

# Experiments with artificially generated noise for cleansing noisy text

Phani Gadde, Rahul Goutam, Rakshit Shah, Hemanth Sagar  
Language Technologies Research Centre, IIT-Hyderabad, India  
{phani.gadde,rahul.goutam,rakshit.shah,hemanth.sagar}@research.iit.ac.in

L. V. Subramaniam  
IBM Research, New Delhi, India  
lvsubram@in.ibm.com

## ABSTRACT

Recent works show that the problem of noisy text normalization can be treated as a machine translation (MT) problem with convincing results. There have been supervised MT approaches which use noisy-regular parallel data for training an MT model, as well as unsupervised models which learn the translation probabilities in alternative ways and try to mimic the MT-based approach. While the supervised approaches suffer from data annotation and domain adaptation difficulties, the unsupervised models lack a holistic approach catering to all types of noise. In this paper, we propose an algorithm to artificially generate noisy text in a controlled way, from any regular English text. We see this approach as an alternative to the unsupervised approaches while getting the advantages of a parallel corpus based MT approach. We generate parallel noisy text from two widely used regular English datasets and test the MT-based approach for text normalization. Semi-supervised approaches were also tried to explore different ways of improving the parallel corpus (manually annotated) based MT approach by using the generated noisy text. An extensive analysis based on comparison of our approaches with both the supervised as well as unsupervised approaches is presented.

## Keywords

cleansing noisy text, machine translation, artificial noise, noisy text analytics, NLP, generated noise

## 1. INTRODUCTION AND RELATED WORK

Noise can be defined as any kind of difference in the surface form of an electronic text from the original, intended or actual text [18]. The distortion of words in short message service (SMS) and on-line forums like twitter, chat,

discussion boards and social networking sites is essentially because the recipient can understand the message even if longer words are represented in short forms, thereby reducing the time and effort of the sender. With the amount of text increasing exponentially on the web, there has been a recent surge in developing applications such as opinion mining [6], on-line information diffusion [5] and text driven forecasting [3] on texts which need to work on noisy texts. Cleansing the noisy text would really help, as the off-the-shelf NLP techniques generally fail [15, 13] to work because of several reasons like sparsity, out-of-vocabulary words and irregular syntactic structures in such texts.

[4] studied SMS normalization as a machine translation problem. They used an SMS-regular parallel data of size 4000 sentences to train a phrase based machine translation model from the SMS to the regular side and achieved convincing results. [25] gives a detailed account of the MT-based approach with performance tuning and results on different datasets and [17] extends this work to twitter. [19] uses a combination of the MT-based approach with an ASR-based approach to achieve better results for SMS normalization in French.

Apart from the difficulty in annotating the parallel data, the MT-based approaches suffer from domain difference, as noisy text in different forms of computer-mediated communication like discussion forums, internet chats, SMSes and social networking sites vary in their noise level and orthography. To overcome these issues, [9, 11] follow a noisy channel model to estimate the target regular sentence given the source noisy sentence. [10] use unsupervised cluster membership confidence between a noisy word and a regular word as the translation probability thereby imitating the supervised MT-based approach. However, all these unsupervised approaches perform less at the phrase level (like for *Word Merging* introduced in section 2) than at word level due to the lack of a holistic context dependence as in an MT-based approach.

In this paper, we try to achieve the advantage of both the approaches by artificially generating noisy text from regular text which serves as the parallel data needed for the MT-based approach. We draw our motivation from [8, 12, 14] where automatically created grammatical mistakes were used to make the existing NLP techniques robust towards noise. [2] generate spelling errors resulting from typing mistakes in contrast to the real word spelling errors that are generated in [12]. This way, we not only overcome the data an-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

J-MOCR-AND '11 Beijing, China

Copyright 2011 ACM 978-1-4503-0685-0/11/09 ...\$10.00.

notation difficulty, but also try to generate noisy text which matches a given genre, to adapt to various forms of computer mediated communication. We explore different approaches to use the artificially generated noise as an aid to the manually annotated parallel corpus in the MT based approaches across different domains, thereby comparing our approach with the unsupervised techniques as well. We show that accuracies comparable to other unsupervised approaches can be achieved with the proposed approaches inspite of a lot of complexity in the wake of domain and noise-percentage differences between the generated and real world noisy texts.

The rest of the paper is organized as follows: section 2 gives a survey of different types of noise in computer mediated communication. Section 3 presents our noise generation algorithm and section 4 describes the MT experiments to normalize noisy text. In section 5, we give a detailed error analysis of the task and conclude in section 6.

