Unsupervised Mining of Lexical Variants from Noisy Text

Stephan Gouws[*], **Dirk Hovy** and **Donald Metzler**
stephan@ml.sun.ac.za, {dirkh, metzler}@isi.edu
USC Information Sciences Institute
Marina del Rey, CA
90292, USA

Abstract

The amount of data produced in user-generated content continues to grow at a staggering rate. However, the text found in these media can deviate wildly from the standard rules of orthography, syntax and even semantics and present significant problems to downstream applications which make use of this noisy data. In this paper we present a novel unsupervised method for extracting domain-specific lexical variants given a large volume of text. We demonstrate the utility of this method by applying it to normalize text messages found in the online social media service, Twitter, into their most likely standard English versions. Our method yields a 20% reduction in word error rate over an existing state-of-the-art approach.

1 Introduction

The amount of data produced in user-generated content, e.g. in online social media, and from machine-generated sources such as optical character recognition (OCR) and automatic speech recognition (ASR), surpasses that found in more traditional media by orders of magnitude and continues to grow at a staggering rate. However, the text found in these media can deviate wildly from the standard rules of orthography, syntax and even semantics and present significant problems to downstream applications which make use of this 'noisy' data. In social media this noise might result from the need for social identity, simple spelling errors due to high input cost associated with the device (e.g. typing on a mobile phone), space constraints imposed by the specific medium or even a user's location (Gouws et al., 2011). In machine-generated texts, noise might result from imperfect inputs, imperfect conversion algorithms, or various degrees of each.

Recently, several works have looked at the process of *normalizing* these 'noisy' types of text into more standard English, or in other words, to convert the various forms of idiosyncratic spelling and writing errors found in these media into what would normally be considered standard English orthography. Many of these works rely on supervised methods which share the common burden of requiring training data in the form of noisy input and clean output pairs. The problem with developing large amounts of annotated training data is that it is costly and requires annotators with sufficient expertise. However, the volume of data that is available in these media makes this a suitable domain for applying semi- and even fully unsupervised methods.

One interesting observation is that these noisy out-of-vocabulary (OOV) words are typically formed through some semi-deterministic process which doesn't render them *completely* indiscernible at a lexical level from the original words they are meant to represent. We therefore refer to these OOV tokens as *lexical variants* of the clean in-vocabulary (IV) tokens they are derived from. For instance, in social media '2morrow' '2morow' and '2mrw' still share at least *some* lexical resemblance with 'tomorrow', due to the fact that it is mainly the

[*]This work was done while the first author was a visiting student at ISI from the MIH Media Lab at Stellenbosch University, South Africa.

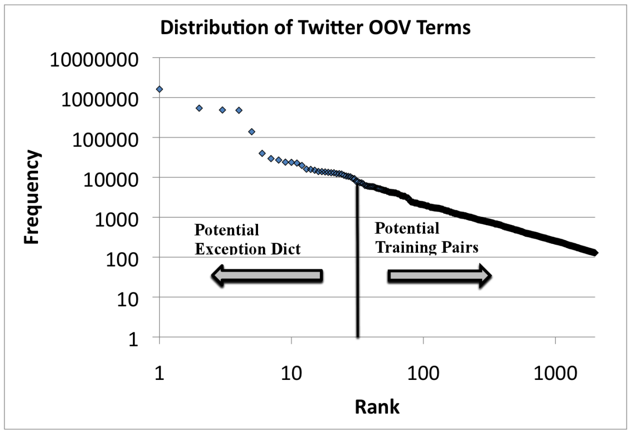

Figure 1: A plot of the OOV distribution found in Twitter. Also indicated is the potential for using (OOV,most-likely-IV) training pairs found on this curve for either exception dictionary entries (the most frequent pairs), or for learning lexical transformations (the long tail). The threshold between the two (vertical bar) is domain-specific.

result of a phonetic transliteration procedure. Also, 'computer' and 'conpu7er' share strong lexical overlap, and might be the result of noise in the OCR process.

