MoNoise: A Multi-lingual and Easy-to-use Lexical Normalization Tool

Rob van der Goot
Center for Language and Cognition
University of Groningen
r.van.der.goot@rug.nl

Abstract

In this paper, we introduce and demonstrate the online demo as well as the command line interface of a lexical normalization system (MoNoise) for a variety of languages. We further improve this model by using features from the original word for every normalization candidate. For comparison with future work, we propose the bundling of seven datasets in six languages to form a new benchmark, together with a novel evaluation metric which is particularly suitable for cross-dataset comparisons. MoNoise reaches a new state-of-art performance for six out of seven of these datasets. Furthermore, we allow the user to tune the 'aggressiveness' of the normalization, and show how the model can be made more efficient with only a small loss in performance. The online demo can be found on: http://www.robvandergoot.com/monoise and the corresponding code on: https://bitbucket.org/robvanderg/monoise/

1 Lexical Normalization

Because many natural language processing (NLP) systems are designed with standard texts in mind, they suffer performance drops when applied to texts from other domains. In recent years, social media has become a major source of information. Due to their hasty and spontaneous nature, texts on social media are particularly non-standard.

One solution to adapt NLP systems, is to 'translate' these non-standard texts to their standard equivalent. This task is also called *lexical normalization*, see Figure 1 for the normalization of "most social pple r troublesome". Previous work on normalization is fragmented; a variety of approaches is evaluated on a variety of benchmarks, using a variety of evaluation metrics and assumptions. Furthermore, most normalization systems are not opensource or publicly available.

most	social	pple	r	troublesome
most	social	people	are	troublesome

Figure 1: Example normalization of "most social pple r troublesome"

In this paper, we present MoNoise, an easy-to-use normalization system, consisting of an online demo as well a more elaborate command line interface. We include benchmarks, results and pre-trained models for a variety of languages.

2 Multi-lingual Normalization Benchmark

The manually annotated datasets on which we will evaluate MoNoise are summarized in Table 1. In all datasets, gold tokenization was assumed. When 1-N is indicated in the table, this means that splitting of words was included in the annotation, and in some rare cases also merging. Since capitalization is usually not corrected (merely kept) in almost all of these datasets, we will lowercase everything in our evaluation. For datasets with existing splits, those are used, in other cases the data is split 80%-10%-10% (train-dev-test). For LexNorm1.2 we use LiLiu as training and development data, in line with previous work (Li and Liu, 2014, 2015).[1].

3 MoNoise

In this section, we will give a summary of the normalization model MoNoise (van der Goot and van Noord, 2017). Additionally, we added one group of features which improves the performance of this model.

[1]This benchmark can be obtained by running `./scripts/0.getNormData.sh` from the repository

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 201–206
Florence, Italy, July 28 - August 2, 2019.

Corpus Source	Words	Lang.	%normed	1-N	Caps
GhentNorm De Clercq et al. (2014)	12,901	NL	4.8	+	+-
TweetNorm Alegria et al. (2013)	13,542	ES	6.3	+	+-
LexNorm1.2 Yang and Eisenstein (2013)	10,576	EN	11.6	-	-
LiLiu Li and Liu (2014)	40,560	EN	10.5	-	+-
LexNorm2015 Baldwin et al. (2015)	73,806	EN	9.1	+	-
IWT Eryiğit and Torunoğ-Selamet (2017)	38,918	TR	8.5	+	+
Janes-Norm Erjavec et al. (2017)	75,276	SL	15.0	-	+-
ReLDI-hr Ljubešić et al. (2017a)	89,052	HR	9.0	-	+-
ReLDI-sr Ljubešić et al. (2017b)	91,738	SR	8.0	-	+-

Table 1: Comparison of the normalization corpora used in this work. %normed indicates the percentage of words which is normalized. The '1-N' column indicates whether words are split/merged in the annotation, the 'caps' column indicates whether everything was lowercased (-), capitalization was transferred to the normalization (+-), or corrected (+).

3.1 The Architecture

MoNoise splits the normalization task in two subtasks; candidate generation and candidate ranking. In contrast to most other systems, no error detection is performed beforehand, the decision whether to normalize is made during ranking. Because the normalization task consists of a variety of replacement types (van der Goot et al., 2018), MoNoise is developed in a modular way. Some of the modules are based on raw, external data. This dependency allows the model to be transferred to new domains and timespans more easily. For both sub-tasks a variety of modules is designed, which are described in the following two paragraphs.

