Synthetic Data for English Lexical Normalization: How Close Can We Get to Manually Annotated Data?

Kelly Dekker, Rob van der Goot
University of Groningen, IT University of Copenhagen
k.dekker.5@student.rug.nl, robv@itu.dk

Abstract

Social media is a valuable data resource for various natural language processing (NLP) tasks. However, standard NLP tools were often designed with standard texts in mind, and their performance decreases heavily when applied to social media data. One solution to this problem is to adapt the input text to a more standard form, a task also referred to as normalization. Automatic approaches to normalization have shown that they can be used to improve performance on a variety of NLP tasks. However, all of these systems are supervised, thereby being heavily dependent on the availability of training data for the correct language and domain. In this work, we attempt to overcome this dependence by automatically generating training data for lexical normalization. Starting with raw tweets, we attempt two directions, to insert non-standardness (noise) and to automatically normalize in an unsupervised setting. Our best results are achieved by automatically inserting noise. We evaluate our approaches by using an existing lexical normalization system; our best scores are achieved by custom error generation system, which makes use of some manually created datasets. With this system, we score 94.29 accuracy on the test data, compared to 95.22 when it is trained on human-annotated data. Our best system which does not depend on any type of annotation is based on word embeddings and scores 92.04 accuracy. Finally, we perform an experiment in which we asked humans to predict whether a sentence was written by a human or generated by our best model. This experiment showed that in most cases it is hard for a human to detect automatically generated sentences.

Keywords: Social Media Processing, Statistical and Machine Learning Methods, Tools, Systems, Applications

1. Introduction

With the emergence of social media, a novel type of language evolved. This 'online' language contains many new phenomena, like slang, acronyms and shortened words. However, most natural language processing tools are designed to process standard texts. Due to these differences, performance of NLP tools often drop dramatically when used on social media data (Liu et al., 2011; Schulz et al., 2016)

Multiple solutions have been proposed to tackle this problem of domain adaptation. These solutions can be broadly divided in two categories: adapting the model to the new data, or adapting the data to the model (Eisenstein, 2013). In this work, we focus on lexical normalization, a solution which adapts the data to the model by 'translating' words to their standard form. This has the main advantage that the adaptation only has to be done once for multiple types of NLP applications. An example of a sentence annotated with its normalization is shown in Figure 1.

However, one downside of this solution is that existing normalization systems are supervised[1], and annotating such data is expensive and time consuming. Currently, annotated data is only available for a handful of languages. Furthermore, language develops continuously (especially on social media), and there are multiple sub-domains online which contain different types of language. For these reasons, we focus on the task of automatically generating training data for lexical normalization; the task is to generate pairs of sentences like shown in Figure 1. Naturally, there are two directions to generate these pairs, starting from the clean text and inserting noise, or the other way around. We will try a variety of methods for both of these strategies to normalize English tweets without training data. We opted to use English tweets to ensure that the results can be compared to previous work, and we can interpret the quality of the generated data.

wanting	to	make	a	yt	vid
wanting	to	make	a	Youtube	video

Figure 1: Example of an annotated tweet, top: original text, bottom: normalized text

This leads to the following research questions:

1. To what extent can we replace human annotated data with automatically generated training data for lexical normalization?
2. How does the performance of training on data generated with clean data as a basis compare to training data based on noisy data?
3. Is the automatically generated data of our best method similar to human generated data?

2. Related Work

To the best of our knowledge, we are the first to automatically generate and evaluate English training data for the lexical normalization task. For other languages, we are only familiar with work on Turkish (Çolakoğlu et al., 2019), in which data is automatically generated by inserting spelling variants and is then used to train a machine translation system. However, for the two approaches mentioned in the introduction (noisy↦clean and clean↦noisy) there is previous work for sub-tasks of lexical normalization for English, which we will discuss in the following paragraphs.

[1]There are publications about 'unsupervised normalization', however, this is usually focused only on the task of candidate selection and can thus not be used in a real-world scenario where it is unknown whether a word has to be normalized.

The noisy↦clean setting can be seen as a form of unsupervised normalization in itself. In our setup, we will focus mainly on precision instead of recall, since the quality of the data is important and data is abundantly available, see Section 4.2. for more details. Previous work on unsupervised normalization focused mainly on finding the correct replacement of words when it is already known that they need replacement. For this task, a variety of approaches are used; Contractor et al. (2010) modify a statistical machine translation system to work in an unsupervised setting by predicting word translations by a lexical similarity measure. Han and Baldwin (2011) use a combination of lexical and phonetic edit distances between noisy words and clean words to pick the correct replacement, whereas Xu et al. (2015) consider similarities on the syllable level. Other previous work constructs translation dictionaries in an unsupervised setting (Gouws et al., 2011; Han et al., 2012), these can be used to efficiently convert noisy text, however the recall will be limited. We will use existing methods for unsupervised grammar correction as well as translation dictionaries.