## 2. NOISE TYPES

Text in Computer Mediated Communication (CMC) is distorted in a variety of ways. Words are shortened in order to reduce the time and effort of typing. However, the sender ensures the correct understanding of the text on the receiver’s side while shortening the words, which makes us believe that there are specific word formation processes that are being followed. Assuming all of the different CMC texts exhibit similar types of noise in varying percentages, we attempt to classify the noise in text into different types. We intend to exploit this fact, so that we have definite ways of generating each type of noise possibly imitating their natural generation heuristics.

There are a lot of studies about different types of noise in CMC data. The orthographic and syntactic structures used in the CMC data have been studied not only for English, but also for several other languages, such as Arabic, German, Japanese and Swedish ([9] and the references therein). We studied the publicly available NUS corpus [16] to identify the noisy word generation heuristics predominant in SMS texts. We later observed the datasets used in the earlier works on text normalization [4, 9, 27, 11] to accommodate the types of noise described in those works. We arrived at a list of 10 types of noise which includes a) Capitalization and decapitalization, b) Phonetic substitutions, c) Character deletion, d) Typing mistakes, e) Code switching, f) Dialectal usages and slangs, g) Abbreviations, h) Merging words, i) Dropping of words (functional, punctuations) and j) Emoticons.

This paper does not deal with 4 types of noise namely a) Code switching, b) Dialectal usage, c) Abbreviations and d) Emoticons. While code switching requires the application of other NLP techniques like transliteration to generate, we assume that abbreviations are definite and can be dealt with a dictionary look-up (pre-processing) before any NLP task. We notice that generating dialectal usage (*think* → *fink* and *the* → *e*) is genre and location dependent and Emoticons provide additional information which is not in the text and hence should not be removed (might be converted) while cleansing noisy text. The remaining 6 types of noise are being generated in this paper. They can be seen with examples in Table 1.

Note that *G-clipping*, *H-clipping*, *prefix and suffix clippings* described in [11] come under *character deletion*. Similarly, with reference to [4], *dropping verb* and *dropping ?* occur under *word dropping* and *dropping vowel* comes under

**Table 1: Table showing different types of noise.**

Noise Type	Examples
Decapitalization/ Capitalization	<i>Paris</i> → <i>paris</i> <i>beep</i> → <i>BEEP</i>
Phonetic substitution	<i>then</i> → <i>den</i> <i>for</i> → <i>4</i>
Character deletion	<i>introduce</i> → <i>intro</i> <i>coming</i> → <i>cmin</i>
Typing errors	<i>dear</i> → <i>desr</i> <i>like</i> → <i>lik2</i>
Word dropping	<i>are you coming</i> → <i>u coming</i>
Word merging	<i>I am</i> → <i>im</i> <i>at least</i> → <i>atleast</i>

*character deletion*.

## 3. GENERATING NOISY TEXT

This section describes the procedure for automatically creating a parallel corpus of noisy text given a regular English text. As described in the previous section, it is ensured that the generated text contains six types of noise described in Table 1. As said earlier, the automatic noisy text generation procedure is inspired by the artificial noise created in the works [14, 12, 2], where using the properties of a certain kind of noise, noisy words are created given a regular word. However, in contrast to the algorithms used in those works, we aim to add multiple kinds of noise to a single word or a phrase. This decision is in line with our observations in cases like *im*, where *I* and *am* are merged and *a* is deleted. At the same time, the noisy text generation procedure can also be applied to its own output to yield words with more complex noise as done in the previous approaches.

**Algorithm 1** Overall algorithm

---

```

wordIndex ← 0
for word in input do
    inputWords[wordIndex] ← word
    mapping[wordIndex] ← word
    wordIndex++
end for
totalWords ← wordIndex
noiseLevels ← (0.5, 0.05, 0.42, 0.42, 0.1, 0.01, 0.50)
pNoisy ← noiseLevels[0]
noisyTotal ← pNoisy * totalWords
isNoisy[] ← false
mergeNoisy ← mergeWords()
phoneticNoisy ← phoneticWords()
charDelNoisy ← charDelWords()
dropWordsNoisy ← dropWords()
typoNoisy ← typoWords()
capDecapNoisy ← capDecapWords()

```

---

The overall algorithm is described in algorithm 1. The percentage of each type of noise and the overall noise is set in the *noiseLevels* variable, to be able to control the overall and individual noise levels in the generated text. The default values (in Algo. 1) have been set to reflect the approximate observed distribution of noise in the NUS [16] and TMT [9] datasets (more about this in the experiments section).

Note that each type of noise is generated by calling the corresponding sub-routines (described in the further sections). All the sub-routines work on the entire corpus instead of working on each sentence (or word as in [12]). However, the alignments between the input words (in each sentence) and the corresponding output noisy words is being stored in the *mapping* data structure and being updated in each of the sub-routines to arrive at gold standard alignments to train a phrase based translation model. The variables and data structures shown in Algo. 1 are made global in the actual code, to enable the sub-routines access the data as well as add multiple types of noise to a single word. To handle the inter-dependency between them, the sub-routines are called in a particular order (different from the order of their description in the further section). For example, since merged words can still be phonetically substituted or character deleted, *mergeWords()* is being called before *phoneticWords()* and *charDelWords()*.