Important modules for candidate generation are Aspell[2], a dictionary learned from the training data and word embeddings (Mikolov et al., 2013) trained on non-standard data (where the 40 closests words using cosine distance are used). Because no normalization detection is done, the original word is also included as a candidate.

For the ranking of candidates, features from the generation are complemented with additional features. The additional features are: N-gram probabilities over non-standard text as well as standard texts, a feature which indicates whether a word contains alphanumeric characters or is a domain-specific token (hashtags, usernames and URLs) and the length of the original word and the candidate. All these features are combined in a random forest classifier, and the probability that a candidate belongs to the 'correct' class is used to rank the candidates.

3.2 Re-use Features of Original Word

A word should only be normalized when a substantially better candidate is found, this was not taken into account in the original model. To incorporate this intuition, we copy the features from the orginal word to all the other normalization candidates as additional features.

3.3 Models

For reproducability and reusability of the system, we provide pre-trained models for all the languages available in our multi-lingual benchmark (Section 2). These models are all trained using the default settings of MoNoise. These models exploit raw data from the source (non-standard) as well as the target (standard) domain, as n-gram probabilities and word embeddings are derived from these. As target domain data we use Wikipedia dumps from 01-01-2019[3]. For the non-standard data, we use raw data based on an in-house twitter collection[4]. In contrast to van der Goot and van Noord (2017), we do not not use language specific collections, but collect random tweets provided by the Twitter API during 2012 and 2018, and filter these by language based on the FastText language identifier (Joulin et al., 2016). Furthermore, we train embeddings with only 100 dimensions as opposed to van der Goot and van Noord (2017), who used 400. Because of these new embeddings, MoNoise uses 2-3 times less RAM, while experiencing only a very minor performance loss.

The pre-trained models can be found on: http://www.robvandergoot.com/data/monoise

[2] http://aspell.net/

[3] cleaned with https://github.com/attardi/wikiextractor

[4] https://developer.twitter.com/

Corpus	Lang	ERR	Precision	Recall	Prev. SOTA	Metric	Prev.	MoNoise
GhentNorm	NL	44.62	89.19	50.77	Schulz et al. (2016)	WER	3.2	1.36[5]
TweetNorm	ES	38.73	94.37	41.19	Porta and Sancho (2013)	OOV-Precision	63.4	70.40
LexNorm1.2	EN	59.21	80.87	77.56	Li and Liu (2015)	OOV Accuracy	87.58	87.63
LexNorm2015	EN	77.09	95.49	80.91	Jin (2015)	F1	84.21	86.58
IWT	TR	28.94	96.24	30.12	Eryiğit et al. (2017)	OOV Accuracy	67.37	48.99
Janes-Norm	SL	31.67	85.19	0.3833	Ljubešic et al. (2016) L1	CER	0.38	0.53
Janes-Norm	SL	63.90	95.66	0.6694	Ljubešic et al. (2016) L3	CER	1.58	2.24
ReLDI-hr	HR	51.65	95.66	0.541				
ReLDI-sr	SR	64.61	94.70	68.43				

Table 2: Results of MoNoise on the test data and a comparison with previous benchmarks. For WER and CER lower scores are better. Words normalized to the wrong candidate are classified as false positive (recall).

4 Evaluation

In this section, we first discuss existing evaluation metrics and their shortcomings and then introduce our novel metric. Secondly, we evaluate MoNoise on the test-splits of the multi-lingual benchmark, this is done twofold: using our preferred metric as well as a comparison to previous work.

4.1 Evaluation Metrics

Evaluation beyond word-level Some previous work used evaluation metrics which allow for evaluation beyond the word level. However, most of the normalization corpora do not include annotation beyond the word-level, except for the work of Zhang et al. (2013). Sometimes, BLEU score is used, whereas others use word error rate (WER) or character error rate (CER). We consider these metrics to be overly complex, since the word-order is not altered during annotation.

