Unsupervised Text Normalization Using Distributed Representations of Words and Phrases

Vivek Kumar Rangarajan Sridhar*
AT&T Labs - Research
1 AT&T Way, Bedminster, NJ 07920

Abstract

Text normalization techniques that use rule-based normalization or string similarity based on static dictionaries are typically unable to capture domain-specific abbreviations (*custy, cx* → *customer*) and shorthands (*5ever, 7ever* → *forever*) used in informal texts. In this work, we exploit the property that noisy and canonical forms of a particular word share similar context in a large noisy text collection (millions or billions of social media feeds from Twitter, Facebook, etc.). We learn distributed representations of words to capture the notion of contextual similarity and subsequently learn normalization lexicons from these representations in a completely unsupervised manner. We experiment with linear and non-linear distributed representations obtained from log-linear models and neural networks, respectively. We apply our framework for normalizing customer care notes and Twitter. We also extend our approach to learn phrase normalization lexicons (*g2g* → *got to go*) by training distributed representations over compound words. Our approach outperforms Microsoft Word, Aspell and a manually compiled urban dictionary from the Web and achieves state-of-the-art results on a publicly available Twitter dataset.

1 Introduction

Text normalization is a prerequisite for a variety of tasks involving speech and language. Most natural language processing (NLP) tasks require a tight and compact vocabulary to reduce the model complexity in terms of feature size. As a consequence, applications such as syntactic tagging and parsing, semantic tagging, named entity extraction, information extraction, machine translation, language models for speech recognition, etc., are trained on clean data that is normalized and restricted to some user defined vocabulary. Conventionally, most NLP researchers perform such normalization through rule-based mapping that can get unwieldily and cumbersome for extremely noisy texts as in SMS, chat or social media.

Unnormalized text, as witnessed in social media forums such as Facebook, Twitter and message boards, or short messaging service (SMS), have a variety of issues with spelling that include repeating letters, eliminating vowels, using phonetic spellings, substituting letters (typically syllables) with numbers, using shorthands and user created abbreviations for phrases. The remarkable property of such texts is that new variants of canonical words and phrases evolve constantly (e.g., *jghome* → *just got home*). Hence, it is important to design a framework that can learn the mapping between unnormalized and canonical forms of such words and phrases in an unsupervised and extensible manner.

Conventional edit distance (Levenshtein, 1966) based approaches are not accurate for predicting spelling correction for large number of edits in abbreviations and shorthands found in informal texts. In this work, we exploit the property that noisy and canonical forms of a particular word share similar context in a large noisy text collection (millions or billions of social media feeds from Twitter, Facebook, etc.). We represent the words in a vector space using distributed representations to capture the notion contextual similarity and subsequently learn normalization lexicons from these representations. The distributed representations are induced either through neural networks (non-linear embeddings) or log-linear models (linear embeddings). The proposed approach uses the property of contextual similarity between canonical and noisy versions of a particular word to cluster them in $\mathbb{R}^D$, where D is the dimension of the distributed representation. We also extend our framework to learn one-to-many mappings (e.g., *ily* → *i love you*, *nbd* → *no big deal*

* The author is currently with Apple, Inc., and can be contacted at vrangarajansridh@apple.com.

Proceedings of NAACL-HLT 2015, pages 8–16,
Denver, Colorado, May 31 – June 5, 2015.

by learning distributed representations over words and phrases.

We demonstrate the fidelity of our approach on customer care domain and Twitter. We also compare our approach with Microsoft Word, Aspell, custom dictionaries compiled from the Web as well as state-of-the-art techniques for unsupervised normalization.

2 Related Work

Text normalization has been traditionally performed in a task specific manner through string edit operations. While a large proportion of NLP researchers still perform this exercise manually by writing regular expression patterns, several automatic procedures have been proposed. A simple way to perform this string edit operation is by using a noisy channel model (Brill and Moore, 2000). However, this requires supervised training data in the form of the canonical and erroneous strings. Since words are spoken using phonetics, it is instructive to look at the problem from the point of pronunciation changes. For example, (Toutanova and Moore, 2002) extended the noisy channel framework to include word pronunciation information. The **aspell** tool for spelling correction also works on a phonetic algorithm for string normalization (Philips, 1990).