### 3.1 Phonetic substitution

Given a word in regular English, phonetic substitution module generates words that sound similar to the given word. In the context of SMS texts, phonetically substituted words denote nascent specific dialectal usages. For example, *muzik* or *myuzik* is used for *music*, *skool* for *school* in noisy texts. This process is a very creative one (English being non-phonetic language) and cannot be dictated by a small set of rules. We view the task of generating all possible phonetically substituted words (for a given English word) as an SMT problem. Our ultimate goal is to learn a model that can automatically generate possible phonetically substituted words (by humans) for a given regular English word.

Since a parallel corpus of regular English words and their phonetically substituted counterparts is not available, we use phoneme sequences as an intermediate language. This essentially means we learn two models. The first model (forward model) tries to capture the correspondences between regular English words and phoneme sequences and the second model (reverse model) captures the correspondences in the reverse direction. The corpus required for the individual models is provided by the publicly available CMU phonetic dictionary [28]. It contains 125,000 English words and their corresponding North American phonetic transcriptions. The character sequence of the English word in the CMU dictionary and the corresponding phoneme sequence form a sentence pair for SMT task. In essence, each character in a word acts like a word in the source and each phoneme is considered as a word on the target side. Such pairs are collected over the entire CMU dictionary to form the parallel corpus required to build the forward and reverse models.

We use the publicly available Moses tool kit [20] to estimate translation probabilities from regular English character sequences to phoneme sequences in the forward model and from phoneme sequences to character sequences in the reverse model. The phrase tables (translation probabilities between English and phoneme character sequences) thus obtained are combined to get an English to English character sequence, keeping the phoneme sequences as the anchor. We believe that this combined phrase table captures the phonetic substitution process effectively. The beam search decoder (part of Moses toolkit) is used to generate phonetically substituted words for all words in the input corpus. Other configuration details of Moses include setting the dis-

Figure 1: Example showing Phonetic Substitution

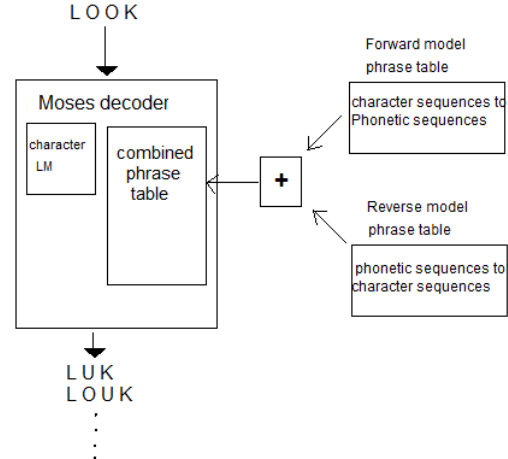


Table 2: Phonemic Rules with Examples. # means end of the word. C: consonant, V: vowel

Phonological Rules	Condition	Example
$VC_1C_2 \rightarrow VC_1$	$C_1 = C_2$	error $\rightarrow$ eror
$C_1VC_2 \rightarrow C_1C_2$		ring $\rightarrow$ rng
$V_1V_2C \rightarrow C$		please $\rightarrow$ plse
$CV\# \rightarrow C$	$V = 'e'$	move $\rightarrow$ mov
$V_1V_2 \rightarrow V_1$	$V_1 = V_2$	coffee $\rightarrow$ coffe
$V_1V_2 \rightarrow V_1$	$V_2 = 'e'$	variety $\rightarrow$ varity

tortion limit to zero (as phonetic substitution is a monotonic translation) and setting the word penalty to positive numbers to see that the output word is shorter in length than the original word. The entire procedure is shown in Figure 1.

### 3.2 Character deletion

Deletion of characters is one of the predominant techniques for compression of words in noisy texts. The commonly observed patterns include deletion of vowels ('msg' for 'message'), deletion of repeated character ('tomorrow' for 'tomorrow') and truncation ('tom' for 'tomorrow') [9, 27, 4]. In [21], the authors have shown that phonology affects the pattern of deletions in text messages and the deletion of characters is governed by phonological principles. They have also reported the ratio of 9:1 for dropping of vowels to dropping of consonants in character deletion. Based on these observations as well as our study of the noisy texts, we have come up with certain rules which can be applied to a regular word to compress it while at the same time, ensuring that the phonemic representation of the regular word and the compressed word are close.

The algorithm for Character Deletion is described in algorithm 2. All special characters like '-', ',', etc. are dropped from the word (for example, *black-board* goes to *blackboard*) as a preprocessing step. The phonological rules used are presented in Table 2.