F1 In the shared task on normalization hosted at WNUT (Baldwin et al., 2015), F1 score was used as main evaluation. During development of MoNoise, we found multiple reasons why this might not be the preferable metric:

- Hard to interpret (how much of the problem is solved with an F1 score of 0.35?)
- It is unclear what to do with a word which should be normalized, but is normalized incorrectly, does this harm recall (false positive), precision (false negative) or both? [6]
- Because of the previous point, reproducibility and comparison with previous work can be difficult.

[5]Results are not directly comparable as different splits and tokenization is used

[6]For a more extensive discussion on this, we refer to van der Goot (2019). In the WNUT share task, they are FP and FN, in this paper they are considered only FP (recall).

Accuracy Early work on normalization often used accuracy over the words in need of normalization as main evaluation (Han and Baldwin, 2011; Liu et al., 2012) (OOV accuracy in Table 2). However, in this setting, the detection of which words need to be normalized is not taken into account. To include the full task of normalization, accuracy over all the words could be used. Accuracy is much easier interpretable compared to F1 score. However, accuracy does not allow for easy comparison across corpora, as different percentages of words might be in need of normalization. This is the main motivation for a novel evaluation metric, discussed in the next paragraph.

Error Reduction Rate Because previous metrics are overly complicated, hard to interpret or do not allow for an easy comparison between different datasets, we introduce the Error Reduction Rate (ERR). Error reduction rate can be interpreted as accuracy normalized for the number of words that are normalized in the gold standard. This allows for a direct comparison with a baseline which always copies the original word, the accuracy of such a baseline is equal to the percentage of words which need to be normalized. The formula for ERR is:

$$ERR = \frac{Accuracy_{system} - Accuracy_{baseline}}{1.0 - Accuracy_{baseline}} \quad (1)$$

The ERR will usually have a value between 0.0 and 1.0. A negative ERR indicates that the system normalizes makes more erroneous than correct normalizations. A baseline which always keeps the original word scores exactly 0.0, and a perfect system will score 1.0. We will use ERR as main evaluation metric and additionally report precision and recall, to gain more insights into the strengths and weaknesses of the system. A more

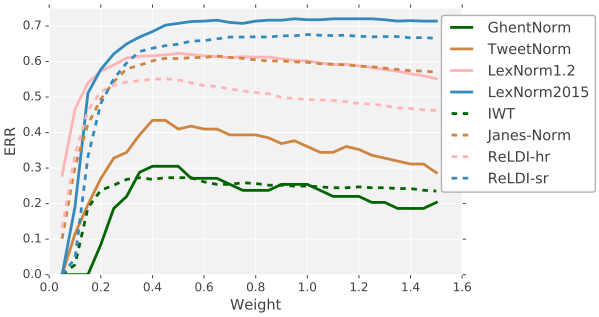

Figure 2: The effect of tuning the weight of the original candidate. A low weight indicates a more aggressive system.

detailed discussion on the motivation behind ERR can be found in (van der Goot, 2019).

4.2 Results

We present the results on all corpora of the multilingual benchmark in Table 2. We report the ERR, precision, recall, as well as the standard metric for each dataset. During development, no tuning was done on the test data, as described in more detail in (van der Goot and van Noord, 2017).

As mentioned in Section 4.1, it is disputable whether words which are normalized to the wrong normalization candidate should be counted as false positive or false negative. We choose to categorize these as false negatives, and thus be accounted for in the recall.

The ERR differs quite substantially across the datasets, this is due to different sizes of training data as well as differences in annotation. In the LexNorm2015 dataset, for example, phrasal abbreviations are expanded ('lol'↦'laughing out loud'), leading to a lot of very common replacements, which can easily be learned from the training data. On all datasets, the precision is higher than the recall. In other words, the model is conservative. This is arguably a desirable result, as it is important to avoid over-normalization.

When we look at the comparison to the previous state-of-the-art systems, we see that MoNoise performs highly competitive. Only on the Slovenian and Turkish datasets, the previous state-of-the-art is not surpassed.

5 Aggressiveness

To gain more control over the output normalization, we introduce a parameter which controls for the aggressiveness of the model. This is done by weighing the confidence score estimated for the original word (see also Section 3.1). If this is given a high weight, the original word is more likely to rank high.