As with many aspects of NLP, the distribution of these OOV tokens resemble a power law distribution (see Figure 1 for the OOV distribution in Twitter). Thus, some words are commonly converted to some OOV representation (e.g. domain-specific abbreviations in social media, or words which are commonly incorrectly detected in OCR) and these account for most of the errors, with the rest making up the long tail. If one could somehow automatically extract a list of all the domain-specific OOV tokens found in a collection of texts, along with the most likely clean word (or words) each represents, then this could play a key role in for instance normalizing individual messages. Very frequent (noisy, clean) pairs at the head of the distribution could be used for extracting common domain-specific abbreviations, and word-pairs in the long tail may be used as input to learning algorithms for automatically learning the types of transformations found in these media, as shown in Figure 1.

For example, taking Twitter as our target domain, examples for learning common exception pairs may include '*gf*'→'*girlfriend*'. For learning types of lexical transformations, one might learn from '*thinking*'→'*thinkin*' and '*walking*'→'*walkin*' that '*ng*' could go to '*n*' (known as 'g-clipping').

In this paper we present a novel unsupervised method for extracting an approximation to such a domain-specific list of (noisy, clean) pairs, given only a large volume of representative text. We furthermore demonstrate the utility of this method by applying it to normalize text messages found in the online social media service, Twitter, into their most likely standard English versions.

The primary contributions of this paper are:

- We present an unsupervised method that mines (noisy, clean) pairs and requires only large amounts of domain-specific noisy data
- We demonstrate the utility of this method by incorporating it into a standard method for noisy text normalization, which results in a significant reduction in the word error rate compared to the original method.

2 Training Pair Mining

Given a large corpus of noisy text, our challenge is to automatically mine pairs of domain-specific lexical variants that can be used as training data for a variety of natural language processing tasks. The key challenge is how to develop an effective approach that is both domain-specific and robust to noisy corpora. Our proposed approach requires nothing more than a large "common English" corpus (e.g., a large newswire corpus) and a large corpus of domain text (e.g., a large corpus of Twitter data, a query log, OCR output, etc.). Using these two sources of evidence, the approach mines domain-specific lexical variants in a fully unsupervised manner.

Before describing the details of our approach, we first describe the characteristics that we would like the mined lexical variants to have. First, the variants should be *semantically related* to each other. Pairs of words that are lexically similar, but semantically unrelated are not of particular interest since such pairs can be found using basic edit distance-based approaches. Second, the variants should be *domain-specific*. Variants that capture common English lexical variations (e.g., "running" and "run") can be captured using standard normalization procedures, such

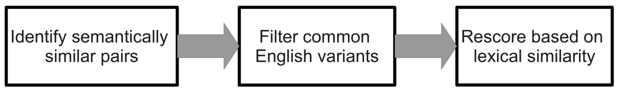
Identify semantically similar pairs
Filter common English variants
Rescore based on lexical similarity

Figure 2: Flow chart illustrating our procedure for mining pairs of lexical variants.

as stemming. Instead, we are interested in identifying domain-specific variations (e.g., "u" and "you" in the SMS and Twitter domains) that cannot easily be handled by existing approaches. Finally, the variants should be *lexically similar*, by definition. Hence, ideal variants will be domain-specific, lexically similar, and semantically related.

To mine such variants we synthesize ideas from natural language processing and large-scale text mining to derive a novel mining procedure. Our procedure can be divided into three atomic steps. First we identify semantically similar pairs, then we filter out common English variants, and finally we rescore the resulting list based on lexical similarity (see Figure 2). The remainder of this section describes the complete details of each of these steps.

2.1 Identifying Semantically Similar Pairs

The first step of our mining procedure harvests semantically similar pairs of terms from both the common English corpus and the domain corpus. There are many different ways to measure semantic relatedness. In this work, we use distributional similarity as our measure of semantic similarity. However, since we are taking a fully unsupervised approach, we do not know *a priori* which pairs of terms may be related to each other. Hence, we must compute the semantic similarity between all possible pairs of terms within the lexicon. To solve this computationally challenging task, we use a large-scale all-pairs distributional similarity approach similar to the one originally proposed by Pasca and Dienes (Pasca and Dienes, 2005). Our implementation, which makes use of Hadoop's MapReduce distributed programming paradigm, can efficiently compute all-pairs distributional similarity over very large corpora (e.g., the Twitter pairs we use later were mined from a corpus of half a billion Twitter messages).

