Available online at www.sciencedirect.com

SciVerse ScienceDirect

Computer Speech and Language 28 (2014) 256–277

COMPUTER SPEECH AND LANGUAGE

www.elsevier.com/locate/csl

Normalization of informal text

Deana L. Pennell *, Yang Liu

The University of Texas at Dallas, 800 West Campbell Road, Richardson, TX 75080, United States

Received 17 August 2011; received in revised form 2 March 2013; accepted 12 July 2013
Available online 23 July 2013

Abstract

This paper describes a noisy-channel approach for the normalization of informal text, such as that found in emails, chat rooms, and SMS messages. In particular, we introduce two character-level methods for the abbreviation modeling aspect of the noisy channel model: a statistical classifier using language-based features to decide whether a character is likely to be removed from a word, and a character-level machine translation model. A two-phase approach is used; in the first stage the possible candidates are generated using the selected abbreviation model and in the second stage we choose the best candidate by decoding using a language model. Overall we find that this approach works well and is on par with current research in the field.
Published by Elsevier Ltd.

Keywords: Text normalization; Noisy text; NLP applications

1. Introduction

Text messaging is a rapidly growing form of alternative communication for cell phones. This popularity has caused safety concerns leading many US states to pass laws prohibiting texting while driving. The technology is also difficult for users with visual impairments or physical handicaps to use. We believe a text-to-speech (TTS) system for cell phones can decrease these problems to promote safe travel and ease of use for all. Normalization is the usual first step for TTS.

SMS lingo is similar to the chatspeak that is prolific on forums, blogs and chatrooms. Screen readers will thus benefit from such technology, enabling visually impaired users to take part in internet culture. In addition, normalizing informal text is important for tasks such as information retrieval, summarization, and keyword, topic, sentiment and emotion detection, which are currently receiving a lot of attention for informal domains.

Normalization of informal text is complicated by the large number of abbreviations used. Some previous work on this problem used phrase-based machine translation (MT) for abbreviation normalization; however, a large annotated corpus is required for such a method since the learning is performed at the word level. By definition, this method cannot make a hypothesis for an abbreviation it did not see in training. This is a serious limitation in a domain where new words are created frequently and irregularly.

This work is an extension of our work in Pennell and Liu (2010, 2011, 2011). In this paper, we establish two sets of baseline results for this problem on our data set. The first uses a language model for decoding without use of an

* Corresponding author. Tel.: +1 469 585 5182.
E-mail addresses: deana@hlt.utdallas.edu (D.L. Pennell), yangl@hlt.utdallas.edu (Y. Liu).

0885-2308/$ – see front matter. Published by Elsevier Ltd.
http://dx.doi.org/10.1016/j.csl.2013.07.001

Table 1
Methods for processing unseen tokens during normalization.

Method	Formal example	Informal example
as chars	RSVP	"cu" (see you)
as word	NATO	"l8r" (later)
expand	Corp.	"prof" (professor)
combine	WinNT	"neway" (anyway)

abbreviation model, while the second utilizes a state-of-the art spell checking module, Jazzy Idzelis (2005). We then compare the use of our two abbreviation models for decoding informal text sentences. We also determine the effects on decoding accuracy when more or less context is available. Finally, we combine the two systems in various ways and demonstrate that a combined model performs better than both systems individually.

2. Related work

This section briefly describes relevant work in fields directly related to our research, though not always directly applied to informal text. We describe the tasks of modeling and expanding abbreviations in text as well as research on normalization of text in both formal and informal domains.

2.1. Abbreviation modeling and expansion

Abbreviation expansion is a common problem when processing text from any domain. Willis et al. (2002) studied abbreviation generation in the hopes of lowering the effort of text entry for people with motor disabilities; the user could enter abbreviated text that would be expanded by a system to be read by his or her conversation partner. They asked a group of young people to abbreviate a text of 500 characters to progressively smaller lengths, assuming they are charged per letter but there is a hefty fee for every error in decoding by another person. Although they do not attempt to expand the abbreviations automatically, they produce a set of rules by which participants produced abbreviations, using both deletion and substitution.