When transforming clean to noisy data, we are replacing canonical words with non-canonical words. To the best of our knowledge, this is not done before in the context of normalization, however, Foster and Andersen (2009) automatically generate grammatical errors on the character level (based on insertion, deletion, substition and move) and use this data to improve the detection of grammatical sentences. Kasewa et al. (2018) instead used neural machine translation to learn the distribution of naturally occurring grammatical errors and leveraged this knowledge to generate realistic grammar errors. We opt to use a method similar to Foster and Andersen (2009) for simplicity and efficiency. It should be noted that grammatical error correction is a sub-task of normalization, and the aforementioned approaches need to be supplemented. More recently, Doval et al. (2020) generated a confusion set for each word in a sentence by deleting one character at a time. The resulting set is then used to train word-embeddings, which showed to be more robust (especially for social media data) compared to training only on the original word.

For evaluation of our generated data, we need an automatic normalization system. We decided to use MoNoise (van der Goot and van Noord, 2017) because it achieved state-of-the-art results on various corpora (e.g. LexNorm1.2 and LexNorm2015) and the code is publicly available. MoNoise is based on the observation that the normalization problem consists of multiple sub-problems; for each of these sub-problems specific modules are designed for the generation and ranking of normalization candidates. The most important modules are a spell-checker, word embeddings, a translation dictionary and N-gram probabilities.

3. Data

3.1. Raw data

For this research, we used English data collected from the Twitter API during November 2016 based on the 100 most frequent words of the Oxford English Corpus[2]. We discarded truncated tweets, as we are mainly interested in processing complete sentences, we also discarded tweets containing URL's to avoid tweets just mentioning articles. Next, we removed duplicate tweets, as we require some variety in our training data. Finally, we use the NLTK TweetTokenizer, to make the tokenization similar to the evaluation datasets.

Then, we classify each tweet as being clean or noisy, resulting in two datasplits which we will refer to as respetively CLEAN and NOISY data. To perform this split we use the dictionary of the Enchant library[3], if all words of a tweet occur in this dictionary, it is classified as clean. We removed all single alphabetical characters from the vocabulary, except 'i' and 'a' and treat usernames and hashtags as in-vocabulary to improve the classification.

3.2. Evaluation data

To evaluate our generated data and the resulting normalization model, we need some manually annotated lexical normalization data. We will use the LexNorm1.2 data (Yang and Eisenstein, 2013) for final evaluation, as this does not include multi-word normalizations (which we leave for future work). This data consists of 549 tweets collected in 2012 from a random sample. For tuning the system we use the same development data as van der Goot (2019b), taken from (Li and Liu, 2014), for easy comparison and because it is annotated in a similar style. The development data consists of 518 tweets (10,576 words) collected during 2009 and 2010 and filtered based on the probability that it contains a certain amount of noisiness. In the development data, 11.5% of the words is changed after normalization annotation.

4. Method

We propose two directions to automatically generate training data for normalization: clean $\mapsto$ noisy and noisy $\mapsto$ clean. In the next two subsections, we will describe a variety of methods for both directions.

4.1. Clean $\mapsto$ Noisy

For the insertion of noise, we propose three approaches: Brown clusters, word embeddings and a custom error generation tool.

4.1.1. Brown clusters

Brown clusters (Brown et al., 1992) can be trained on raw data and provide an hierarchical ordering of words in a binary tree. Words which occur in similar contexts, end up in similar clusters. Here, we are building on the assumption that non-canonical words will end up in a cluster with their normalized equivalent. We use the publicly available clusters trained on Twitter data by Owoputi et al. (2012). These clusters are trained on more than 56 millions Tweets (over 847 million tokens) collected between 2008 and 2012.

To generate normalization pairs, random words are selected from a clean tweet and converted to a noisy variant based

[2] https://en.wikipedia.org/wiki/Most_common_words_in_English

[3] https://abiword.github.io/enchant/

on its cluster. To make it more probable that we select a lexical variant instead of a semantically related other word we established some constraints:

- The candidate word should start with the same letter as the original word OR the candidate word may start with a digit (to include noisy words such as *2morrow*)
- The candidate word should end with the same letter as the original word OR the second-to-last letter should be the same as the last letter of the original word OR end with a digit (to include noisy words such as *no1*)
- The candidate word should not contain punctuation marks (except ') or emojis[4]

After the above rules were applied, every randomly selected word gets a list of suggestions with noisy candidates. From this list, a random noisy word was chosen for the normalization pair.

4.1.2. Word Embeddings

Word embeddings are representations of words as vectors of continuous values, these are commonly derived from large amounts of raw texts. These embeddings spaces are known to carry semantic as well as syntactic signal. In this work, we will make use two different embeddings models:

- MoNoise embeddings: The embeddings used my MoNoise (van der Goot and van Noord, 2017), which are skip-gram embeddings trained on 760 million tweets with the default settings of word2vec (Mikolov et al., 2013), except for a vector size of 400.
- GloVe Twitter embeddings: GloVe embeddings with 50 dimensions trained on 2 billion tweets (Pennington et al., 2014).