Using a similar strategy as Pasca and Dienes, we define term contexts as the bigrams that appear to the left and to the right of a given word (Pasca and Dienes, 2005). Following standard practice, the contextual vectors are weighted according to pointwise mutual information and the similarity between the vectors is computed using the cosine similarity metric (Lin and Pantel, 2001; Bhagat and Ravichandran, 2008). It is important to note that there are many other possible ways to compute distributional and semantic similarity, and that just about any approach can be used within our framework. The approach used here was chosen because we had an existing implementation. Indeed, other approaches may be more apt for other data sets and tasks.

This approach is applied to both the common English corpus and the domain corpus. This yields two sets of semantically (distributionally) similar word pairs that will ultimately be used to distill unsupervised lexical variants.

2.2 Filtering Common English Variants

Given these two sets of semantically similar word pairs, the next step in our procedure is designed to identify the domain-specific pairs by filtering out the common English variants. The procedure that we follow is very simple, yet highly effective. Given the semantically similar word pairs harvested from the domain corpus, we eliminate all of the pairs that are also found in the semantically similar common English pairs.

Any type of "common English" corpus can be used for this purpose, depending on the task. However, we found that a large corpus of newswire articles tends to work well. Most of the semantically similar word pairs harvested from such a corpus are common lexical variants and synonyms. By eliminating these common variants from the harvested domain corpus pairs, we are left with only the domain-specific semantically similar word pairs.

2.3 Lexical Similarity-Based Re-ordering

The first step of our mining procedure identified semantically similar term pairs using distributional similarity, while the second identified those that were domain-specific by filtering out common English variants. The third, and final, step of our procedure re-orders the output of the second step to account for lexical similarity.

For each word pair (from the second step of our procedure), we compute two scores: 1) a semantic similarity score, and 2) a lexical similarity score. The final score of the pair is then simply the product of the two scores. In this work, we use the cosine similarity score as our semantic similarity score, since it is already computed during the first step of our procedure.

In the social media domain, as in the mobile texting domain, compressed writing schemes typically involve deleting characters or replacing one or more characters with some other characters. For example, users might delete vowels (‘*tomorrow*’→‘*tmrrw*’), or replace ‘*ph*’ with its phonetic equivalent ‘*f*’, as in ‘*phone*’→‘*fone*’. We make use of a subsequence similarity function (Lodhi et al., 2002) which can still capture the structural overlap (in the form of string subsequences) between the remaining unchanged letters in the noisy word and the original clean word from which it was derived. In this work we use a subsequence length of 2, but as with the other steps in our procedure, this one is purposefully defined in a general way. Any semantic similarity score, lexical similarity score, and combination function can be used in practice.

The output of the entire procedure is a scored list of word pairs that are semantically related, domain-specific, and lexically similar, thereby exhibiting the characteristics that we initially defined as important. We treat these (scored) pairs as *pseudo training data* that has been derived in a fully unsupervised manner. We anticipate that these pairs will serve as powerful training data for a variety of tasks, such as noisy text normalization, which we will return to in Section 3.

2.4 Example and Error Analysis

As an illustrative example of this procedure in practice, Table 1 shows the actual output of our system for each step of the mining procedure. To generate this example, we used a corpus of 2GB of English news articles as our “common English” corpus and a corpus of approximately 500 million Twitter messages as our domain corpus. In this way, our goal is to identify Twitter-specific lexical variants, which we will use in the next section to normalize noisy Twitter messages.