Early work by Pakhomov (2002) showed that medical abbreviations can be modeled and expanded using maximum entropy modeling. He used contextual information to help disambiguate medical terms assuming that an abbreviation and its correct expansion will be found in similar contexts.

Wong et al. (2006) introduced their ISSAC system that works on top of a spell checker (Aspell) to simultaneously perform spelling correction, abbreviation expansion and capitalization restoration. Their model gives weight to the normalized edit distance, domain significance, number of hits for a word on Google, appearance of the word/abbreviation pair (WAP) in an online abbreviation dictionary and the original weight given to a suggested correction by Aspell. In addition, a word is given more weight if it has been seen paired with the current abbreviation earlier in the document. Their reranking scheme provides a significant increase in accuracy over Aspell alone.

Yang et al. (2009) work with abbreviations for spoken Chinese rather than for English text messages, but their process is quite similar to our CRF system. They first perform an abbreviation generation task for words and then reverse the mapping in a look-up table. They use conditional random fields as a binary classifier to determine the probability of removing a Chinese character to form an abbreviation. They rerank the resulting abbreviations by using a length prior modeled from their training data and co-occurrence of the original word and generated abbreviation using web search.

2.2. Text normalization

Abbreviation expansion is just one of many techniques needed for the task of text normalization. Text normalization is an important first step for any text-to-speech (TTS) system. Regardless of the size of the training corpus, there will always be tokens that do not appear and have unknown pronunciations. Text normalization has been widely studied in many formal domains. Sproat et al. (2001) provides a good resource for text normalization and its associated problems.

In Table 1, we show some generally accepted processing methods for unknown words with examples from text messages and more formal text domain. Text message normalization presents many difficulties that are not encountered in other domains. This domain is constantly evolving; new abbreviations appear frequently and inconsistently. One user may abbreviate a single word in multiple ways. Abbreviations containing numbers and symbols are very uncommon in formal text but often seen in text messages. Spell-checking algorithms are mostly ineffective on this data, perhaps because they generally do not account for the characteristics of text messages. They instead focus on single typographic errors using edit distance, such as Kukich (1992), or a combination of this approach and pronunciation modeling, such as Toutanova and Moore (2002).

One line of research in this domain views normalization as a noisy channel problem. Choudhury et al. (2007) describe a supervised noisy channel model using HMMs for SMS normalization. Cook and Stevenson (2009) extend this work to create an unsupervised noisy channel approach using probabilistic models for common abbreviation types and choosing the English word with the highest probability after combining the models. Deletion-based abbreviations were addressed in our past work using statistical models (maximum entropy and conditional random fields) combined with an in-domain language model (LM) (Pennell and Liu, 2010, 2011). Liu et al. (2011) extend the statistical model to be independent of abbreviation type with good results. This system was later augmented to account for the effects of visual priming and unintentional spelling mistakes (Liu et al., 2012).

Whitelaw et al. (2009) used a noisy channel model based on orthographic edit distance using the web to generate a large set of automatically generated (noisy) WAPs to be used for training and for spelling suggestions. Although they use the web for collection, they do not focus on informal text but rather on unintentional spelling mistakes. Beaufort et al. (2010) combine a noisy channel model with a rule-based finite-state transducer and got reasonable results on French SMS, but have not tried their method on English text. Han and Baldwin (2011) first determine whether a given out-of-vocabulary (OOV) word needs to be expanded or is some other type of properly-formed OOV. For those predicted to be ill-formed, a confusion set of possible candidate words is generated based on a combination of lexical and phonetic edit distance and the top word is chosen in context using LM and dependency parse information. Han et al. (2012) attempts to automatically build a normalization dictionary offline using the distributional similarity of tokens combined with their string edit distance.