Since more than one rule can be applied on a word in a cascaded fashion, we applied all the possible permutations of the rules on a given word. After applying each rule, the word formed is added to the set of possible character-deleted words. Finally, a word is chosen randomly from the set of

---

**Algorithm 2** Character Deletion

---

```
pCharDel ← noiseLevels[3]
nCharDel ← pCharDel * noisyTotal
nMergeCharDel ← (pCharDel/8) * mergeNoisy.size()
exclusiveCharDel ← randomly select nCharDel non-
noisy words from inputWords
mergeCharDel ← randomly select nMergeCharDel
words from mergeNoisy
charDel ← exclusiveCharDel ∪ mergeCharDel

for word in charDel do
  noOfRules ← 6
  p ← permutations(noOfRules)
  noisyChDelSet ← ε
  for permutation q in p do
    for rule in q do
      noisyWord ← apply rule on word
      noisyChDelSet ← noisyChDelSet' ∪ noisyWord
    end for
  end for
  update mapping
end for
return charDel
```

---

possible character-deleted words corresponding to the regular word.

Truncation of words (*introduction* → *intro*) was generated using a statistical approach in which word prefixes (length  $\geq 3$ ) are generated and ranked according to their frequency in the NUS corpus [16]. We tried a statistical way of generating the overall character deletion too, by exhaustively generating the  $2^n$  possible words given a word of length  $n$  characters and ranking them using a character based language model trained on the NUS corpus. The language model somehow could not capture the phonemic constraint generating irrelevant words which motivated us towards a rule based approach.

### 3.3 Word merging

Word merging can be described as the deletion of space between two words to form a single word, like typing *thankyou* for *thank you* and *newyork* for *New York*. It can also be followed by *character deletion* and *decapitalization* like in the case of typing *im* for *I am*. This is one of the types of noise (*word dropping* being the other) which not only makes a word noisy but also effects the structure of the sentence and hence would not be handled well with unsupervised techniques [9, 11, 10] because of the assumption of context independence in noise generation. Manual inspection of noisy text showed that intentional deletion of space takes place only under two specific situations: (1) for commonly used pairs of words (*thank you*) and (2) for certain grammatical categories like the auxiliaries (*what is* → *whats*) and negative marker (*would not* → *wudnt*).

In order to generate this type of noise, we assume that the most frequent bi-grams are the ones that are being merged, with the intuition that one can make out the meaning of the phrase even if the space is not present. In other words, if the same two words have to be typed together frequently, the words will most likely be merged into a single word (like *thankyou*). We take the most frequent bi-grams in the BNC sampler corpus [1] of size 2 million words and rank them in

---

**Algorithm 3** Merging

---

```
mergedWordList ← ε
K ← 500
pMerge ← noiseLevels[1]
bigramList ← positions of Top-K bigrams in inputWords
nMerge ← pMerge * noisyTotal
rList ← randomly select nMerge positions from
bigramList

for position of each bigram in rList do
  word_1 ← inputWords[position]
  word_2 ← inputWords[position+1]
  if word_1 and word_2 not noisy then
    mergedWord ← word_1 + word_2
    update mapping
    mergedWordList ← position
  end if
end for
return mergedWordList
```

---

descending order based on their bi-gram frequency. Only the top- $K$  bi-grams are considered for merging, the default value of  $K$  being 500. *Character dropping* and *decapitalization* on merged words are done in their respective sub-routines. The algorithm is described in algorithm 3. The set *bigramList* contains the indices of all the occurrences of top- $K$  bi-grams in the input. *nMerge* indices are randomly selected from this set for merging. Currently, we do not handle merging of more than two words.

### 3.4 Typing errors

Unlike in the regular text, typing errors are generally neglected in noisy text. Hence, we consider correcting typing errors as a part of cleansing noisy text. [2] simulate typing errors by assigning weights to different types of typing errors like *insertion*, *deletion*, *substitution*, *transposition* and *duplication*. However, in our analysis of noisy texts, we found that *substitution* is the most frequent kind of typing error while *transposition* and *duplication* exist in negligible counts. With deletion already handled in character deletion type of noise, in this work, we only generate typing errors by substitution of characters. The algorithm for Typing Errors is described in algorithm 4. The variable *typoMap* is a map between each character and the corresponding set of characters which can possibly replace that character, constructed from the the characters that are close to the key in a QWERTY keyboard.

Note that a typing error can either occur exclusively or overlap with other types of noises such as *merging*, *character deletion* and *phonetic substitution*. The set *typoList*, hence contains randomly selected words from the non-noisy words in the input as well as words which are already made noisy from one of the above types of noise. While *exclusiveTypo* contains the list of words which are chosen from non-noisy words, *nonExclusiveTypo* contains randomly selected words from the already merged, character deleted and phonetically substituted words. For each selected word, a single character in the word is randomly selected and substituted by a randomly selected character from its *typoMap*.