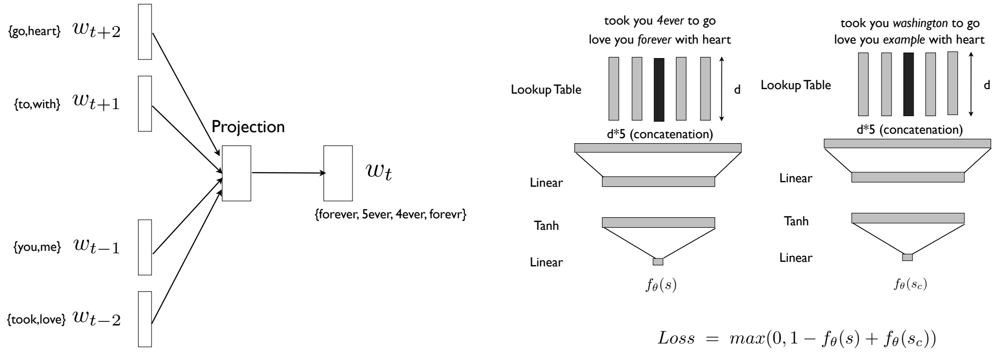
(Cook and Stevenson, 2009) introduced an unsupervised noisy channel model that considered several word formation processes in a generative model. Another popular way to normalize or even punctuate text is by using phrase-based machine translation. (Aw et al., 2009) used a character level phrase-based machine translation approach to translate SMS text into clean English text. However, such an approach still requires supervised training data. Furthermore, noisy channel models typically do not use wider context in resolving the normalization problem. Clearly, many of the unnormalized forms appear in the same context as the canonical form and exploiting such information is critical.

Social media text normalization using contextual graph random walks was recently proposed in (Hassan and Menezes, 2013). They use a lexicon based approach where the normalization lexicon is obtained in an unsupervised manner by performing random walks on contextual similarity graphs (bipartite) constructed from n-gram sequences. A similar approach using distributional similarity was also proposed in (Han et al., 2012a) where a pairwise similarity deems two words with identical context to be normalization equivalences. Due to the pairwise computation, it does not result in a globally optimized equivalence. Our framework is most similar to (Hassan and Menezes, 2013) as we also use the notion of distributional similarity between strings at a corpus level to identify normalization equivalences in an unsupervised manner. In contrast, we use distributed representation of words to capture contextual similarity and learn unsupervised lexicons using both lexical and vector space feature functions. The proposed approach is relatively simple, scalable and easily reproducible.

3 Distributed Representation of Words

Conventional NLP applications typically use one-hot encoding where each word in the vocabulary is represented by a bit vector. Such a representation exacerbates the data sparsity problem and does not exploit any semantic or syntactic relationship that may be present amongst subset of words. Distributed representation of words (also called word embeddings or continuous space representation of words) has become a popular way for capturing distributional similarity (lexical, semantic or even syntactic) between words. The basic idea is to represent each word $w_i \in V$ with a real-valued vector of some fixed dimension D, i.e., $w_i \in \mathbb{R}^D \quad \forall \quad i = 1, \cdots, V$. The idea of representing words in vector space was originally proposed in (Rumelhart et al., 1986; Elman, 1991). However, improved training techniques and tools in the recent past have made it possible to obtain such representations for large vocabularies.

Distributed representations can be induced for a given vocabulary V in several ways. While they are typically induced in the context of a deep neural network framework for a given task (Bengio et al., 2003; Collobert and Weston, 2008; Bengio et al., 2009; Turian et al., 2010; Mikolov et al., 2010), recent work in (Mikolov et al., 2013) has also shown that they can also be induced by using simple log-linear models. Since in many practical NLP applications, the distributed representations are learnt along with the task, the word vectors will have some notion of task dependent distributional similarity. It is this exact notion of contextual and distributional similarity that we exploit in this work to learn normalization lexicons in an un-



(a) Continuous Bag-of-Words Architecture

(b) Neural Network Architecture

Figure 1: Illustration of obtaining distributed representations for text normalization using two different approaches

supervised manner.