Machine translation (MT) techniques trained at the word- or phrase-level are also common. Translation of SMS from one language to another led Bangalore et al. (2002) to use consensus translations to bootstrap a translation system for instant messages and chat rooms where abbreviations are common. Aw et al. (2006) view SMS lingo as if it were another language with its own words and grammar to produce grammatically correct English sentences using MT. Henríquez and Hernández (2009) trained an MT system using three on-line SMS dictionaries for normalizing chat-like messages on Twitter. Kobus et al. (2008) incorporate a second phase in the translation model that maps characters in the texting abbreviation to phonemes, which are viewed as the output of an automatic speech recognition (ASR) system. They use a non-deterministic phonemic transducer to decode the phonemes into English words. The technical paper of Raghunathan and Krawczyk (2009) details an explicit study varying language model orders, distortion limit and maximum phrase length allowed by the MT system during decoding. Contractor et al. (2010) also uses an SMT model; however, in an attempt to get around the problem of collecting and annotating a large parallel corpus, they automatically create a noisy list of WAPs for training using some heuristics. As far as we know, our work was the first to use an MT system at the character-level for this task (Pennell and Liu, 2011). Li and Liu (2012) extended this model to include phonetic information.

3. Data

Three small SMS corpora are currently publicly available for current research: How and Kan (2005), Fairon and Paumier (2006), Choudhury et al. (2007). In addition, there is the Edinburgh Twitter Corpus (Petrovic et al., 2010), which is quite large but does not have the corresponding standard English transcription. The small Twitter corpus used in Han and Baldwin (2011) has also been released; this corpus has annotation and context but only contains 549 messages. Due to the lack of a large parallel corpus suitable for our study, we have built an annotated SMS-like corpus using status updates (called *tweets*) from http://twitter.com. Twitter allows users to update status messages by sending an SMS message to a number designated for the Twitter service. To ensure that our corpus is representative of the domain we are modeling, we use Twitter's metadata to collect only messages sent via SMS. Some examples of highly abbreviated messages from our corpus are shown below.

(a) Aye.oops,dat was spose to be a txt
(b) Rndm fct bout wife: n the past 10 yrs I can cnt on one hand the num Xs she's 4gotn to unlock my car door
(c) OMG I LOVE YOU GUYS. You pwn:) !!!
(d) i need to qo to bedd qotta wakee up at 7am for school....
(e) heard it again! xD 3 TIMES.a sng i nvr hear!

3.1. Choosing messages for annotation

An annotator's time is wasted if he is presented with many messages containing no abbreviations, or with sentences all containing the same, very common abbreviations. A scoring system to determine the order of annotation was thus devised using the following metrics:

1 **Word count index.** A low word count index indicates that a message is close to the mean message length. Messages with fewer than five words are removed from consideration. We calculate the index as $|N - E(N)|/\sigma(N)$, where N is the number of words in the message and mean and standard deviation are calculated over the entire corpus.
2 **Perplexity scores.** We use two perplexity scores calculated against character-level language models. We first find the perplexity of the message compared to standard English text. A lower score indicates that the message is less likely to be in a foreign language or non-linguistic content. We also calculate the perplexity of the message compared to our own corpus. A low score here indicates that the message is more representative of the domain. We remove the sentence completely if in either case the perplexity value is greater than a threshold (1000 in our study).
3 **OOV count.** This is a simple count of the number of out of vocabulary (OOV) words in the message compared to an English dictionary, which we denote N_{OOV}. This metric helps guarantee that we select messages containing many OOV words. We remove the sentence completely when $N_{\text{OOV}} = 0$.
4 **OOV percentages.** This metric consists of two scores: the first is N_{OOV}/N; the second is a non-duplicate OOV percentage, where we remove all repeated words and then recalculate the percentage. If the first score is greater than 0.5 but the second is not, we remove the message from consideration.
5 **OOV Frequency Score.** For each OOV token (including emoticons) we find the frequency of the token across the entire corpus and sum these across the entire *tweet*. This ensures that we annotate those abbreviations that are commonly used.

One sorted list is generated for each metric. A final score is generated for each sentence by a weighted average of its position in each list, where more weight is given to the non-duplicate OOV percentage list and less weight is given to the OOV frequency scores. The sentences are ranked in a penultimate list based on this final score. Finally, we complete a post processing step wherein we iterate through the list and remove sentences introducing no new OOV words when compared to higher-ranked sentences. Messages were then annotated in the order they appeared in the final scored list.