For each of these models, we use a similar method as with the Brown clusters. We pick random words from a clean tweet and find the 40 closest words in the embeddings space using cosine distance. The same constraints as used for the Brown clusters are used to pre-filter the list of candidates, and a random candidate is selected from the filtered list.

4.1.3. Custom error generation (CEG)

Finally, we will experiment with a more linguistically-informed approach, a custom error generation method. To make sure we generate all types of replacements which are annotated for the normalization task, we will make use of an existing categorization. We choose to use the taxonomy by van der Goot et al. (2018), as it is the most fine-grained for this type of annotation. All multi-word replacement categories are ignored, because they are not annotated in the datasets we use, which leaves us with 10 categories. We assign every tweet in the CLEAN dataset with one category and attempt to inject errors only from that category. If this is impossible (for example, removing an apostrophe when there is none), the tweet is kept as-is. The 10 categories are described in more detail below:

[4]detected by the Python emoji library: `https://pypi.org/project/emoji/`

Typographical error
A typographical error includes word errors which can be the result of mistyping on a keyboard. To generate these errors, we take random words from the tweets, and substitute or insert a character (as deletion is already handled by other categories). We use the QWERTY lay-out to pick characters which are close to the original character, to make the errors more 'human-like'. Some examples:
nerved → nervwd amazing → anazing

Missing apostrophe
To generate the error of leaving out apostrophes in words, random words which contain an apostrophe are selected from the clean data set and the apostrophe has simply been removed. Some examples:
Won't → Wont Didn't → Didnt

Spelling error
A spelling error is similar to a typographical error. However, it is not the result of mistyping a key on the keyboard, but is an effect of a lack of grammatical/spelling knowledge. To insert these errors, a look up list from Wikipedia of common misspelled words (Wikipedia:Lists of common misspellings, 2017) has been used. Some examples:
tomorrow → tommorrow until → untill

Repetition
People emphasize words on social media by repeating characters. To artificially generate these repetitions, random words were selected and also a random number (1-4), was generated to specify the number of character repetitions. This repetition was based on the last character of a word, since other categories, like typographical error, spelling error and slang already covers repetition of characters mid-words.
Some examples:
thing → thinggg No → Nooo

Shortening vowels
On social media, vowels are sometimes left out to shorten a word. To make these kind of errors, random words were selected and 1 or more vowels were left out. Some examples:
with → wth better → btter

Shortening end & Shortening other
Both the categories shortening end and shortening other have been merged together in one module. This is done, because for these categories a look-up list has been created by scraping websites[5] containing both these shortening categories. Some examples:
minutes → mins birthday → bday

[5]`https://www.englishclub.com/ref/Slang/Shortened_Words/`
`https://englishharmony.com/33-word-shortenings/`
`https://www.npr.org/sections/alltechconsidered/2014/01/16/263096375/researchers-are-totes-studying-how-ppl-shorten-words-on-twitter`

Regular transformation
In social media it is common to transform the endings of words. For English, common transformations are 'ing' $\mapsto$ 'in' and 'er' $\mapsto$ 'a'. Some examples:
thinking $\rightarrow$ thinkin forever $\rightarrow$ foreva

Slang
Novel words are constantly invented on the internet. For this category we use the dictionary created by Tiwari (2015). In line with the normalization datasets we use, we ignore all replacements which consist of multiple words. Some examples:
What $\rightarrow$ Wut These $\rightarrow$ deez

Phonetic transformation
Words can be altered based on their pronunciation. (Proudfoot, 2011) has analyzed this phenomenon for informal SMS language and has identified multiple categories of replacements. To ensure that the replacements are used on the correct pronunciations, the Python library pronouncing[6] is used. Some examples:
tomorrow $\rightarrow$ 2morrow welcome $\rightarrow$ welcum

4.2. Noisy $\mapsto$ Clean

Instead of starting with clean data and artificially creating errors, it is also possible to start with informal text itself and automatically normalize it (unsupervised normalization). When unsupervised normalization is used to generate data, precision should be preferred over recall, hence our method should be conservative. We divide this task in three steps: identification of words in need of normalization, candidate generation, and candidate selection.

4.2.1. Out-of-vocabulary (OOV) word detection

The first step is to detect words in need of normalization. We used the following constrictions to filter words:

- The word is not a punctuation mark
- The word is not in the dictionary (Pyenchant library)
- The word is not a common Twitter construction (RT, hashtag, username)
- The word is not a phrasal abbreviation like: lmao, lol, wtf, omg etc. [7]
- The word is not a data value or metric unit, e.g. 4AM, 17:00 or 17km. (Regular expressions used)

After this detection, we first check whether character repetition is used. If there is a sequence of more than two of the same characters, we reduce this sequence to a length of one and a length of two. If one of the generated options occurs in the Pyenchant library, we consider the replacement correct and skip the following steps.

4.2.2. Candidate generation

To ensure we generate candidates for a variety of replacement types, multiple modules are used:

Brown	SpellCheck	Pyenchant	MoNoise emb.	GloVe emb.	Lookup
holiday	decay	vacation	vacation	vacation	vacation
carvery	vacate	vacate	getaway		
visitation	vacancy	vacant	holiday		
valentine	away	vacancy	hols		
vacation	vary				
	and more..				