Figure 1 shows two different architectures for inducing distributed representations. On the left side, the architecture for the continuous bag-of-words model (Mikolov et al., 2013) is shown while the neural network learning architecture for inducing distributed representations in language models (Collobert and Weston, 2008) is shown on the right. Both these frameworks essentially perform a similar function in that the word representations are created based on contextual similarity. Figure 1 also shows an example of the contextual similarity that can be exploited such that canonical and noisy versions of a particular word have similar vector representation (in terms of some similarity metric). It is shown that the words {*forever, 4ever, 5ever, forevr*} share similar context. It is also interesting to note that the word *5ever* that is used to mean *longer than 4ever* can be identified to mean *forever* that edit distance matching is typically not able to capture.

	Language	
	en	
Corpus	Vocabulary	#Sentences
Customer care	7846840	870491324
Twitter	8371078	178770137

Table 1: Statistics of the data used to learn distributed representations

4 Data

We use two sources of data in our work. One is internal anonymized customer care notes and the other is Twitter. The customer care data refers to notes made by agents at mobility call centers when customers make a call. Each call typically results in one record and the notes typically consist of a brief summary of the call from the representative side. The data we use does not contain any meta-data beyond the text description. We used all the data between Dec 2012 and Jan 2014. The text data is extremely noisy as the agents are making these notes either during their interaction or immediately afterward. Hence, the data contains several spelling errors and abbreviations that need to be corrected before performing any large scale data analytics.

We also acquired a 10% random sample of Twitter firehose data across all languages. As a first step, we filtered the tweets by language code. Since the language code is a property set in the user profile, the language code does not guarantee that all tweets are in the same language. We used a simple frequency threshold for language identification based on language specific word lists. Subsequently, we performed some basic clean-up such as replacing usernames, hashtags, web addresses and numerals with generic symbols such as *_user_*, *_hashtags_*, *_url_* and *_number_*. Finally, we removed all punctuations from the strings and lowercased the text. In this work, we perform our experiments on English.

5 Training Distributed Representations

We used two approaches (see Figure 1) for learning both linear and non-linear distributed representations of words. For the non-linear neural network approach, we used an architecture identical to that in (Collobert and Weston, 2008), i.e., the network consisted of a lookup table, hidden layer with 100 nodes and a linear layer with one output. However, we used a right and left context of 5 (or 7) words and corrupted the centre word instead of the last word to learn the distributed representations. Given a text window $s = \{w\}_1^{wlen}$, $wlen$[1] is the window length, and a set of parameters associated with the network θ, the network outputs a score $f_\theta(x)$. The approach then minimizes the ranking criterion with respect to θ such that:

$$\theta \mapsto \sum_{s \in \mathcal{X}} \sum_{w \in \mathcal{V}} max\{0, 1 - f_\theta(s) + f_\theta(s_c)\} \quad (1)$$

where $\mathcal{X}$ is the set of all windows of length $wlen$ in the training data, $\mathcal{V}$ is the vocabulary and s_c denotes the corrupted version of s with the middle word replaced by a random word w in $\mathcal{V}$. We used a frequency threshold of 10 occurrences for the centre word, i.e., all words below this frequency was not considered in training. We performed stochastic gradient minimization over 1000 epochs on each dataset and used the Torch toolkit (Collobert et al., 2011) to train the representation.

We also used a log-linear model for inducing the distributed representations using the continuous-bag-of-words architecture proposed in (Mikolov et al., 2013). The continuous-bag-of-words model is similar to the neural network language model (Bengio et al., 2003) with the non-linear layer replaced by a sum pooling layer, i.e., the model uses a bag of surrounding words to predict the centre word. Since the implementation of this architecture was readily available through the word2vec tool[2], we used it for inducing the representations. We used hierarchical sampling for reducing the vocabulary during training and used a minimum count of 10 occurrences for each word.