---

**Algorithm 4** Typing Errors

---

```
ptypo ← noisyLevels[5]
ntypo ← ptypo * noisyTotal
typoMap ← a map of characters to possible replacement
           characters
exclusiveTypo ← alphabetic non-noisy words in
                 inputWords
nonExclusiveTypo ← mergeNoisy ∪ phoneticNoisy ∪
                   charDelNoisy
exclusiveTypoList ← randomly select nTypo words from
                    exclusiveTypo
nonExclusiveTypoList ← randomly select nTypo words
                       from nonExclusiveTypo
typoList ← exclusiveTypoList ∪ nonExclusiveTypoList

for word in typoList do
  char ← randomly select a character from word
  replacementChar ← typoMap(char)
  noisyWord ← replace char by replacementChar in
              word
  update mapping
end for
return typoList
```

---

### 3.5 Word dropping

While communicating over texting media there is a tendency to drop predictable words in text. Function words are predictable as opposed to the lexical ones since function words do not carry the main meaning like lexical words do [21]. We observed that one of the predominant techniques of compressing text is to drop function words from the text. Apart from function words, punctuations like comma and dot and words showing denominations and numbers like dollar and hash constitute the common word droppings. We consider both these types while generating the noisy text.

---

**Algorithm 5** Word Dropping

---

```
wordList ← funcWords ∪ {comma(,), dot(.), #, $,
                       }, (:, :, “,”)}
wordPositions ← positions of words from wordList
                 in inputWords
pDropWord ← noiseLevels[4]
nDropWord ← pDropWord * noisyTotal
dropWords ← randomly select nDropWord indices from
             wordPositions

for position of word to be dropped in dropWords do
  mapping[position] ← NULL
  isNoisy[position] ← true
end for
return dropWords
```

---

The overall algorithm is described in algorithm 5. Function words (*funcWords*) are extracted from the WSJ corpus [22] based on the part of speech tags of the words. Randomly selected function words and punctuations from the input (*dropWords*) are dropped and the mapping between the regular and noisy sides is updated.

### 3.6 Decapitalization/Capitalization

Decapitalizing proper nouns is one of the most prevalent types of noise across different genres of noisy text like SMS, chats, twitter, social networking posts and discussion forums. Also, using all capital letters for specific words to increase their focus is a common practice. We introduce both the types of noise into the given regular text. Predefined percentages of words are randomly selected from the input for both *capitalization* and *decapitalization* and correspondingly converted to the opposite case. This way, we aim to make the translation model learn to convert the case of the words in the given noisy text accordingly.

## 4. EXPERIMENTS WITH GENERATED NOISY TEXT

We use the procedure described in the previous section on WSJ [22] and BNC sampler (BNC) [1] regular English corpora to generate the noisy-regular parallel datasets. While WSJ contains 1 million words of only written English (newswire), BNC contains spoken English of 1 million words along with written English of 1 million words, collected from a variety of domains.

The individual noise percentages have been set to approximate the observed distribution of noise in the NUS [16], TMT [9] datasets as well as close to the distribution presented by [10]. These are shown in algorithm 1. To make the generated noisy corpus representative of varying noise levels found in different noisy texts, we vary the overall noise (*pNoisy* in Algo. 1) in the algorithm from 10% to 90% (100% noisy text is not possible to create as we don't make cardinals noisy) with intervals of 10% and combine all the 9 sets to arrive at the noisy-regular parallel dataset. Figure 2 shows the algorithm's outputs (*OUT*) on sentences from WSJ and BNC (*INP*). We also show (in Fig. 2) the output of our noise generation procedure on the manually annotated English sentences (*REF*) for the real world NUS SMSes (*REAL*) by [4], to see how the generated noise compares with the real world SMSes.

**Figure 2: Outputs of the noise generation algo.**

BNC  
INP: *For Mr Prokhrorov , there is another side to things .*  
OUT: *fr Mr Prokav , thereis nor side to Thngs .*  
  
INP: *Another was seen later with a breast wound .*  
OUT: *Another was sn Ltr witha brst wnd .*  
  
WSJ  
INP: *Excluding these orders , backlogs declined 0.3 % .*  
OUT: *Excludng thes ord , bcklgs dclend 0.3 % .*  
  
INP: *Grace holds three of Grace Energy 's seven board seats .*  
OUT: *Grace hlds thre Grace Energy 's sven brd seats .*  
  
Aw et. al. NUS SMS  
INP: *Dila , how fair are you ? Bad question ?*  
OUT: *dla , hw fr r you ? Bad qstn ?*  
REAL: *Dila hw fair r u ? Bd ques ?*  
  
INP: *Thanks but my birthday is over already .*  
OUT: *thnks but my birday is ovr ardi .*  
REAL: *Thanx but my birthday is over already .*

We conduct two sets of experiments using the generated noisy-regular parallel data. The first set of experiments are completely unsupervised, where we aim to see how the generated parallel data performs on the real world noisy text. The second set of experiments are semi-supervised, aimed at investigating if the generated parallel data acts as an aid for a small seed parallel corpus of 4000 sms-regular corpus (**AwTrain**) which was used by [4]. The unsupervised and semi-supervised experiments are presented in sections 4.1 and 4.2 respectively.