3.2. Annotation

Five undergraduate students were hired for annotation. The students were asked to use all available resources (including web searches or asking friends) to help when they were unsure of a term's meaning. In total, 4661 *tweets* were annotated[1]. Table 2 shows the number of annotations made per student[2]. **Tokens** refers to the total number of annotations made, including duplicates (*i.e.*, the same abbreviation was used in multiple *tweets*). **Types** refers to the number of annotations when duplicates are removed. **Unique** refers to those word/abbreviation pairs (WAPs) that only appeared in this student's annotations. **Unknown** refers to the number of tokens for which this annotator could not determine the standard English form.

[1] Our annotated data is available from http://www.hlt.utdallas.edu/~deana. Please cite this paper when using our data.
[2] Originally, the *tweets* were divided equally amongst the annotators, but some annotators left the university before the task was completed.

Table 2
Division of annotations in our data.

Annotator	1	2	3	4	5
Sentences	600	547	892	977	1925
Tokens	866	784	1251	1676	3192
Types	593	570	894	1125	2017
Unique	344	311	542	704	1439
Unknown	51	36	45	68	92

Table 3
Pairwise boolean agreement (%) between annotators. *A* indicates the author.

Annotator	1	2	3	4	5
A	98.6	96.8	96.3	97.2	97.7
1		97.0	97.0	96.7	98.1
2			96.4	95.7	97.0
3				97.0	96.4
4					97.1

Table 4
Distribution of maximum annotators in agreement per token.

Annotators in agreement	All disagree	2	3	4	5	6
# of tokens	0	3	5	15	30	425

Seventy-four messages were used for inter-annotator agreement. These messages were also annotated by the first author as a standard set for comparison. Unfortunately no single hired annotator completed all 74 messages. We first compute agreement at the boolean level; that is, whether a token was marked as an abbreviation or not regardless of the provided translation. Table 3 shows the pairwise agreement between annotators on tokens given to both people, including the 396 that both agreed were not abbreviations. We also computed Fleiss' Kappa; our calculated value of $\kappa = 0.891$ is quite high so the number of annotators can probably be reduced.

We also looked at the non-boolean agreement; that is, if two annotators marked a token as an abbreviation but provided different translations we consider them to be in disagreement. For this task, we consider the maximum number of annotators (per token) who are in agreement. As an example, if one annotator says a token is not an abbreviation at all, two say it is an annotation and translate it as *A* and the remaining three also believe it is an abbreviation but translate it as *B*, we consider the agreement to be 3.

Table 4 shows the distribution across only those tokens seen by all five annotators and the author across all 478 abbreviations. However, only 82 tokens were marked as an abbreviation by at least one person. Of these, only 29 had agreement between all six annotators.

3.3. Types of abbreviations

Abbreviations can be categorized by formation method. Cook and Stevenson (2009) proposed eleven categories that are somewhat subjective due to overlap. We propose five broad categories (three with subcategories) loosely based on edit distance in Table 5. Abbreviations can mostly be categorized by looking at the characters alone using our divisions, though ambiguity arises from silent 'e' removal (does "mayb" remove silent 'e', or replace "be" with 'b'?). Insertions and swaps often result from typos. The count of each type of abbreviation in our data is shown in Table 6. The vast majority of insertions are of the repetition type. For this reason, we categorize them separately here. Insertions of the phonetic type (such as *dawg* for *dog*) are considered substitutions for the purposes of this categorization.

Table 5
Abbreviation types. Subcategories are italicized below their parent category.

Category	Definition	Example
Deletion	Deletions only.	tmro (tomorrow)
Clipping	*Entire syllable(s) deleted*	*prof (professor)*
Location	*Silent 'e', Initial 'h' or -ing 'g'*	*talkin (talking)*
General	*Single word, not covered above*	*abt (about)*
Initialization	*First letter of each word*	*jk (just kidding)*
Substitution	Replacing characters with others	2nite (tonight)
As Sound	*Reading the token as a word*	*l8r (later)*[a]
As Character	*Pronouncing character's names*	*ne(any)*[a]
Symbolic	*Symbol(s) that look similar*	*$top (stop)*
Insertion	Word length is increased	lightining (lightning)
Repetition	*Letters repeated for emphasis*	*yesssss (yes)*
Stylistic	*Intentional changes*	*confrused (confused)*
Error	*Generally caused by a typo*	*jusyt (just)*
Swap	Correct letters in wrong positions	freind (friend)
Combination	Use of two or more of the above.	2sdy (Tuesday)

[a] As opposed to "ell-ate-arr" and "neh".