The framework presented in this paper can also work with word vectors obtained using other techniques such as latent semantic indexing, convolutional neural networks, recurrent neural networks, etc.

6 Learning Normalization Lexicons

Once we obtain the set of word embeddings $w_i \mapsto \mathbf{d}_i$, $\forall i \in V; \mathbf{d}_i \in \mathbb{R}^D$, our framework requires a list of canonical words as input. For English, we used a wordlist from Project Gutenberg (`http://www.gutenberg.org/ebooks/3201`) consisting of 113809 words. Given a canonical word **s1**, we find the K-nearest neighbors in the vector space and objectively measure the similarity between **s1** and the neighbors, i.e., from each pair of strings **s1** and **s2** with corresponding vectors **u** and **v**, we obtain lexical and vector space features described below.

6.1 Similarity Cost

The cosine distance between two D-dimensional vectors **u** and **v** is defined as,

$$cosine\ \ similarity = \frac{\sum_{i=1}^{D} u_i \times v_i}{\sqrt{\sum_{i=1}^{D} (u_i)^2 \times \sum_{i=1}^{D} (v_i)^2}} \quad (2)$$

The lexical similarity cost is computed similar to that presented in (Hassan and Menezes, 2013).

$$lexical\ \ similarity(\mathbf{s1}, \mathbf{s2}) = \frac{LCSR(\mathbf{s1}, \mathbf{s2})}{ED(\mathbf{s1}, \mathbf{s2})} \quad (3)$$

$$LCSR(\mathbf{s1}, \mathbf{s2}) = \frac{LCS(\mathbf{s1}, \mathbf{s2})}{Max\ Length(\mathbf{s1}, \mathbf{s2})} \quad (4)$$

where LCSR refers to the Longest Common Subsequence Ratio (Melamed, 1995), LCS refers to Longest Common Subsequence and ED refers to the edit distance between the two strings. For English, the edit distance computation was modified to find the distance between the consonant skeleton of the two strings **s1** and **s2**, i.e., all the vowels were removed. Repetition in the strings was reduced to a single letter and numbers in the words were substituted by their equivalent letters. The general algorithm for learning a normalization lexicon through our approach is presented in Algorithm 1. While it is possible to learn optimal weights for several feature functions through minimum error rate training (Och, 2003), we use uniform weights in the absence of a significant held-out set for optimization.

6.2 Representation of Lexicons using FSTs

We compile the lexicon $\mathcal{L}$ obtained using Algorithm 1 into a finite-state transducer with the arc score equal to the negative logarithm of the similarity cost (for finding the path with least cost).

[1] $wlen$ in our work is an odd number, e.g., $wlen = 11$ implies a left and right context of 5 words

[2] https://code.google.com/p/word2vec/

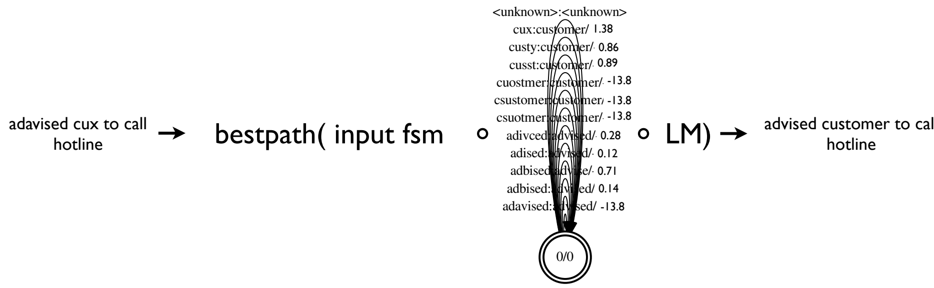


Figure 2: Illustration of the normalization technique using finite state transducers. The unknown words in the input are preserved in the output.