Table 1: Example candidate generation using different modules for the word: *vacay*

- Brown clusters: used similarly as described in Section 4.1.1.
- GloVe Twitter embeddings: used similarly as described in Section 4.1.2.
- MoNoise Embeddings: used similarly as described in Section 4.1.2.
- Lookup dictionaries: similar as in Section 4.1.3.
- Pyenchant Python library spellchecker[8]
- Spellchecker Python library [9]

4.2.3. Candidate selection

All the methods mentioned in section 4.2.2. generate a list of candidates. To make a final decision whether a word is replaced, we count for each candidate by how many methods it is generated. Then, we choose to normalize to the most frequently generated candidate, if it is generated at least twice. If there is a second candidate which is equally frequently generated, we choose to keep the original word. Table 1 gives a demonstration of candidates generated by the different systems for the OOV word: 'vacay'. As can be seen from this table, 'vacation' is generated by at least two modules and suggested the most (5 times) out of all different suggestions given. So, the system selects this candidate for the training data.

4.3. Hybrid

To test whether our proposed methods are complementary, we also evaluate a normalization model trained on the concatenation of the two best performing generation strategies. Here, we use the CEG system, and the noisy $\mapsto$ clean strategies, and simply concatenate their generated data, and shuffle before training the normalization model.

5. Results

After generating training data using different methods, the data sets have been used to train a lexical normalization model called MoNoise (van der Goot and van Noord, 2017). There seems to be no general agreement on what evaluation metric should be used for the normalization task. Various researchers have evaluated their models with different metrics. In this research, we use accuracy as main evaluation metric, and precision and recall to gain a more detailed picture. In consonance with (van der Goot, 2019a), a slightly different interpretation of the precision and recall

[6] https://pronouncing.readthedocs.io/en/latest/
[7] list from Tiwari (2015) used to check
[8] https://pypi.org/project/pyenchant/
[9] https://pypi.org/project/pyspellchecker/

	Acc.	Prec.	Rec.	%Norm.	Trainsize
Baseline	89.43			10.53	32,478
Upper bound	95.93	85.05	74.59	10.53	32,478
Brown	92.12	72.28	41.22	8.89	80,000
GloVe-w2v	92.08	68.97	45.55	4.35	50,000
MoNoise-w2v	92.59	69.47	53.28	12.05	60,000
CEG	94.05	74.97	65.57	5.1	60,000
Noisy↦clean	92.49	74.36	44.15	4.67	40,000
Hybrid	93.47	77.44	53.86	4.55	100,000

Table 2: Baseline, and results of MoNoise with human annotated data (upper bound) and our various generated datasets. Trainsize is reported in number of tokens, and is motivated in Section 6.4.. %Norm.: percentage of words normalized. Brown: brown clusters, Glove-w2v: glove word embeddings, MoNoise-w2v: MoNoise word embeddings, CEG: custom error generation, Noisy ↦ Clean: unsupervised normalization

metric will be used. In previous studies, a model was penalized twice after normalizing an error wrongfully. This was accounted in both the precision (FP) and recall (FN). In this evaluation however, these errors affect recall (FN) only.

5.1. Development data

In this section, we report results for the normalization task on the development data (Section 3.2.). As a point of reference, a baseline is set. The baseline considered for this research is a model that returns the original word in all cases. In other words, it does not do any normalization on the given data which makes the baseline accuracy equivalent to the proportion of words which are not normalized. The development part of the LiLiu dataset (Li and Liu, 2014) is used to tune the approaches. To see how MoNoise performs with the generated training sets in comparison to a manually annotated training set, the LiLiu training split will be used as reference as well. This can be considered a theoretical upper bound.

As can be observed from table 2, the CEG tool for creating errors in CLEAN data performs considerably better than all other methods. The CEG approach is apparently better in injecting common mistakes, but also diverse enough to help MoNoise capture errors in the development set which other approaches do not succeed in. The approach which leverages the GloVe Twitter embeddings performs worst on the development set. The MoNoise embeddings and the GloVe approach are similar with the only exception being the word embedding model they adopt. The difference in vocabulary size has resulted in quite a difference in the percentage of words normalized for each approach which, again, could explain the differences in performance. We elaborate on this in Section 6.2..

The hybrid approach does not result in higher performance compared to just using the CEG. This shows that the quality of the noisy↦clean training data is not adequate enough to boost the performance and even lowers the quality of the training data. Additionally, datasets generated by the different methods with clean data as a starting point have been concatenated and randomly shuffled to see how this would influence the results. The sizes used for each corresponding approach are the same as reported in table 2. All possible combinations of the various approaches have been tried. However, combining the datasets of the approaches did not result in better performance compared to using the CEG method alone.

