[Verb]' also occurred in the annotation generated by PurePos analyzed as an adjective in addition to the correct nominal and verbal analyses. However, only the forms *vár[N]* 'castle' and *vár[V]* 'wait' appeared as base forms in the output, the adjective analysis was produced by the lemmatizer guesser of PurePos as an erroneous analysis of another word. Starting with just a few dozen such words, we explored the nearest neighbors of these words in a word embedding model built from the analyzed corpus (see the `POS` model in Section 2.2), exploiting the earlier observation that words containing similar errors are placed near each other in the word embedding space [6]. Table 1 shows some examples for erroneously lemmatized words and their neighbors in the model. We applied agglomerative hierarchical clustering to the nearest neighbor lists [10] Finally, we created the list used as training data for the category `notlem` by manually checking these sets of neighbors of such incorrect words.

Finally, the training set consisted of 11066 erroneously analysed words, 82582 correct lemmata and 80000 inflected words, from which we separated 1000 randomly selected items from each category as the test set.

vár[Adj] ‘wait’	alma[V] ‘apple’	szép[V] ‘beautiful’
tud[Adj] ‘know’	funda[V] [Latin word fragment]	ilo[V] [word fragment]
kér[Adj] ‘ask for’	mangus[V] [Latin word fragment]	nari[V] [word fragment]
jé[Adj] ‘gee’	manda[V] [Latin word fragment]	evian[V] [word fragment]
fogad[Adj] ‘receive’	spiritualité[N] ‘spiritualité’	insa[V] [word fragment]
cso[Adj] [word fragment]	Tibi[V] [fragment of *Tibita*]	manam[V] [word fragment]

Table 1. Examples for incorrectly lemmatized and analysed words, and their nearest neighbours in the (analysed) word embedding model

2.2 Word embedding models

Neural word embedding models place words present in the corpus used for creating them in a few hundred dimensional semantic space, where words with similar meaning and/or syntactic, grammatical or stylistic features are placed near to each other, while less similar words are further apart [4,3]. In this research we used two models described in [7]. One of them was trained on a tokenized but otherwise raw corpus. Words are present in their original, possibly inflected surface forms in this model. In the other case, PoS tagging and morphological analysis was applied to the corpus, and the model was trained on a preprocessed version of the corpus where each original token was represented by one or two tokens. Uninflected words and punctuation are represented by a single token consisting of the lemma concatenated with the main PoS label of the token. In the case of inflected words, this is followed by another token, a complex tag consisting of the rest of the morphosyntactic features of the word (such as case, mood, tense, etc.). The detailed comparison of these models can be found in [7]. Both word embedding models were trained using the CBOW architecture with window size set to 5, and the resulting vectors were 300 dimensional.

2.3 A character-based recurrent neural network classifier

One of the most successful paradigms of recent years is applying neural networks and deep learning for solving different tasks of various fields, such as text processing. Thus, in our present research we also experimented with training a recurrent neural network using the training set for the classification task. The characters of the lemma candidate to be classified are fed to the input layer of the network one by one, followed by the PoS tag as a single token. The characters and the PoS tags are represented as one-hot vectors. The length of these vectors is equal to the sum of the number of different characters and PoS tags found in the training set. Each element of these vectors is 0, except for the element at the index corresponding to the actual character or PoS tag, where the value is set to 1.

The recurrent neural network (RNN) is a two-layer network, with one hidden state. In each step, it uses the vector of the actual input and the result of the preceding step (at the beginning it is set to 0). The output of each step is a 3-element-long vector with the value at each position corresponding to the weight ('probability') assigned by the model implemented in the network to the hypothesis that the input belongs to the given class. This partial result is used as the hidden state for creating the prediction at the next step. The final output is normalized by a LogSoftmax layer resulting in real probability distribution. The architecture of the RNN is shown in Figure 1. We used the PyTorch framework[1] for implementation with the following parameters: the dimension of the hidden layer was set to 256, the system was trained through 200000 epochs and the learning rate was 0.005.

Given an input in the form of `word[TAG]`, the RNN predicts the probabilities of the input belonging to each class. Even though in the training set an item may occur in more than one class, during evaluation we only considered top-ranked class selected by the network.

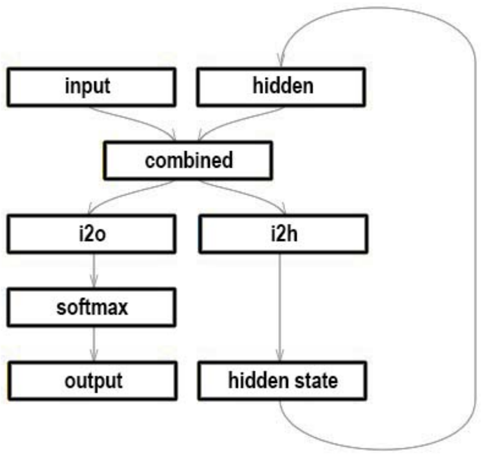

Fig. 1: Schematic structure of a recurrent neural network

2.4 Support Vector Machine classifier

In another experiment, we used a more traditional classifier using the word embedding vector representation of words as features. A Support Vector Machine (SVM) implements a supervised learning algorithm, where each item is a point in an n dimensional space and the task is to find those hyperplanes that separate the elements of the different classes best. When using the trained models, each test element is placed in the same space by the model, and this position determines the class assigned to it. The output in this case is a single class, rather than a predicted probability distribution of class membership.