Algorithm 1 Unsupervised Lexicon Learning

input $\{\mathbf{d}_i\}_{i=1}^{|V|}$: distributed representation of words for vocabulary $|V|$
input *K*: number of nearest neighbors
input *COST*: lexical similarity metric
input *W*: list of canonical words
for each $w \in W$ **do**
  for each $i \in |V|$ **do**
    if $w_i \mapsto \mathbf{d}_i \notin W$ **then**
      Compute cosine distance between $\mathbf{d}_i$ and $\mathbf{d}(w)$
      Store top *K* neighbors in map *L*(*w*)
for each $w \in W$ **do**
  for each $o \in L(w)$ **do**
    Compute *COST(w,o)*
    Push $w \mapsto \{o, COST(w,o)\}$ into *D*
Invert the map *D* to obtain lexicon $\mathcal{L}$

The normalization lexicon is converted into a single state finite-state transducer (FST) with the input and output labels being the noisy and canonical word, respectively. In all our experiments, we used the number of nearest neighbors $K = 25$.

Given a sentence that needs to be normalized, we form a linear FSM s from the text string and compose it with the FST lexicon $\mathbf{N}$. The resulting FSM is then composed with a language model (LM) $\mathbf{L}$, if available, and the best path is found to obtain s_{norm}. We used a trigram language model that was trained on a variety of texts (English Gigaword, Web, Opensubtitles, etc.). We used Kneser-Ney discounting and the LM was not optimized in any way.

$$s_{norm} = bestpath(s \circ \mathbf{N} \circ \mathbf{L}) \tag{5}$$

Figure 2 illustrates this procedure. The unknown words in the input are preserved in the output (the language model is trained with an open vocabulary).

6.3 Evaluation

First, we evaluated our approach on customer care data. A set of 300 sentences from the customer care data was randomly selected and the reference sentences were created manually by a professional transcriber. A total of 2387 tokens were normalized by the transcribers. The distributed representation was trained on the remaining customer care data through neural network learning approach (Collobert and Weston, 2008) over a window of 11 words with a vector dimension of 100. We compare our approach with Microsoft Word and Aspell, where the best option was manually chosen (oracle) from the suggestion list. If no option was appropriate, the word was left in it's original form. We measure the fidelity of normalization using precision and recall. The results are presented in Table 2.

Tokens	Model	Precision (%)	Recall (%)	F1 (%)
All	Microsoft Word (Oracle)	53.2	55.0	54.0
	Aspell (Oracle)	33.0	41.2	36.7
	Our approach without LM	59.8	58.2	59.0
	Our approach with LM	64.2	70.4	67.1
Edit distance > 2	Microsoft Word (Oracle)	62.3	49.7	55.3
	Aspell (Oracle)	41.9	36.4	38.9
	Our approach without LM	44.4	58.2	50.4
	Our approach with LM	50.61	75.1	60.5

Table 2: Sentence level normalization on customer care notes

Our approach achieves good performance on the customer care notes. We achieve precision and recall of 64.2% and 70.4%. The performance using our approach outperforms the oracle accuracy using Microsoft Word and Aspell. It is important

Category	Model	Precision (%)	Recall (%)	F1 (%)
Word	Microsoft Word (Oracle)	72.7	30.8	43.3
	Aspell (Oracle)	83.0	35.4	49.6
	Web dictionary with LM	79.8	24.2	37.1
	Neural Network ($wlen$:11+D:100) lexicon without LM	53.4	74.7	62.3
	Neural Network ($wlen$:11+D:100) lexicon with LM	54.4	77.1	63.8
	Neural Network ($wlen$:11+D:200) lexicon with LM	50.5	75.1	60.4
	Neural Network ($wlen$:15+D:100) lexicon with LM	54.2	73.5	62.4
	Neural Network ($wlen$:15+D:200) lexicon with LM	48.5	75.4	59.0
	Log-Linear Model ($wlen$:11+D:100) lexicon with LM	54.5	77.2	63.9
	Log-Linear Model ($wlen$:11+D:200) lexicon with LM	51.2	75.9	61.1
	Log-Linear Model ($wlen$:15+D:100) lexicon with LM	54.6	76.1	63.5
	Log-Linear Model ($wlen$:15+D:200) lexicon with LM	47.1	75.1	57.9

Table 3: Sentence level word normalization on English Twitter data

to note that while our approach is customized to the domain, the baseline comparisons are not. The performance for noisy words that differ in edit distance by more than 2 from the canonical word is also shown in Table 2. Our framework achieves significantly better performance for abbreviations that typically have edit distance > 2. Since our approach combines the strength of distributional and lexical similarity as opposed to most approaches that rely only on string similarity, we are also able to correctly normalize domain specific abbreviations, e.g., *custy* → *customer*, *cx* → *customer*, *lqd* → *liquid*, *bal* → *balance*, *exp* → *expectations*, etc. The use of a language model significantly improves the normalization accuracy.