Column (A) of the table shows that our distributional similarity approach is capable of identifying a variety of semantically similar terms in the Twitter corpus. However, the list contains a large number of common English variants that are not specific to Twitter. Column (B) shows the outcome of eliminating all of the pairs that were found in the newswire corpus. Many of the common pairs have been eliminated and the list now contains mostly Twitter-specific variants. Finally, Column (C) shows the result of re-ordering the domain-specific pairs to account for lexical similarity.

Rank	**Precision**
P@50	0.90
P@100	0.88

Table 2: Precision at 50 and 100 of the induced exception dictionary.

In our specific case, the output of step 1 yielded a list of roughly 3.3M potential word variants. Filtering out common English variants reduced this to about 314K pairs. In order to estimate the quality of the list we computed the precision at 50 and at 100 for which the results are shown in Table 2. Furthermore, we find that up to position 500 the pairs are still of reasonable quality. Thereafter, the number of errors start to increase noticeably. In particular, we find that the most common types of errors are

1. Number-related: e.g. ‘30’ and ‘30pm’ (due to incorrect tokenization), or ‘5800’ and ‘5530’;

2. Lemma-related: e.g. ‘incorrect’ and ‘incorrectly’; and

3. Negations: e.g. ‘could’ and ‘couldnt’.

Performance can thus be improved by making use of better tokenization, lemmatizing words, filtering out common negations and filtering out pairs of numbers.

Still, the resulting pairs satisfy all of our desired qualities rather well, and hence we hypothesize would serve as useful training data for a number of different Twitter-related natural language processing tasks. Indeed, we will now describe one such possible application and empirically validate the utility of the automatically mined pairs.

(A)	(B)	(C)
i ↔ you	u ↔ you	ur ↔ your
my ↔ the	seeking ↔ seeks	wit ↔ with
u ↔ you	2 ↔ to	to ↔ too
is ↔ was	lost ↔ won	goin ↔ going
a ↔ the	q ↔ que	kno ↔ know
i ↔ we	f*ck ↔ hell	about ↔ bout
my ↔ your	feat ↔ ft	wat ↔ what
and ↔ but	bday ↔ birthday	jus ↔ just
seeking ↔ seeks	ff ↔ followfriday	talkin ↔ talking
me ↔ you	yang ↔ yg	gettin ↔ getting
2 ↔ to	wit ↔ with	doin ↔ doing
am ↔ was	a ↔ my	so ↔ soo
are ↔ were	are ↔ r	you ↔ your
lost ↔ won	amazing ↔ awesome	dnt ↔ dont
he ↔ she	til ↔ till	bday ↔ birthday
q ↔ que	fav ↔ favorite	nothin ↔ nothing
it ↔ that	mostly ↔ partly	people ↔ ppl
f*ck ↔ hell	northbound ↔ southbound	lil ↔ little
can ↔ could	hung ↔ toned	sayin ↔ saying
im ↔ its	love ↔ miss	so ↔ sooo

Table 1: Column (A) shows the highest weighted distributionally similar terms harvested from a large Twitter corpus. Column (B) shows which pairs from (A) remain after filtering out distributionally similar word pairs mined from a large news corpus. Column (C) shows the effect of reordering the pairs from (B) using a string similarity kernel.

3 Deriving A Common Exception Dictionary for Text Normalization as a Use Case for Mining Lexical Variants

As discussed in Section 1, these training pairs may aid methods which attempt to normalize noisy text by translating from the ill-formed text into standard English. Since the OOV distribution in noisy text mostly resemble a power law distribution (see Figure 1), one may use the highest scoring training pairs to induce 'exception dictionaries' (lists of *(noisy word)→(most likely clean word)*) of the most common domain-specific abbreviations found in the text.

We will demonstrate the utility of our derived pairs in one specific use case, namely inducing a domain-specific exception dictionary to augment a vanilla normalization method. We leave the second proposed use-case, namely using pairs in the long tail for learning transformation rules, for future work.

We evaluate the first use case in Section 4.

3.1 Baseline Normalization Method

We make use of a competitive heuristic text normalization method over Twitter data as a baseline, and compare its accuracy to an augmented method which makes use of an automatically induced exception dictionary (using the method described in Section 2) as a first step, before resorting to the same baseline method as a 'back-off' for words not found in the dictionary.