[1] http://pytorch.org/

The input of the SVM is an n dimensional vector, constructed as the concatenation of three components:

- the 300 dimensional embedding vector assigned to the form of the lemma candidate (without its PoS tag) by the embedding model created from the raw corpus. If the word is not found in the model, a 300 dimensional vector of zeros is used. This component provides information about the the presence of the lemma candidate in the original corpus as an independent word form, and, if it did occur in the corpus, then its vector representation encodes distributional/grammatical characteristics of the word. In general, the lemma of a word is one of its existing forms. Which member of the inflectional paradigm of a word is used as a lemma depends on the part of speech of the word, and is determined by grammatical tradition. In general, a very frequent form is selected. This component is based on two assumptions: first, that the form used as the lemma of an inflected form of a not-very-infrequent lexical item should also appear in the raw corpus, and, second, that there is a systematic relationship between the part of speech of the lexical item and the vector representation of the lemma candidate form both in the case of real lemmata and in the case of other members of the paradigm (i.e. we can assume that the model can learn to distinguish base forms from other inflected forms for each part of speech, and to identify lemma candidates that were assigned the wrong part of speech).
- the 300 dimensional vector representation of the lemma candidate (with its PoS tag) retrieved from word embedding model created from the analyzed corpus. If the word is not found in the model, a 300 dimensional vector of zeros is used.
- the one-hot vector of the PoS tag, as used in the input of the RNN: a vector of length equal to the number of different PoS tags, containing all zeros, except for the index of the actual tag, where it is set to 1. There were 23 different part of speech tags in the corpus we used, thus the length of this component was 23.

Thus, the input representation of words for the SVM was a 623 dimensional vector. In order to ensure the possibility of non-linear separation based on the training data, we used the RBF kernel for training the model.

3 Results

For evaluation, 1000 items per class (not present in the training set) were used. Since one word could occur in more than one class, there were some duplicates in the list of randomly selected test words, but for the accuracy of the evaluation, we removed these words. Thus, the number of test words used for evaluating the models was 2974. Quantitative results are shown in Table 2.

We investigated the quality of the classifiers from two aspects. First, we measured the accuracy of the systems for the three-class classification. In this evaluation, only predictions exactly identifying the correct class were considered correct (see *Precision (3-class)*). The three-class classification accuracy of the RNN classifier was 74.37%, and that of the SVM classifier was 87.86%. In both cases, at least about 3/4 of the tested words were classified correctly, but the SVM almost reached 90% precision. Since our initial goal was to create an algorithm that is able to decide whether a lemma with a given part-of-speech is correct, in the other evaluation scenario we did not distinguish the classes `badana` and `notlem`: we did not consider it an error if the model confused these two classes. The `notlem` vs. `badana` distinction is only relevant because texts containing a high number of items from this class can be identified as orthographically substandard. In the latter evaluation scenario, i.e. predicting whether a lemma candidate is correct or not, the precision of the RNN was 80.53%, while that of the SVM was 92.13% (see *Precision (2-class)*). The SVM classifier using word embedding vector representations thus outperformed the character-based RNN in both evaluation scenarios. Even though the word ending plays an important role in determining the morphological class of a word, and the RNN model also had access to orthographic information concerning typical mistyping patterns in the `badana` class, it did not have access to any information about the context each lemma candidate usually appears in, while this information turned out to be both important for the task and effectively encoded in the embedding vectors and efficiently exploited by the SVM.

	Precision (3-class)	Precision (2-class)
RNN	74,37%	80,53%
SVM	87,86%	92,13%

Table 2. The precision of the two systems in the 3-class and 2-class scenario (`lem` vs. {`notlem|badana`})

In a more detailed evaluation, we also quantified the specific types of errors, as shown in Table 3. As it can be seen from the confusion matrices, the RNN assigned more words to the class `badana`, but almost one third of them (31.64%) should have been classified as either an inflected word or a correct lemma. The SVN, however, though having a lower recall, only assigned this class to 5 words erroneously. It can be concluded that the SVM has high precision, i.e. those words that were judged as invalid lemmata could be considered incorrect with high probability. Regarding confusion between the classes `notlem` and `lem`, the number of errors are also smaller for the SVM. 97.8% of those lemma candidates that were judged as not lemma were indeed not lemmata. The SVM model has very good recall for the `notlem` and `lem` classes. Unfortunately, the precision of the correct lemma class is smaller for the SVM, it is 83.3%. Comparing the two systems, the RNN performed better only for the recall of the class `badana`. Items belonging to this class were more often assigned to one of the other classes by the SVM. For the two-class (lemma/not lemma) classification, the SVM model has $R = 96.15\%, P = 83.30\%, F = 89.27\%$.