We also performed sentence (tweet) level normalization on Twitter data. We manually annotated (expanded abbreviations, shorthands and spelling errors) 1000 tweets and performed normalization using our approach. The annotation was performed serially by two professional transcribers. We compare our approach with Microsoft Word, Aspell and a dictionary compiled from several websites. We use a log-linear model (continuous-bag-of-words) as well as a neural network (see Section 5) to automatically learn normalization lexicons. For each model, we experimented with window length ($wlen$) of 11 and 15 while the dimension of distributed representation was either 100 or 200. The results in Table 3 indicate that using Algorithm 1 we achieve impressive performance with both models in comparison with the other schemes. The log-linear model works just as well as the non-linear model and is much quicker to train. One should note that the results from Microsoft Word and Aspell overestimate the fidelity of normalization since the task was performed manually, i.e., we picked the best option from the suggestion list. In case of no correct suggestion, we left the original form as is. Hence, the results are skewed towards achieving high precision. In contrast, our approach is completely unsupervised in design and evaluation. We also compared our approach with a Twitter and SMS dictionary compiled from several websites. The dictionary contained entries for 3864 words and 3536 phrases. The dictionary was compiled into a FST and the procedure in Section 6.2 was used for evaluation. Since, the dictionary entries do not have an associated score, the FST lexicon **N** is unweighted. Our results clearly indicate that for construction of the normalization lexicon all we need is a reliable distributed representation trained on large amount of noisy text. The non-linearity with the neural network does not help significantly for this task.

Approach	Precision (%)	Recall (%)	F1 (%)
(Han et al., 2012b)	70.0	17.9	26.3
(Hassan and Menezes, 2013)	85.3	56.4	69.9
Our approach	64.8	76.3	70.1

Table 4: Sentence level normalization on Twitter test set from (Han et al., 2012b)

We also tested our approach on a publicly available Twitter test set (Han et al., 2012b) comprising of 548 sentences to compare our framework with other state-of-the-art approaches. The training data and approach for each of these schemes is different and we did not optimize our model in any way on the test domain or data. The normalization was performed at the sentence level and we used the language model described in Section 6.2. The results in Table 4 demonstrates that our framework performs favorably in comparison with other techniques.

Category	Model	Precision (%)	Recall (%)	F1 (%)
Phrase	Microsoft Word (Oracle)	99.2	18.7	31.5
	Aspell (Oracle)	75.0	0.4	0.8
	Web dictionary with LM	34.0	19.0	24.4
	Neural Network ($wlen$:11+D:100) without LM	91.4	60.7	73.0
	Neural Network ($wlen$:11+D:100) with LM	92.4	71.3	80.5
	Neural Network ($wlen$:11+D:200) lexicon with LM	92.5	71.8	80.8
	Neural Network ($wlen$:15+D:100) lexicon with LM	92.4	71.4	80.6
	Neural Network ($wlen$:15+D:200) lexicon with LM	92.5	71.8	80.8
	Log-Linear Model ($wlen$:11+D:100) lexicon with LM	92.6	72.0	81.0
	Log-Linear Model ($wlen$:11+D:200) lexicon with LM	92.3	71.1	80.3
	Log-Linear Model ($wlen$:15+D:100) lexicon with LM	92.6	71.5	80.6
	Log-Linear Model ($wlen$:15+D:200) lexicon with LM	92.0	70.9	80.0

Table 5: Sentence level phrase normalization on English Twitter data

7 Learning Phrase Normalizations

A major drawback of inducing normalization lexicons using most approaches described in Section 2 is that they are restricted to learning one-to-one word mappings. However, social media text is strewn with abbreviations that span multiple words, e.g., *ily2* refers to *i love you too*. With our framework, one can obtain 1-to-many (or vice versa) mappings if the training data is modified to contain compound words, i.e., *i love you too* is replaced with *i_love_you_too* and treated as a single token. The biggest obstacle is to get a reliable list of such phrases since they keep changing and growing. Unsupervised phrase induction using likelihood ratio test, point-wise mutual information, etc., may be used for such a task but they typically do not capture phrases formed from high frequency function words.