As we point out in Section 5, there are various metaphors within which the noisy text normalization problem has been approached. In general, however, the problem of noisy text normalization may be approached by using a three step process (Gouws et al., 2011):

1. In the **out-of-vocabulary (OOV) detection** step, we detect unknown words which are candidates for normalization

2. In the **candidate selection** step, we find the weighted lists of most likely candidates (from a list of in-vocabulary (IV) words) for the OOV words and group them into a confusion set. The

confusion sets are then appended to one another to create a confusion- network or lattice

3. Finally, in the **decoding** step, we use a language model to rescore the confusion network, and then find the most likely posterior path (Viterbi path) through this network.

The words at each node in the resulting posterior Viterbi path represents the words of the hypothesized original clean sentence.

In this work, we reimplement the method described in Contractor (2010) as our baseline method. We next describe the details of this method in the context of the framework presented above. See (Gouws et al., 2011) for more details.

OOV DETECTION is a crucial part of the normalizaton process, since false-positives will result in undesirable attempts to 'correct' IV words, hence bringing down the method's accuracy. We implement OOV detection as a simple lexicon-lookup procedure, with heuristics for handling specific out-of-vocabulary-but-valid tokens such as hash tags and @usernames.

CANDIDATE SELECTION involves comparing an unknown OOV word to a list of words which are deemed in-vocabulary, and producing a top-K ranked list with candidate words and their estimated probabilities of relevance as output. This process requires a function with which to compute the similarity or alternatively, distance, between two words. More traditional string-similarity functions like the simple Lehvenshtein string edit distance do not fare too well in this domain.

We implement the IBM-similarity (Contractor et al., 2010) which employs a slightly more advanced similarity function. It finds the length of the longest common subsequence (LCS) between two strings s_1 and s_2, normalized by the edit distance (ED) between the consonants in each string (referred to as the 'consonant skeleton' (CS)), thus

$$\text{sim}(s_1, s_2) = \frac{\text{LCS}(s_1, s_2)}{\text{ED}(\text{CS}(s_1), \text{CS}(s_2))}$$

Finally, the **DECODING** step takes an input word lattice (lattice of concatenated, weighted confusion sets), and produces a new lattice by incorporating the probabilities from an n-gram language model with the prior probabilities in the lattice to produce a reranked posterior lattice. The most likely (Viterbi) path through this lattice represents the decoded clean output. We use SRI-LM (Stolcke, 2002) for this.

3.2 Augmenting the Baseline: Our Method

In order to demonstrate the utility of the mined lexical variant pairs, we first construct a (noisy, clean) lookup table from the mined pairs. We (arbitrarily) use the 50 mined pairs with the highest overall combined score (see Section 2.3) for the exception dictionary. For each pair, we map the OOV term (noisy and typically shorter) to the IV term (clean and usually longer). The exception lookup list is then used to augment the baseline method (see Section 3.1) in the following way: When the method encounters a new word, it first checks to see if the word is in the exception dictionary. If it is, we normalize to the value in the dictionary. If it is not, we pass the ill-formed word to the baseline method to proceed as normal.

4 Evaluation

4.1 Dataset

We make use of the Twitter dataset discussed in Han (2011). It consists of a random sampling of 549 English tweets, annotated by three independent annotators. All OOV words were pre-identified and the annotators were requested to determine the standard form (gold standard) for each ill-formed word.