The effect of tuning this parameter is plotted in Figure 2. It becomes apparent that the default weight of 1.0 leads to rather stable performance. For some datasets, minor gains can be achieved by a more aggressive setting. Furthermore, we can see that the parameter only becomes effective with extreme values, which is because the classifier is relatively certain about its predictions, and often gives very high scores to a certain candidate (>0.99). Although this parameter only leads to minor gains, it can still be useful to adapt the model to other domains, or to inspect whether the model was close to the correct normalization.

6 Efficiency

The main bottleneck for efficiency in the model is the searching of the 40 closest words in the word-embeddings. In the original word2vec (Mikolov et al., 2013) code, this is done by a for-loop which iterates through the whole vocabulary, and calculates the cosine distance for every word. There are more efficient techniques to calculate these distance, like the one used by Gensim (Řehůřek and Sojka, 2010). However, this still takes half a second per word on a modern pc for our English embeddings with a vocabulary size of 4,500,000. Therefore, we cache the 40 closest candidates in the embeddings for each word. We included the cached embeddings with each of our models. [7]

The next bottleneck of the model is the random forest classifier. So, further gains in efficiency can be gained by limiting the number of allowed candidates, as previously done by (Jin, 2015). This can be done by only considering words which occur in the corrected data, or a larger list by also including the Aspell dictionary.

The effect of filtering the candidates on the LiLiu dataset is shown in Table 3. Results on the other datasets showed similar trends. Filtering candidates based on only the training data results in a huge speedup, however also substantially harms performance. However, if we add the Aspell dictionary to the list of allowed candidates, performance remains relatively close, and a speedup of factor 2 is achieved.

[7]The code to cache embeddings (including a python wrapper) is available at: https://bitbucket.org/robvanderg/cacheembeds.

Restrictions	ERR	avg. cands	words/ sec	trainTime (seconds)
None	61.83	84	29	2,171
Train	51.64	12	137	280
Train + Aspell	61.12	43	62	1,104

Table 3: ERR when filtering candidates before ranking, and speed of the model when predicting and training. All reported results are the average of five runs on the LiLiu development set.

7 Interface

We provide two interfaces to use MoNoise; a command line application and a demo website.

7.1 Command Line

The only requirement to install MoNoise is a somewhat recent c++ compiler (c++11 or newer). In this section, we will highlight the most useful commands, for the full list of options we refer to the repository.

`--cands N` Outputs at most N candidates for each word and their probability. These probabilities are obtained by normalizing the confidence scores of the classifier so that they sum to 1.0.

`--caps` Do not lowercase everything, can be used during training as well as testing/running. This parameter is enabled for the online demo.

`--feats` Lets you provide a bytestring with which specific modules can be disabled, the modules are explained in van der Goot and van Noord (2017) and listed in `utils/feats.txt`.

`--known N` Only allow normalization candidates which occur in the normalized version of the training data(N=1), or allow candidates which occur in the training data or the Aspell dictionary(N=2).

`--tokenize` Employ a conservative tokenizer, which splits sequences of punctuation (∗.”?!(){} :;/, \˜&) from the beginning and end of a word. Here, we assume that the normalization model will take care of other irregularities.

`--weight N` Weigh the confidence of the original word with N, thereby tuning the aggressiveness (Section 5).

7.2 Online

In the online interface (Figure 3), the user can type a sentence and get the predicted normalization. The dropdown menu includes all languages for which a pre-trained model is available. The aggressiveness (Section 5) can be tuned with a slider which converts this agressiveness factor to a weight for the original word. This allows the user to inspect beyond the top-1 predicted normalization sequence. Additionally, some example social media posts are displayed for the user, which are not shown in Figure 3 due to privacy issues. The online demo can be used on: `http://www.robvandergoot.com/monoise`

Figure 3: The layout of the online demo on a low resolution screen.

8 Conclusion

In this paper, we have demonstrated the online interface and command line interface of MoNoise. For this system, we release models for 6 different languages, of which a new state-of-the-art is reached for multiple datasets. The system is easy to install and use on Unix-based systems and has many useful extra options. For even easier usage, the online demo can be used from any device with internet access and a browser. We discussed multiple practical issues, like evaluation, efficiency, and extra tuning parameters.