Both the unsupervised and semi-supervised approaches are tested on three test sets: a) **AwTest**: A set of 1000 SMSes of the NUS corpus used by [4] (Their entire dataset - **AwTrain**), b) **TMT**: The dataset used by [9], which consists 427 SMSes of North American users, c) **MixSMS** A set of 711 SMSes used by [10] which is a mixture of SMSes from various domains.

#### 4.1 Unsupervised MT experiments

We use the synthetic data generated by our approach (from both WSJ and BNC) to build a phrase based MT system in a completely unsupervised way (or *simulated supervised*). We use Moses [20], a state-of-the-art, open source MT toolkit to build the translation model between the noisy and regular sides. Since we are storing the mapping between the regular and noisy sides while generating noise, we use gold word alignments instead of approximated ones. The phrase table along with a trigram language model trained with Good Turing smoothing using SRILM toolkit [26] are used by the Moses decoder to generate regular English sentences given a noisy sentence.

The results of these experiments can be seen in the unsupervised section of table 3. Each row shows the result of an experiment with the details of the training data used while building the phrase table and the language model used while decoding. The first row shows the baseline bleu [24] scores between the noisy and regular sides of the test sets. Detailed analysis of the results are given in section 5.

#### 4.2 Semi-supervised MT experiments

The aim of these experiments is to see if the generated noisy-regular parallel data (from both WSJ and BNC) can be used to aid a translation model learnt on a seed corpus of manually annotated noisy-regular parallel text. As said earlier, we use **AwTrain** as our seed corpus. To do this, we add each of the generated datasets to **AwTrain** and build translation models using Moses as done in the previous section. The first two rows of the semi-supervised section of table 3 show the results of these experiments.

Since the noisy text obtained by running a regular-noisy translation model (on any regular text) built on just the seed corpus can also be assumed as a synthetic noise. To compare such noisy text with the noisy text being generated by our approach, we perform a second set of experiments, where the seed-model generated parallel data (from both WSJ and BNC) is added to the seed data and translation models were built. We use a set of 27000 SMSes from the NUS corpus as a language model while self-generating the noisy corpora. The final translation models are built using Moses, as described in the above section. Third and fourth rows in the semi-supervised section of table 3 show the results of these experiments.

Table 3: Bleu scores for various experiments

Unsupervised				
Training Set	LM	AwTest	TMT	MixSMS
<b>Baseline</b>		<b>0.448</b>	<b>0.396</b>	<b>0.410</b>
2*WSJ	AwTrain	0.519	0.444	0.469
	BNC	0.470	0.407	0.421
2*BNC	AwTrain	0.531	0.482	0.503
	BNC	0.475	0.426	0.428
Semi-Supervised				
Training Set	LM	AwTest	TMT	MixSMS
AwTrain+	AwTrain	0.665	0.582	<b>0.561</b>
Gen WSJ	BNC	0.590	0.534	0.516
AwTrain+	AwTrain	0.649	0.523	0.527
Gen BNC	BNC	0.543	0.459	0.458
AwTrain+	AwTrain	0.703	0.591	0.535
Self WSJ	BNC	0.646	0.563	0.507
AwTrain+	AwTrain	0.700	0.577	0.521
Self BNC	BNC	0.641	0.521	0.486
Supervised				
Training Set	LM	AwTest	TMT	MixSMS
2*AwTrain	AwTrain	<b>0.710</b>	<b>0.599</b>	0.541
	BNC	<b>0.662</b>	<b>0.596</b>	<b>0.544</b>

#### 4.3 Discussion

The BNC language model has been tried while decoding in all the experiments, as the target language is regular English. However, using AwTrain language model performed well as it worked as an in-domain language model *EP* with reference to [7] in a work of domain adaptation for SMT. Though the performance of an MT system can fluctuate a lot with small changes in the decoding parameters, no other optimization was done in any of the experiments because of the lack of in-domain development datasets. We did try using Mert [23] to optimize the unsupervised approaches on the synthetic development data, but observed over-fitting and a prominent decrease in performance on the real world noisy texts.

The best bleu scores on each dataset are made bold in table 3. The results show that it is very hard to beat the supervised approach using our unsupervised or semi-supervised approaches, though we are getting comparable results with other unsupervised approaches (0.539 by [10] on the MixSMS test set). However, the semi-supervised approach by adding synthetic parallel data created from WSJ to the seed data gave a slightly better performance on the MixSMS test set.