4.2 Evaluation Metrics

In this study, we are interested in measuring the quality of our mined training pairs by evaluating its utility on an external task: Using the training pairs to induce a (noisy$\rightarrow$clean) exception dictionary to augment the working of a standard noisy text normalization system. Hence, our focus is entirely on the accuracy of the candidate selection procedure as defined in Section 3.1. We compute this accuracy in terms of the word error rate (WER), defined as the number of token substitutions, insertions or deletions one has to make to turn the system output into the gold standard, normalized by the total number of tokens in the output. In order to remove the possible bias introduced by our very basic OOV-detection

Method	WER	% Change
Naive baseline	10.7%	–
IBM-baseline	7.8%	-27.1%
Our method	$\mathbf{5.6\%}$	$\mathbf{-47.7\%}$

Table 3: Word error rate (WER, lower is better) results of our method against a naive baseline and the much stronger IBM-baseline (Contractor et al., 2010). We also show the relative change in WER for our method and the IBM-baseline compared to the naive baseline.

mechanism, we evaluate the output of all systems only on the *oracle pairs*. Oracle pairs are defined as the (input,system-output,gold) pairs where input and gold do not match. In other words, we remove the possible confounding impact of imperfect OOV detection on the accuracy of the normalization process by assuming a perfect OOV-detection step.

4.3 Discussion of Results

The results of our experiments are displayed in Table 3. It is important to note that the focus is not on achieving the best WER compared to other systems (although we achieve very competitive scores), but to evaluate the *added utility* of integrating an exception dictionary which is based purely on the mined (noisy, clean) pairs with an already competitive baseline method.

The 'naive baseline' shows the results if we make no changes to the input tokens for all oracle pairs. Therefore it reflects the total level of errors that are present in the corpus.

The IBM-method is seen to reduce the amount of errors by a substantial 27.1%. However, the augmented method results in a further 20.6% reduction in errors, for a total reduction of 47.7% of all errors in the dataset, compared to the IBM-baseline's 27.1%.

Since we replace matches in the dictionary indiscriminately, and since the dictionary comprise those pairs that typically occur most frequently in the corpus from which they were mined, it is important to note that if these pairs are of poor quality, then their sheer frequency will drive the overall system accuracy down. Therefore, the accuracy of these pairs are strongly reflected in the WER performance of the augmented method.

Noisy	Clean	% Oracle Pairs
u	you	8.7
n	and	1.4
ppl	people	1
da	the	1
w	with	0.7
cuz	because	0.5
y	why	0.5
yu	you	0.5
lil	little	0.5
dat	that	0.5
wat	what	0.4
tha	the	0.4
kno	know	0.4
r	are	0.4

Table 4: Error analysis for all (noisy, clean) normalizations missed by the vanilla IBM-baseline method, but included in the top-50 pairs used for constructing the exception dictionary. We also show the percentage of all oracle pairs that are corrected by including each pair in an exception dictionary.

Table 4 shows the errors missed by the IBM-baseline, but contained in the mined exception dictionary. We also show each pair's frequency of occurrence in the oracle pairs (hence its contribution towards lowering WER).

5 Related work

To the best of our knowledge, we are the first to address the problem of mining pairs of lexical variants from noisy text in an unsupervised and purely statistical manner that does not require aligned noisy and clean messages. To obtain aligned clean and noisy text without annotated data implies the use of some normalizing method first. Yvon (2010) presents one such approach, where they generate exception dictionaries from their finite-state system's normalized output. However, their method is still trained on annotated training pairs, and hence supervised. A related direction is 'transliteration mining' (Jiampojamarn et al., 2010) which aims to automatically obtain bilingual lists of names written in different scripts. They also employ string-similarity measures to find similar string pairs written in different scripts. However, their input data is constrained to Wikipedia articles written in different languages, whereas we impose no constrains on our input data, and merely require a large collection thereof.

Noisy text normalization, on the other hand, has recently received a lot of focus. Most works construe the problem in the metaphors of either machine translation (MT) (Bangalore et al., 2002; Aw et al., 2006; Kaufmann and Kalita, 2010), spelling correction (Choudhury et al., 2007; Cook and Stevenson, 2009), or automated speech recognition (ASR) (Kobus et al., 2008). For our evaluation, we developed an implementation of Contractor (2010) which works on the same general approach as Han (2011).

6 Conclusions and Future Work

The ability to automatically extract lexical variants from large noisy corpora has many practical applications, including noisy text normalization, query spelling suggestion, fixing OCR errors, and so on. This paper developed a novel methodology for automatically mining such pairs from a large domain-specific corpus. The approach makes use of distributional similarity for measuring semantic similarity, a novel approach for filtering common English pairs by comparing against pairs mined from a large news corpus, and a substring similarity measure for re-ordering the pairs according to their lexical similarity.