Approach	Acc.	Prec.	Rec.
CEG	94.29	75.64	74.53
MoNoise-w2v	92.04	73.60	48.40
Noisy ↦ clean	91.54	75.23	39.8
Baseline	88.45	-	-
Upper bound	95.22	79.04	79.69

Table 3: Performance of MoNoise on the test data

5.2. Test data

On the test data, we compare the performance of the best approach (CEG), to the best fully unsupervised method (noisy ↦ clean, MoNoise-w2v). We also report the baseline which copies the original word, and the upperbound, which is MoNoise trained on manually annotated data. The results of using CEG training data approaches the results of the upper bound closely; it scores only 0.93 percentage point lower on accuracy. These results also confirm that additional resources are necessary to obtain results comparable to using gold data, as the performance of MoNoise-w2v is substantially lower.

6. Analysis

While generating training data showed promising results, there is still plenty of room for improvement. In this section we explore several factors that lead to a decrease in accuracy for the different types of generated training data. In Section 6.1. we list the types of mistakes which are made, which are then discussed in more detail in their own subsection.

6.1. Type of errors

An aspect that could explain the lower accuracy for the generated training data is the presence of incorrect normalization pairs. Since multiple approaches have been applied to generate training data, we have composed four error categories to distinguish between several error types. Examples are given in table 4.

1. Incorrect error identification This category includes examples where the model incorrectly classifies that a word is in need of normalization. These incorrect cases are mostly caused by the Enchant Library, since this library is used to perform this check. However, when taking context into account, it becomes clear that these words should have been considered for normalization.

2. Incorrect normalization pairs generated with word embedding models While both embedding models generates common anomalies, like *tmrw* or *ppl*, they also generate normalizations which are only semantically related, and thus incorrect. In this category, we place all incorrect

Category	Examples
1.	*cant, ur, nite, wont*
2.	*smiling↦squinting*, *defend↦demean*, *description↦depiction*
3.	*swooning↦sleeping*, *lifter↦leader*
4.	*kpop↦pop*, *yay↦hooray*, *cus↦cause*, *fav↦fave*

Table 4: Examples of errors in the generated training data

normalization pairs produced by both embedding models (MoNoise and GloVe).

3. Incorrect normalization pairs generated with Brown clusters Similar to the word embedding models, using Brown clusters also has the disadvantage that it generates incorrect normalization pairs. Erroneous pairs generated with Brown clusters are included in this category.

4. Unsupervised normalization This category includes all cases where words did not need normalization, but are normalized by the system, or cases which are normalized incorrectly when using `noisy data` as a starting point. While this category has some overlap with the first category (both categories incorrectly identify words as errors), this category contains wrong normalization pairs.

6.2. Vocabulary size of embeddings

Aside to generating incorrect normalization pairs, there is also a huge difference between the performances of the two word embedding models. The results showed that the GloVe embeddings scores the worst out of all approaches. This is due to the difference in vocabulary size. While the MoNoise embeddings model has a vocabulary size of approximately 4.5 million words, the GloVe embeddings has a size of "only" 1.2 million. This is quite a substantial difference. The difference in the percentage out-of-vocabulary (OOV) words is one explanation. However, a higher amount of OOV words doesn't necessarily imply that the performance will be better. That is why we have looked at the training data in more detail of both generated sets, and especially some common used words. For example, the word *before*. The GloVe model didn't generate at least one pair, whereas the MoNoise embeddings generates over 20 options of noise to insert (e.g. *before↦be4* or *before↦bfor*). So this example shows that it misses some interesting opportunities that other models do capture which affects the quality of the data.

6.3. Performance per error category

In this section, the custom error generation tool is analyzed in more detail. An ablation experiment has been run to see which categories contributes to the normalization task most. The results of this experiment are presented in figure 2.

The bar graph shows that leaving out the 'Missing apostrophe' and 'Phonetic transformation' class results in the lowest accuracy. These categories seem therefore to be most important for the normalization task. This can be explained by the fact that these categories appear more frequently in the corpora. This is also in line with the observations by (van der Goot et al., 2018), who annotated the LexNorm2015 dataset. The most common categories were: phonetic transformation, phrasal abbreviation and missing apostrophe. The phrasal abbreviation category is not included in this research but the other two are.

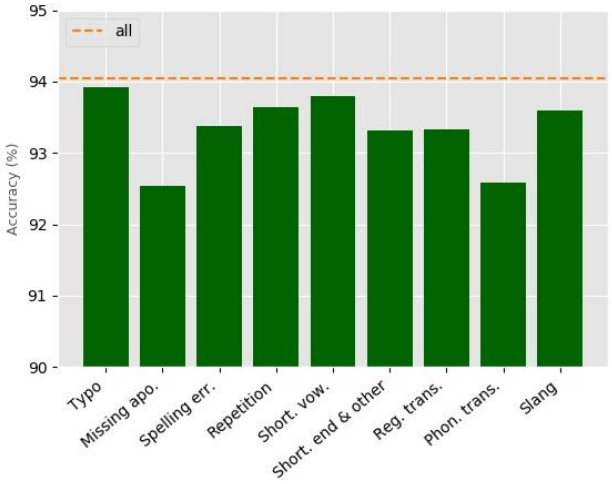

Figure 2: Performance on development data of CEG when leaving a specific category out

The categories which have almost no influence at all are typographical error, shortening vowel and repetition. With the category typographical error, random typos are generated based on the keyboard layout. So the chance of a specific typo appearing in the evaluation set is very small which makes it harder for MoNoise to generate and/or select the correct candidate. For the category shortening vowels, the category shortening other covers some of this. The shortening other category is based on a lookup list, which also includes pairs where vowels are left out, e.g. *abt→about*. The repetition category is simply not so frequent in the data to have a large effect on the performance. Still, a combination of all categories is necessary to reach the highest accuracy: 94.05%.