Acknowledgements

I would like to thank Kevin Humphreys for his help with integrating Aspell into MoNoise, Ian Matroos for providing the python wrapper for the cached embeddings, Gertjan van Noord for his suggestions during the development of MoNoise, and Hessel Haagsma for the suggesting the name ‘error reduction rate’. This system is developed in the ‘Parsing Algorithms for Uncertain Input’ project, funded by the Nuance foundation.

References

Inaki Alegria, Nora Aranberri, Víctor Fresno, Pablo Gamallo, Lluis Padró, Inaki San Vicente, Jordi Turmo, and Arkaitz Zubiaga. 2013. Introducción a la tarea compartida Tweet-Norm 2013: Normalización léxica de tuits en español. In *Tweet-Norm@ SEPLN*, pages 1–9.

Timothy Baldwin, Marie-Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. 2015. Shared tasks of the 2015 workshop on noisy user-generated text: Twitter lexical normalization and named entity recognition. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 126–135, Beijing, China.

Orphée De Clercq, Sarah Schulz, Bart Desmet, and Véronique Hoste. 2014. Towards shared datasets for normalization research. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA).

Tomaž Erjavec, Darja Fišer, Jaka Čibej, Špela Arhar Holdt, Nikola Ljubešić, and Katja Zupan. 2017. CMC training corpus janes-tag 2.0.

Gülşen Eryiğit and Dilara Torunoğ-Selamet. 2017. Social media text normalization for turkish. *Natural Language Engineering*, 23(6):835–875.

Rob van der Goot. 2019. *Normalization and Parsing Algorithms for Uncertain Input.* Ph.D. thesis, University of Groningen.

Rob van der Goot and Gertjan van Noord. 2017. MoNoise: Modeling noise using a modular normalization system. *Computational Linguistics in the Netherlands Journal*, 7:129–144.

Rob van der Goot, Rik van Noord, and Gertjan van Noord. 2018. A taxonomy for in-depth evaluation of normalization for user generated content. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Bo Han and Timothy Baldwin. 2011. Lexical normalisation of short text messages: Makn sens a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 368–378, Portland, Oregon, USA.

Ning Jin. 2015. NCSU-SAS-Ning: Candidate generation and feature engineering for supervised lexical normalization. In *Proceedings of the Workshop on Noisy User-generated Text*, pages 87–92, Beijing, China.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Chen Li and Yang Liu. 2014. Improving text normalization via unsupervised model and discriminative reranking. In *Proceedings of the ACL 2014 Student Research Workshop*, pages 86–93, Baltimore, Maryland, USA.

Chen Li and Yang Liu. 2015. Joint POS tagging and text normalization for informal text. In *Proceedings of IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1263–1269.

Fei Liu, Fuliang Weng, and Xiao Jiang. 2012. A broad-coverage normalization system for social media language. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1035–1044, Jeju Island, Korea.

Nikola Ljubešić, Tomaž Erjavec, Maja Miličević, and Tanja Samardžić. 2017a. Croatian Twitter training corpus ReLDI-NormTagNER-hr 2.0.

Nikola Ljubešić, Tomaž Erjavec, Maja Miličević, and Tanja Samardžić. 2017b. Serbian Twitter training corpus ReLDI-NormTagNER-sr 2.0.

Nikola Ljubešic, Katja Zupan, Darja Fišer, and Tomaz Erjavec. 2016. Normalising slovene data: historical texts vs. user-generated content. *Bochumer Linguistische Arbeitsberichte*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*.

Jordi Porta and José-Luis Sancho. 2013. Word normalization in Twitter using finite-state transducers. *Tweet-Norm@ SEPLN*, 1086:49–53.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA.

Sarah Schulz, Guy De Pauw, Orphée De Clercq, Bart Desmet, Véronique Hoste, Walter Daelemans, and Lieve Macken. 2016. Multimodular text normalization of Dutch user-generated content. *ACM Transactions on Intelligent Systems Technology*, 7(4):1–22.

Yi Yang and Jacob Eisenstein. 2013. A log-linear model for unsupervised text normalization. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 61–72, Seattle, Washington, USA.

Congle Zhang, Tyler Baldwin, Howard Ho, Benny Kimelfeld, and Yunyao Li. 2013. Adaptive parser-centric text normalization. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1159–1168, Sofia, Bulgaria.