4. Method

For a given text message sentence, $A = a_1 a_2 \ldots a_n$, the problem of determining the sentence of standard English words, $W = w_1 w_2 ... w_n$, can be formally described as below, similar to speech recognition and machine translation problems:

$$\begin{aligned} \widehat{W} &= arg\max P(W|A) \\ &= arg\max P(W)P(A|W) \\ &\approx arg\max \prod P(w_i|w_{i-n+1}...w_{i-1}) \times \prod P(a_i|w_i) \\ &= arg\max(\sum \log P(w_i|w_{i-n+1}...w_{i-1}) + \sum \log P(a_i|w_i)) \end{aligned} \tag{1}$$

where the approximation is based on the assumption that each abbreviation depends only on the corresponding word (note that we are not considering one-to-many mappings in this study), and a word is dependent on its previous $(n-1)$ words. In other words, this probability is represented by a traditional n-gram language model.

Language modeling, and n-gram modeling in particular, is an important part of natural language processing. We expect modeling the texting language domain will help us achieve better disambiguation when normalizing abbreviations in the text. Word-level language model information, $P(w_i|w_{i-n+1}...w_{i-1})$, is used in Eq. (1) to help disambiguate multiple word hypotheses for an abbreviation when translating an entire text message. In this work we utilize unigram, bigram and trigram language models to test the effects of the language modeling model order on our task.

The abbreviation score in Eq. (1), $P(a_i|w_i)$, represents the likelihood that an abbreviation a_i is derived from word w_i. There are many possible ways to create an abbreviation model (AM) to specify $P(a_i|w_i)$. We utilize two methods: Section 4.1 describes a conditional random field model specific to those abbreviations formed by only deletions and Section 4.2 describes a more general model using a character-level machine translation system. Both of these methods work entirely at the character level and have no knowledge of the surrounding context.

Table 6
Division of abbreviation types in our data.

Deletion	Substitution	Repetition	Swap	Insertion	Combin.
1406	620	510	106	55	140

```
Word: suggestion        Abbr: sgstn

Tags: s/Y u/N g/Y g/N e/N s/Y t/Y i/N o/N n/Y
```

Fig. 1. Character-level tags for one abbreviation of 'suggestion'.

Eq. (1) assumes that the abbreviation model and language model should be weighted equally, but in actuality one model may prove to be more helpful than the other. For this reason, we allow the terms from Eq. (1) to be weighted differently, yielding the final equation

$$\widehat{W} = arg\max(\alpha \sum \log P(w_i|w_{i-n+1}...w_{i-1}) + \beta \sum \log P(a_i|w_i)) \quad (2)$$

where α and β are determined empirically.

Our system is thus a two-stage process: In the first stage we generate the $P(a|w)$ scores, and in the second stage we combine these scores with a language model using Eq. (2) to incorporate context. We use standard language models; our contribution is in the abbreviation modeling aspect of the problem. We describe two methods for abbreviation modeling in the remainder of this section: a sequential model for deletions and a character-level machine translation model that addresses any abbreviation type.

4.1. Deletion modeling

We first focused only on deletion-based abbreviations of single words. Not only does our annotated data show deletions to be the most common abbreviation type, but this is a very difficult type of abbreviation for TTS – our end goal. For the next most frequent abbreviation type (*Substitution:As Sound*), reasonable pronunciations can be obtained using commonly known letter-to-phoneme rules. Even when a TTS system does not include the token in its lexicon, it is pronounced in a way the end user understands.

Concentrating only on the *Deletion* class allows us to view this task as a binary classification problem. For each word/abbreviation pair (WAP), we perform a tagging task at the character level, shown in Fig. 1. A tag of 'N' means that the character should be removed to form the abbreviation, while a tag of 'Y' means that the character should remain in the word. We use conditional random fields (CRFs) as the classifier for this task.