6.4. Amount of training data

Figure 3 shows the accuracy scores by the amount of training data generated by the different approaches. For each approach, the training starts with 100 tokens. The learning curve of the LiLiu training set is plotted as well for reference. The training set of LiLiu contains a total of 32,478 tokens. All the other approaches have been evaluated to a maximum of 100,000 tokens.

From the graph in figure 3 it becomes clear that using more data doesn't necessarily lead to better scores. This is on one hand unfortunate, since generating a lot of data in a relatively small time period is one of the advantages of using an unsupervised approach. On the other hand, training MoNoise on a smaller set is more efficient. Only 5-10 tweets (100 tokens) are necessary to beat the baseline. There is also a huge improvement for both the LiLiu and CEG when using 1000 tokens. The CEG method even slightly outperforms the LiLiu dataset at that point, however, this is probably due to randomness (in data selec-

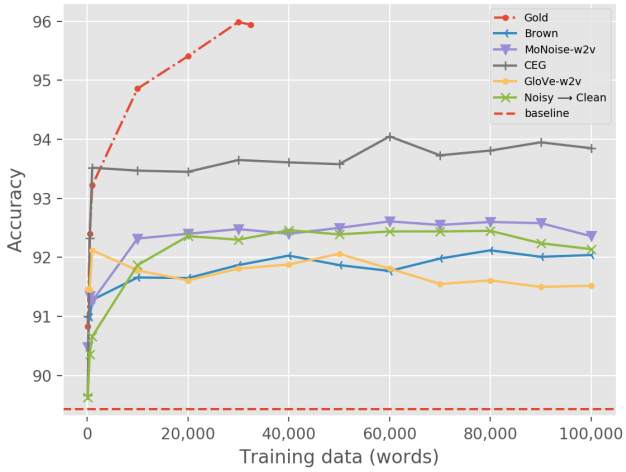

Figure 3: Effect of training data size on accuracy of MoNoise on the development data. Gold: manually annotated normalization data.

tion, model training etc.). All accuracy lines show that it doesn't seem to matter how much data is generated. The line is steep until 20,000 and after that point it becomes rather shallow. For the evaluation (Section 5.), the amount of training data used for each generated training set is the point where the accuracy is the highest.
Some factors that could explain why more data does not necessarily result in higher scores are:

- More data generation leads to more incorrect normalization pairs (section 6.1.)
- More data leads to more incorrect error identification (section 6.1.)
- More data leads to more normalization candidates for certain anomalies
- More data creates more normalization pairs, but most of these pairs are maybe rarely seen in social media text

The first two factors mentioned above have already been discussed in previous sections. The last two will be explained in more detail here. When generating more data, some out-of-vocabulary (OOV) forms can be generated multiple times, but refer to a different in-vocabulary (IV) form. For example, with the MoNoise embeddings, the OOV word *kep* is in a normalization pair with both the IV words *kept* and *keep*. Another example is the OOV token *wrd*, which is either mapped to *word* or *weird*. Both these cases are correct normalization pairs, but it can lead to confusion for the model when presented with such a token since it has seen both pairs. So the model has to choose between the two options and it can choose the wrong option.
Furthermore, while maybe more normalization pairs are created, not all of them are of good quality or are rarely ever used in the social media domain since they are quite specific as an anomaly. This is especially the case with typos (e.g. *theae* or *yzur*). Even though adding errors like this contributes to the variety of errors, these kind of pairs don't contribute to the performance.

7. Human evaluation

In the previous sections, we have shown that automatically generated training data can lead to performance rather close to the scores obtained when using a manually annotated dataset for supervised lexical normalization. Next, we will investigate whether the generated data is comparable to real social media data. In other words; can humans distinguish automatically 'noisified' tweets from real tweets?