To demonstrate the utility of the method, we used automatically mined pairs to construct an unsupervised exception dictionary, that was used in conjunction with a string similarity measure, to form a highly effective hybrid noisy text normalization technique. By exploiting the properties of the power law distribution, the exception dictionary can successfully correct a large number of cases, while the heuristic string similarity-based approach handled many of the less common test cases from the tail of the distribution. The hybrid approach showed substantial reductions in WER (around 20%) versus the string similarity approach, hence validating our proposed approach.

For future work we are interested in exploiting the (noisy, clean) pairs contained in the long tail as input to learning algorithms for acquiring domain-specific lexical transformations.

Acknowledgments

Stephan Gouws would like to thank MIH Holdings Ltd. for financial support during the course of this work.

References

A.T. Aw, M. Zhang, J. Xiao, and J. Su. 2006. A phrase-based statistical model for SMS text normalization. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 33–40. Association for Computational Linguistics.

S. Bangalore, V. Murdock, and G. Riccardi. 2002. Bootstrapping bilingual data using consensus translation for a multilingual instant messaging system. In *Proceedings of the 19th International Conference on Computational Linguistics Volume 1*, pages 1–7. Association for Computational Linguistics.

Rahul Bhagat and Deepak Ravichandran. 2008. Large scale acquisition of paraphrases for learning surface patterns. In *Proceedings of ACL-08: HLT*, pages 674–682, Columbus, Ohio, June. Association for Computational Linguistics.

M. Choudhury, R. Saraf, V. Jain, A. Mukherjee, S. Sarkar, and A. Basu. 2007. Investigation and modeling of the structure of texting language. *International Journal on Document Analysis and Recognition*, 10(3):157–174.

D. Contractor, T.A. Faruquie, and L.V. Subramaniam. 2010. Unsupervised cleansing of noisy text. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 189–196. Association for Computational Linguistics.

P. Cook and S. Stevenson. 2009. An unsupervised model for text message normalization. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 71–78. Association for Computational Linguistics.

S. Gouws, D. Metzler, C. Cai, and E. Hovy. 2011. Contextual Bearing on Linguistic Variation in Social Media. In *Proceedings of the ACL-11 Workshop on Language in Social Media*. Association for Computational Linguistics.

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*. Association for Computational Linguistics.

S. Jiampojamarn, K. Dwyer, S. Bergsma, A. Bhargava, Q. Dou, M.Y. Kim, and G. Kondrak. 2010. Transliteration generation and mining with limited training resources. In *Proceedings of the 2010 Named Entities Workshop*, pages 39–47. Association for Computational Linguistics.

M. Kaufmann and J. Kalita. 2010. Syntactic Normalization of Twitter Messages. In *International Conference on Natural Language Processing, Kharagpur, India*.

C. Kobus, F. Yvon, and G. Damnati. 2008. Normalizing SMS: are two metaphors better than one? In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 441–448. Association for Computational Linguistics.

Dekang Lin and Patrick Pantel. 2001. Discovery of inference rules for question-answering. *Nat. Lang. Eng.*, 7:343–360, December.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. 2002. Text classification using string kernels. *The Journal of Machine Learning Research*, 2:419–444.

Marius Pasca and Pter Dienes. 2005. Aligning needles in a haystack: Paraphrase acquisition across the web. In Robert Dale, Kam-Fai Wong, Jian Su, and Oi Yee Kwong, editors, *Natural Language Processing IJCNLP 2005*, volume 3651 of *Lecture Notes in Computer Science*, pages 119–130. Springer Berlin / Heidelberg.

A. Stolcke. 2002. SRILM-an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, volume 2, pages 901–904. Citeseer.

F. Yvon. 2010. Rewriting the orthography of sms messages. *Journal of Natural Language Engineering*, 16(02):133–159.