	RNN	result		
		`badana`	`notlem`	`lem`
gold	`badana`	713	93	156
	`notlem`	90	840	70
	`lem`	240	113	659

SVM	result		
	`badana`	`notlem`	`lem`
`badana`	646	127	189
`notlem`	0	994	6
`lem`	5	34	973

Table 3. The confusion matrices of the two systems measured on the test set

Looking at the results also revealed that quite often the origin of an incorrect lemmatization was a spelling error. Words that were originally misspelled, and, as a consequence, received an incorrect lemma, were assigned the class `badana` both in the training and the test set, even if the lemma and the PoS tag could have been correct for the correct form of the original word (i.e. the PoS tag is correct and the lemma contains exactly the same spelling error as the original word form). Our algorithm often classified these words as `lem`. This solution could be acceptable in the case of processing noisy texts.

4 Conclusions and future work

In this paper, we have presented and compared two algorithms trained to be able to judge whether an automatically generated lemma with a given part-of-speech is correct for a word or not. The implementation using a Support Vector Machine classifier using word embedding features performed better in the task, reaching an accuracy above 92%, while the recurrent neural network could achieve only slightly above 80% accuracy. Both systems were trained and tested for Hungarian, but no language-specific information was used. For training the models, we used a list of words generated automatically and checked manually and we used two types of word embedding representations for each word and their automatically assigned PoS tag. Though our system is not able to suggest a correct lemma for the words it classified as incorrect, if the classifier was integrated into the PoS tagger, the correct lemma could also be retrieved, since the current version of the integrated lemmatizer generates many lemma candidates for words unknown to the morphological analyzer. Instead of blindly selecting the top candidate from this list, the classifier could be used as a filter to remove erroneous candidates.

Acknowledgments

This research has been implemented with support provided by grants FK125217 and PD125216 of the National Research, Development and Innovation Office of Hungary financed under the FK17 and PD17 funding schemes.

References

1. Brants, T.: TnT: a statistical part-of-speech tagger. In: Proceedings of the Sixth Conference on Applied Natural Language Processing. pp. 224–231. ANLC '00, Association for Computational Linguistics, Stroudsburg, PA, USA (2000). https://doi.org/10.3115/974147.974178, `https://aclanthology.org/A00-1031/`
2. Endrédy, I., Prószéky, G.: A Pázmány Korpusz [The 'Pázmány' Corpus]. Nyelvtudományi Közlemények **112**, 191–206 (2016), `http://real.mtak.hu/79923/1/NyK20112_u.pdf`
3. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States. pp. 3111–3119 (2013), `https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf`
4. Mikolov, T., Yih, W., Zweig, G.: Linguistic regularities in continuous space word representations. In: Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA. pp. 746–751 (2013), `https://aclanthology.org/N13-1090`
5. Novák, A.: Milyen a jó Humor? [What is good Humor like?]. In: I. Magyar Számítógépes Nyelvészeti Konferencia [First Hungarian conference on computational linguistics]. pp. 138–144. SZTE, Szeged (2003), `http://www.morphologic.hu/downloads/publications/na/2003_mszny_Humor_na.pdf`
6. Novák, A.: Improving corpus annotation quality using word embedding models. Polibits **53**, 49–53 (2016), `http://www.scielo.org.mx/pdf/poli/n53/1870-9044-poli-53-00049.pdf`
7. Novák, A., Novák, B.: Magyar szóbeágyazási modellek kézi kiértékelése [Manual evaluation of Hungarian embedding models]. In: XIV. Magyar Számítógépes Nyelvészeti Konferencia. SZTE, Szeged (2018), `http://acta.bibl.u-szeged.hu/59034/1/msznykonf_014_067-077.pdf`
8. Orosz, Gy., Novák, A.: PurePos 2.0: a hybrid tool for morphological disambiguation. In: Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2013). pp. 539–545. Incoma Ltd. Shoumen, Bulgaria, Hissar, Bulgaria (2013), `https://aclanthology.org/R13-1071/`
9. Prószéky, G., Kis, B.: A unification-based approach to morpho-syntactic parsing of agglutinative and other (highly) inflectional languages. In: Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics. pp. 261–268. ACL '99, Association for Computational Linguistics, Stroudsburg, PA, USA (1999), `https://aclanthology.org/P99-1034/`
10. Siklósi, B.: Using embedding models for lexical categorization in morphologically rich languages. In: Gelbukh, A. (ed.) Computational Linguistics and Intelligent Text Processing. pp. 115–126. Springer International Publishing, Cham (2018), `http://users.itk.ppke.hu/~sikbo/pdfs/cicling-embedding-2016.pdf`