7.1. Experimental setup

The CEG approach scored the best during evaluation and will therefore be used for this experiment. A tweet which is generated with the CEG approach will further on be referred to as a machine-generated tweet. To investigate if the generated normalization pairs seem legit in tweets, human annotators have been entrusted with the task to distinguish between human-generated tweets and machine-generated tweets. A human-generated tweet is a tweet from the NOISY TWEETS which have been collected during this research and was thus once really published on Twitter by a user. These tweets are filtered to always contain at least one OOV word.
For this experiment, a balanced sample of 100 tweets was presented to a total of five annotators. The average age of the annotators was 23 years and all annotators are native speakers of Dutch, with an above average English proficiency. Three annotators were male, two were female. Furthermore, four of them are highly-educated, and four of them are active on Twitter.
Every anomaly in the tweet was highlighted in bold to give the annotator a clear indication where the mistake was. This was done because some anomalies are only very small, like a missing apostrophe, and could easily be missed by an annotator. An example from the experiment: *I am very capable of doing this math work but i* ***dont*** *feel like it*
After presenting such a tweet to an annotator, they were asked to chose one out of the following three options:

- Human-generated
- Machine-generated
- I do not know

The first option is where the tweet is actually published once by a Twitter user. The second option refers to tweets which are mutated by the CEG tool during this research. The annotators are also given the choice to indicate that they have no idea to which class the tweet belongs to.

7.2. Results

The results of all the annotators are presented in table 5. Since the 100 tweets contain 50 cases of both classes, a majority-class baseline would reach an accuracy of 50%. The total correct classifications are reported, but also the amount of miss classification per class. The first two columns in table 5 show the amount of cases which were correctly identified by the annotators. The other columns reveal which class is incorrectly classified by an other class. The last column indicates the overall accuracy of each annotator. Interestingly, in general, the human annotators had a bias towards classifying tweets as machine-generated

#	H-H	M-M	H-M	M-H	H-Idk	M-Idk	Accuracy
1	30	33	20	17	0	0	63
2	26	31	24	19	0	0	57
3	31	35	19	15	0	0	66
4	29	34	19	12	2	4	63
5	32	43	18	7	0	0	75
Average	29.6	35.2	20	14	.4	.8	64.8

Table 5: Results of all annotators for classifying the human-generated vs. machine-generated tweets. The category before the - denotes the real class and the category behind the - is the classified class. H: human, M: machine-generated, Idk: classified as unknown.

	Precision	Recall
Machine	63.77	70.4
Human	67.89	59.2

Table 6: Precision and recall for the two classes

From the observations reported in table 5, especially annotator 5 has scored quite good in correctly identifying the machine-generated class. This annotator is an outlier compared to the other four annotators. The other annotators correctly identified between 31 and 35 of the machine-generated classes, while annotator 5 had 43 correct. Still, all annotators were able to beat the overall baseline of 50%.

7.3. Analysis

To analyse the results in more detail, the overall precision and recall is calculated. This is only done for the two most interesting classes: Human-generated and machine-generated. The 'I do not know' class is never a gold option. The goal of this experiment is to determine if human annotators are able to spot the anomalies generated by the CEG tool. So the precision and recall of the machine class is the most interesting to consider. The precision of this class is lower than the recall. This is due to a lot of annotators misclassifying human tweets as machine-generated. The following tweet is an example where all 5 annotators classified the tweet as machine-generated, but it was a human-generated tweet: *"**N ofc** it had to be at a bus stop"*.
During the experiment, there was not one tweet that couldn't be spotted as machine-generated by one of the annotators. So, all machine-generated tweets were at least identified by one of the annotators. However, this was mainly due to one annotator, namely annotator number 5. This person was quite good at identifying the machine class in some cases, while all the other annotators chose the human-generated class. An example of such a tweet where only annotator 5 was correct: *"The last time I flew by myself we had to do an emergency landing **cause** of me... check back for live updates this morning..."*
Even though there were some hard cases and tweets where the annotators were divided, there were also some easy instances where every annotator classified the tweet correctly. Some examples of correctly identified machine-generated tweets by all annotators:

<USERNAME> lol **sew** true

<USERNAME> PS I **wood** appreciate if you followed me as well.
RT <USERNAME> **eyed** give anything just to forget

So, especially tweets containing a word which was replaced by a homophone, were easily identified by the annotators. Also, there were some tweets containing specific characteristics that were all correctly identified as human-generated by the annotators. Another interesting trend which was observed is that some type of error categories were mostly classified as human by the annotators. Tweets in the experiment that contained a repetition of characters were annotated mostly as human-generated. Also, when the mistake is quite small, such as a missing apostrophe, the annotators tended to opt for the human-generated class as well even though they were not human-generated in some cases. Some examples of human-generated tweets correctly classified by all annotators:

why **didnt** i listen to my gut
I am **soo** bad with money man it's now past a joke
RT <USERNAME> <USERNAME> can't wait for the **vid**. #NeverBySons

When looking at the results, all annotators were able to beat the overall baseline of 50%. When looking specifically at the machine class, annotators were better able to spot those than the human class. This was due to some anomalies being quite obvious. To answer the question: "Is the automatically generated data similar to human generated data?", it can be concluded that to extent point the machine-generated tweets have similar characteristics to human-generated tweets. While some tweets were very obvious and correctly classified during the experiment, some characteristics like g-dropping or a missing apostrophe were mostly identified as human by the annotators even though this was not always the case. Furthermore, tweets containing slang were also regarded as a human characteristic by most annotators, and again, not all these instances were always human-generated.