We used a dataset of speech-based SMS message transcriptions for compiling a list of common phrases. The SMS messages were collected through a smartphone application and a majority of them were collected while the users used the application in their cars. We had access to a total of 41.3 million English messages. The speech transcripts were mostly automatic and only a subset of around 400K utterances were manually transcribed. To avoid the use of erroneous transcripts, we sorted the messages by frequency and picked phrases between length 2 and 4 that resulted in 27356 English phrases. The training data was then phrasified (words were compounded) with the above phrase lists and the experiments to learn distributed representations was repeated. We performed this experiment only on Twitter data.

Once the representations were learned, we computed the K-nearest neighbors using the cosine similarity metric for each phrase. The lexical similarity cost was computed differently for the phrases. The first character of each word in the phrase was picked to form a new string (e.g., *i_love_you_too* would be converted into *ilyt*) and a similar technique was used on the nearest neighbors; singleton numbers were expanded into strings (e.g., *ily2* would be converted into *ilyt*). The lexical similarity metric in Equation (3) was then used to compute the distance between the two strings. The normalization table was subsequently inverted and compiled into a FST. Table 6 shows some of the phrase normalizations learnt by our framework. The phrase normalization results for English are presented in Table 5. Our framework learns phrase normalizations quite well. We achieve precision and recall of 92.6% and 72.0%, respectively. Even the mistakes committed are not very different from the ground truth, e.g., *idk* → *i_do_not_know* while the reference is *i_don't_know*, *wtf* → *what_the_hell* instead of *what_the_f**** and *omfg* → *oh_my_god* in place of *oh_my_f****_god*. Our framework does not capture expansions of geographic locations present in the reference data

Tokens	Canonical phrase
tyvm, tysm, ty, thxs	thank you very much
idk, idfk, irdk, idkk	i don't know
ihy, ihu,	i hate you
ily, ilym, ilyy, ilu	i love you
lmk, hum	let me know
omw, omww, otw	on my way
jgh, jghome	just got home
g2g, gottago	got to go
2b, 2ba	to be
2u, 2us	to you
4u, 4you	for you
cme, callme	call me

Table 6: Phrase normalizations learned through our framework

such as *nz* → *new_zealand*, *la* → *los_angeles* and some highly context dependent expansions such as *dw* → *doctor_who*, *dm* → *direct_message*, etc., since the compiled phrase list did not contain these entries.

8 Discussion

The normalization lexicons learned in this work are completely unsupervised. The quality and coverage of the lexicon is dependent on the size and distribution of words in the training data. Our approach assumes that the data contains both the noisy and canonical form of a word. In practice, we have observed that for large text collections, such as millions of tweets, words appear in both canonical and noisy forms. One can also augment noisy corpora with clean text to improve the distributional similarity of the two forms. Choosing the optimal size of noisy data set and appropriate augmentation is beyond the scope of this work.

The main parameters in our model are $wlen$, D and K. The choice of K is dependent on the size of the training data, i.e., a larger K can potentially yield more noisy to canonical mappings. For our datasets, the choice of $K = \{25, 50, 100\}$ did not result in any significant difference in performance. Hence, we used $K = 25$ to increase the speed of Algorithm 1. We also performed several experiments to understand the choice of dimension D. In general, the choice of D is dependent on the vocabulary size of the training data. For vocabulary size between 100K-500K, we found that vector dimension of $D = 50$ is sufficient and for vocabulary size greater than $1M$, $D = 100$ works well empirically. For $wlen$, a context of 5 words to the left and right, i.e., $wlen = 11$ works well and adding more context does not necessarily improve performance. We conjecture that this is due to the length of an average customer care note (12 words) and tweet (15 words). For datasets with longer sentences, larger values of $wlen$ may be beneficial.