### 5. ERROR ANALYSIS

We did a detailed error analysis to know why the difference between the performance of the supervised approach and the unsupervised, semi-supervised approaches is very significant. Our analysis has shown that both the unsupervised and the supervised approaches do perform well most of the time. However, the major difference with the supervised technique comes from the out-of-vocabulary words (OOVs). The unsupervised and semi-supervised translation models do not have any cue to ascertain whether to translate a word in the input or to leave it unchanged. This can

be seen in example (3) in figure 3, where a NUS corpus specific word *lor* is being translated to *later* instead of getting dropped. This kind of dropping is very common in the AwTrain corpus, which makes it do it correctly. The conversion of *hi* to *He* in example (1) is also a similar case.

### Figure 3: Example translations by unsupervised experiments

1. SMS: *hi pat ... Care to intro pls .*  
REF: *hi Pat ... Care to introduce please ?*  
OUT: *He put ... Care to introduce please .*
2. SMS: *Who is your favorite lead singer ?*  
REF: *Who is your favorite lead singer ?*  
OUT: *Who is your favorite leaders singer ?*
3. SMS: *Taking as , not os . Taking a break now , so chat lor .*  
REF: *Taking As , not Os . Taking a break now , so chat .*  
OUT: *taking as , not as . taking a break now , so chat later*

BNC, being a corpus having both written as well as spoken text collected across a variety of domains was expected to outperform the WSJ based experiments all the time, if OOVs is the only problem in the unsupervised and supervised approaches. But, the semi-supervised section of table 3 show BNC based experiments consistently performing lower than the WSJ based experiments. Detailed analysis of the generated corpora has shown that over generation in each of the sub-routines in the noise generation algorithm is adding unnecessary ambiguity to the translation model along with increasing the coverage. This can be seen in table 4, where the match between the noisy-regular word pairs between AwTest and each of the training sets is given along with their corresponding unique noisy-regular word pairs. With increase in the corpus size and variety, (BNC > WSJ > AwTrain), there is an increase in the coverage of the translation model and at the same time, very large number of noisy-regular words pairs are being generated which would eventually increase the ambiguity in translation. This can be seen in example (2) from figure 3 where *lead* got translated to *leader* instead of being left unchanged. This fact is supported by the experiments in table 5 where by increasing the variety of noise, there is an increase in the matched noisy-regular word pairs, but a decrease in the percentage of all the unique word pairs being matched, which is eventually reducing the Bleu score.

A major chunk of errors in all the approaches that use the artificially generated noise is also due to the difference in the language models in the training and testing. The test sets contain multiple small sentences in a single SMS while the WSJ and BNC datasets contain larger sentences with a single sentence end marker. This difference makes the MT models neglect the new sentence beginnings in each SMS, making several capitalization/decapitalization errors.

## 6. CONCLUSION AND FUTURE WORK

We proposed an algorithm to generate the major types of noise across various computer mediated communication given a regular text. The algorithm is used to generate noisy-regular parallel data from different regular English

Table 4: Error Analysis. 2909 below the third column name is the number of unique noisy-regular word pairs in AwTest

2*AwTest	Unique pairs	Matched pairs (2909)
AwTrain	6988	1288
WSJTrain	485357	1461
BNCTrain	719715	1605

Table 5: Coverage-ambiguity trade-off. 345.wsj in the second row represent the combined parallel data of overall noise percentages 30,40,50 created from WSJ and so on

Datasets	BLEU	Unique pairs	Matched pairs	%Match
4.wsj	0.486	136626	1324	0.9690
345.wsj	0.484	251765	1418	0.5632
23456.wsj	0.476	328389	1436	0.4373
1-9.wsj	0.471	485357	1461	0.3011

corpora to generate translation models that can be used to correct a given noisy text. We did unsupervised and semi-supervised translation experiments on SMS texts from various domains. However, SMS is supposed to be the hardest of various types of noisy text like chats and tweets. We would like to test our system on the other types of noise too, which would be relatively easy for the system.

We observe that there exists a coverage-ambiguity trade-off when trying to use artificial noisy-regular parallel data for text cleansing. Developing automatic methods to select a balance between coverage and ambiguity, given a monolingual corpus on which the translation system has to be used would be one of the prominent extensions of this work. In this respect, we intend to prune the over generated noisy substitutions of regular words using a noisy-regular parallel development corpus, which can be compared to re-ranking methods in parsing, which, currently give the state-of-the-art syntactic parsing performance.

Another potential extension to this work is to learn the values of various parameters being used in the algorithm like the percentage of the overall and individual types of noise and the parameters in the individual noise creation models, given a monolingual corpus of a particular noisy text type. Such an extension of this approach would make it generic for all the types of data. For instance, a product reviews forum may contain very less amount of noise, but tolerating which, would definitely improve say, opinion mining performance on those product reviews.