8. Conclusion

Manually annotating data for the lexical normalization is time consuming and expensive. To tackle this problem, we have have focused on the task of automatically generating training data for lexical normalization of English. We have experimented with two directions to generate pairs of sentences: starting with clean text and insert 'noise', or the other way around, which is a form of unsupervised normalization itself. We compared a variety of methods/strategies, and have exploited manually created resources as well as fully unsupervised resources like brown clusters and word embeddings.
To evaluate the training data generated with the different methods, the performance of an existing normalization model (MoNoise) on the generated sets was compared to a manually annotated dataset. Automatically inserting noise in clean text has shown to give the most promising results when compared to the performance of training where noisy

data is used as a basis. Our final system, which automatically inserts non-standardness into sentences (RQ2), scores 94.29 accuracy on the test data, compared to 95.22 when using manually annotated data. This indicates that the performance of the generated data is remarkably close to a model trained on human annotated data (RQ1). Our best strategy that does not make use of manually created resources scores is based on word embeddings, and scores 92.04 accuracy. Furthermore, an experiment has shown that some anomalies created by our approach have human-like characteristics and are hard to distinguish from human generated data (RQ3).

The source code of all experiments and a sample of the generated data is available at:

`https://bitbucket.org/kellydekker/data-generation-lexical-normalization/`

Acknowledgements

We would like to thank Barbara Plank and the anonymous reviewers for the valuable feedback.

9. Bibliographical References

Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

Çolakoğlu, T., Sulubacak, U., and Tantuğ, A. C. (2019). Normalizing non-canonical Turkish texts using machine translation approaches. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 267–272, Florence, Italy, July. Association for Computational Linguistics.

Contractor, D., Faruquie, T. A., and Subramaniam, L. V. (2010). Unsupervised cleansing of noisy text. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 189–196, Stroudsburg, PA, USA. Association for Computational Linguistics.

Doval, Y., Vilares, J., and Gómez-Rodríguez, C. (2020). Towards robust word embeddings for noisy texts. *arXiv preprint arXiv:1911.10876.v3*.

Eisenstein, J. (2013). What to do about bad language on the internet. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 359–369, Atlanta, Georgia, June. Association for Computational Linguistics.

Foster, J. and Andersen, Ø. E. (2009). Generrate: generating errors for use in grammatical error detection. In *Proceedings of the fourth workshop on innovative use of nlp for building educational applications*, pages 82–90. Association for Computational Linguistics.

Gouws, S., Hovy, D., and Metzler, D. (2011). Unsupervised mining of lexical variants from noisy text. In *Proceedings of the First workshop on Unsupervised Learning in NLP*, pages 82–90. Association for Computational Linguistics.

Han, B. and Baldwin, T. (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-Volume 1*, pages 368–378. Association for Computational Linguistics.

Han, B., Cook, P., and Baldwin, T. (2012). Automatically constructing a normalisation dictionary for microblogs. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 421–432. Association for Computational Linguistics.

Kasewa, S., Stenetorp, P., and Riedel, S. (2018). Wronging a right: Generating better errors to improve grammatical error detection. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4977–4983, Brussels, Belgium, October-November. Association for Computational Linguistics.

Li, C. and Liu, Y. (2014). Improving text normalization via unsupervised model and discriminative reranking. In *Proceedings of the ACL 2014 Student Research Workshop*, pages 86–93.

Liu, X., Zhang, S., Wei, F., and Zhou, M. (2011). Recognizing named entities in tweets. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 359–367, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*.

Owoputi, O., O'Connor, B., Dyer, C., Gimpel, K., and Schneider, N. (2012). Part-of-speech tagging for twitter: Word clusters and other advances. *School of Computer Science*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation.

Proudfoot, C. (2011). *An analysis of the relationship between writing skills and Short Messaging Service language: a self–regulatory perspective*. Ph.D. thesis, North-West University.

Schulz, S., Pauw, G. D., Clercq, O. D., Desmet, B., Hoste, V., Daelemans, W., and Macken, L. (2016). Multimodular text normalization of dutch user-generated content. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 7(4):61.

Tiwari, H. (2015). Internet slang dataset. `https://floatcode.wordpress.com/2015/11/28/internet-slang-dataset/`. [Online; accessed 4-June-2019].

van der Goot, R. and van Noord, G. (2017). MoNoise: Modeling noise using a modular normalization system. *Computational Linguistics in the Netherlands Journal*, 7:129–144, 12/2017.

van der Goot, R., van Noord, R., and van Noord, G. (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)*.

van der Goot, R. (2019a). MoNoise: A multi-lingual and easy-to-use lexical normalization tool. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 201–206, Florence, Italy, July. Association for Computational Linguistics.

van der Goot, R. (2019b). *Normalization and parsing algorithms for uncertain input*. Ph.D. thesis, University of Groningen.

Wikipedia:Lists of common misspellings. (2017). Wikipedia:lists of common misspellings — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings`. [Online; accessed 8-May-2019].

Xu, K., Xia, Y., and Lee, C.-H. (2015). Tweet normalization with syllables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 920–928, Beijing, China, July. Association for Computational Linguistics.

Yang, Y. and Eisenstein, J. (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.