The performance reported using Microsoft Word and Aspell was obtained by manually selecting the best suggestion. We resorted to this approach since both schemes do not provide an option to automatically normalize a document. In choosing the best suggestion option, we focused on precision, i.e., picked the best suggestion or left the original form as is. If we had forcefully picked an option from the suggestion list for all corrections, the recall would have been higher at the cost of lower precision. The results using the Web dictionary and our unsupervised framework performs a blind evaluation.

In contrast with conventional string similarity based normalization schemes, our approach is good at modeling abbreviations. Abbreviations are generally hard to normalize with Levenshtein distance based approaches but the combination of distributional and lexical similarity is very helpful in learning the mapping between abbreviated and canonical forms. In most off-the-shelf systems, e.g., Microsoft Word, a standard dictionary is used and any corrections for domain specific spellings are typically performed manually. Since our scheme can be trained with raw data, we are able to address the domain specific idiosyncrasies.

The word and phrase normalizations learned in this work use a particular type of lexical similarity metric. While it captures abbreviations well for English, our framework is open to the use of any linguistically motivated lexical similarity metric. Such metrics can be designed by language experts and linguistic knowledge can potentially be incorporated into the unsupervised scheme, thus, lending a way to embed linguistic rules into a statistical framework.

9 Conclusion

We presented an unsupervised framework for normalizing domain-specific and informal noisy texts using distributed representation of words. Our approach exploits the property that noisy and canonical forms of a particular word share similar context in a large noisy text collection (millions or billions of social media feeds from Twitter, Facebook, etc.). Subsequently, we use a combination of distributional and lexical similarity between canonical and noisy form of words to automatically construct a normalization lexicon. The distributed representations were learned using both log-linear and non-linear models and we used a finite-state transducer framework for representing the lexicons and performing normalization. Our experiments on customer care data and Twitter indicate that our approach can capture spelling errors of different types and achieves good performance in comparison with several baselines and state-of-the-art approaches. Finally, we used our framework to learn phrase normalizations by learning distributed representations over compound words.

References

A. Aw, M. Zhang, J. Xiao, and J. Su. 2009. A phrase-based statistical model for SMS text normalization. In *Proceedings of COLING*, pages 33–40.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. 2009. Curriculum learning. In *Proceedings of ICML*.

E. Brill and R. C. Moore. 2000. An improved error model for noisy channel spelling correction. In *Proceedings of ACL*, pages 286–293.

R. Collobert and J. Weston. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of ICML*.

R. Collobert, K. Kavukcuoglu, and C. Farabet. 2011. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*.

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

J. L. Elman. 1991. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7(2-3):195–225.

B. Han, P. Cook, and Baldwin. 2012a. Automatically constructing a normalization dictionary for microblogs. In *Proceedings of EMNLP*, pages 421–432.

B. Han, P. Cook, and T. Baldwin. 2012b. Automatically constructing a normalisation dictionary for microblogs. In *EMNLP-CoNLL 2012*, pages 421–432.

H. Hassan and A. Menezes. 2013. Social text normalization using contextual graph random walks. In *Proceedings of ACL*, pages 1577–1586.

V. I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady (in English)*, 10(8):707710, February.

D. Melamed. 1995. Automatic evaluation and uniform filter cascades for inducing n-best translation lexicons. In *Proceedings of the 3rd ACL Workshop on Very Large Corpora (WVLC)*.

T. Mikolov, S. Kopecký, L. Burget, J. Černocký, and S. Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of Interspeech*.

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

F. J. Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*.

L. Philips. 1990. Hanging on the metaphone. *Computer Language*, 7(12):39–43.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362.

K. Toutanova and R. C. Moore. 2002. Pronunciation modeling for improved spelling correction. In *Proceedings of ACL*, pages 141–151.

J. Turian, L. Ratinov, and Y. Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of ACL*.