## 7. ACKNOWLEDGMENTS

We thank Joseph Kaufmann and Danish Contractor for providing us various datasets used in this paper. We are also grateful to Sudheer Kolachina, Meher Vijay, Aswarth Abhilash and the three anonymous reviewers for their insightful comments and suggestions.

## 8. REFERENCES

- [1] The bnc sampler, xml version., 2005.
- [2] S. Agarwal, S. Godbole, D. Punjani, and S. Roy. How much noise is too much: A study in automatic text classification. In *Seventh IEEE International Conference on Data Mining, 2007*, pages 3–12. IEEE, 2007.
- [3] S. Asur and B. Huberman. Predicting the future with social media. In *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 492–499. IEEE, 2010.
- [4] A. Aw, M. Zhang, J. Xiao, and J. Su. A phrase-based statistical model for sms text normalization. In *COLING-ACL, 2006*, pages 33–40. Association for Computational Linguistics, 2006.
- [5] E. Bakshy, J. Hofman, W. Mason, and D. Watts. Everyone’s an influencer: Quantifying influence on twitter. In *Proceedings of WSDM 2011*, pages 65–74. ACM, 2011.
- [6] L. Barbosa and J. Feng. Robust sentiment detection on twitter from biased and noisy data. In *Proceedings of COLING 2010: Posters*, pages 36–44. Association for Computational Linguistics, 2010.
- [7] N. Bertoldi and M. Federico. Domain adaptation for statistical machine translation with monolingual resources. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, pages 182–189. Citeseer, 2009.
- [8] C. Brockett, W. Dolan, and M. Gamon. Correcting esl errors using phrasal smt techniques. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 249–256. Association for Computational Linguistics, 2006.
- [9] M. Choudhury, R. Sharaf, V. Jain, A. Mukherjee, S. Sarkar, and A. Basu. Investigation and modeling of the structure of texting language. *Int. J. Doc. Anal. Recognition*, 34:157–174, 2007.
- [10] D. Contractor, T. Faruque, and L. Subramaniam. Unsupervised cleansing of noisy text. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pages 189–196. Association for Computational Linguistics, 2010.
- [11] P. Cook and S. Stevenson. 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, 2009.
- [12] J. Foster. Treebanks gone bad. *International Journal on Document Analysis and Recognition*, 10(3):129–145, 2007.
- [13] J. Foster. cba to check the spelling investigating parser performance on discussion forum posts. In *HLT-NAACL, 2010*, pages 381–384. Association for Computational Linguistics, 2010.
- [14] J. Foster and Ø. Andersen. Generate: 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, 2009.
- [15] J. Foster, J. Wagner, and J. Van Genabith. Adapting a wsj-trained parser to grammatically noisy text. In *Proceedings of ACL-HLT, 2008*, pages 221–224. Association for Computational Linguistics, 2008.
- [16] Y. How and M. Kan. Optimizing predictive text entry for short message service on mobile phones. In *Proceedings of HCI*. Citeseer, 2005.
- [17] M. Kaufmann and J. Kalita. Syntactic normalization of twitter messages. In *International Conference on Natural Language Processing (ICON)*, 2010.
- [18] C. Knoblock, D. Lopresti, S. Roy, and L. Subramaniam. Special issue on noisy text analytics. *International Journal on Document Analysis and Recognition*, 10(3):127–128, 2007.
- [19] C. Kobus, F. Yvon, and G. Damnaty. 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, 2008.
- [20] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open source toolkit for statistical machine translation. In *ACL 2007 Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics, 2007.
- [21] M. Kul. Phonology in text messages. *Poznań Studies in Contemporary Linguistics*, 43(2):43–57, 2007.
- [22] M. Marcus, M. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.
- [23] F. Och. Minimum error rate training in statistical machine translation. In *ACL, 2003*, pages 160–167. Association for Computational Linguistics, 2003.
- [24] K. Papineni, S. Roukos, T. Ward, and W. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the ACL 2002*, pages 311–318. Association for Computational Linguistics, 2002.
- [25] K. Raghunathan and S. Krawczyk. Cs224n: Investigating sms text normalization using statistical machine translation. 2009.
- [26] A. Stolcke. Srilmm—an extensible language modeling toolkit. In *Proceedings of the international conference on spoken language processing*, volume 2, pages 901–904. Citeseer, 2002.
- [27] L. Subramaniam, S. Roy, T. Faruque, and S. Negi. A survey of types of text noise and techniques to handle noisy text. In *Proceedings of The Third Workshop on Analytics for Noisy Unstructured Text Data*, pages 115–122. ACM, 2009.
- [28] R. Weide. The cmu pronunciation dictionary, release 0.6, 1998.