4.1.1. Features

We created the following feature set for use with the CRFs after examining representative abbreviations from each class described in Table 5.

1. **Contextual features**

   We hypothesized that the contextual features would help with the *Location* class of abbreviations as well as some *General* deletions. These features can help the classifier learn whether a particular character is more or less likely to be deleted when appearing at the beginning or end of a word, or when surrounded by certain other characters. The tokens <s> and </s> represent the beginning and ending of a word.

   (a) The character itself, c_i.
   (b,c) The two previous characters, c_{i-1} and c_{i-2}.
   (d,e) The following two characters, c_{i+1} and c_{i+2}.
   (f,g) The two bigrams containing current character, $c_{i-1}c_i$ and c_ic_{i+1}.
   (h,i,j) The three trigrams containing the current character, $c_{i-2}c_{i-1}c_i$, $c_{i-1}c_ic_{i+1}$ and $c_ic_{i+1}c_{i+2}$.

2. **Function features**

   The function features were also intended to help with the *General* deletion abbreviations, because we observed that vowels and doubled consonants are more likely to be deleted than other characters. Feature 2d was included to help locate the silent 'e' character since we do not include pronunciation features. We assume any syllable-final 'e' character is silent when preceded by a consonant. A silent 'e' determined in this fashion is marked as a consonant in Feature 2b.

```
s  y  l / l  a / b  l  e
|  |  |   |  |   |  |  |
B  M  E   B  E   B  B  E
```

Fig. 2. An example of syllable positions assigned to characters. The beginning, middle and end positions of a syllable are noted by 'B', 'M' and 'E', respectively.

(a) Whether it is identical to the previous character.
(b) Whether the character serves as a vowel.
(c) Concatenation of features 2a and 2b.
(d) Concatenation of features 1a and 2b.

3. **Syllabic features**

These features were intended to help generate the *Clipping* subclass of abbreviations, where entire syllables are removed from words. These features allow us to know whether a syllable's position or characters increase its likelihood of being deleted. To find syllable boundaries, we currently use the free online dictionary at http://www.dictionary.com. This website does not list words containing prefixes or suffixes as separate entries from the base word, which causes these words to be incorrectly syllablized. To address this issue, we plan to implement an automatic syllabification method Bartlett et al. (2008). Once the syllable information is known, features 3a–3d are easy to extract. For single syllable words, features 3b and 3c fire, while 3d does not.

For feature 3f, we determine a character's position (**B**eginning, **M**iddle or **E**nd) within a syllable as follows. When a consonant falls to the left of the sonorant vowel(s) in the syllable (that is, in the onset of the syllable), it is labeled **B**, while those on the right (the coda) are labeled **E**. Vowels with consonants on both sides are labeled **M**. Those with no consonants to the left are assigned **B** and those with none on the right are assigned **E**. If a syllable consists of only vowels, they are labeled **B**. Again, we consider a silent 'e' to be a consonant, since it does not function as a vowel in the syllable. This feature is included because we observed that vowels in the **M** position seemed to be much more likely to be deleted than vowels in other positions. Note that these do not exactly correspond to a syllable's onset, nucleus and coda. This is by design, since vowels (excepting silent 'e') are always considered to be the syllable's nucleus, which does not give us discriminating power. An example showing positions of characters in the word "syllable" is shown in Fig. 2.

Yang et al. (2009) found that for their Chinese abbreviation CRF model, the best features were the current character, the word in which the character appeared, the character's position within the word, and a combination of the final two features. In our task, this is analogous to using Feature 3a (syllable) and Feature 3f (position in syllable). Thus, we include Features 3g, 3h and 3i. Feature set 3j-n learns whether certain letters are more likely to be removed when in certain positions in a word or syllable.

(a) The characters making up the current syllable.
(b) Whether the character is in the first syllable of the word.
(c) Whether the character is in the last syllable of the word.
(d) Whether the character is in neither the first nor last syllable.
(e) Concatenation of features 3b, 3c and 3d.
(f) The character's position in its syllable.
(g) Concatenation of features 3a and 3f.
(h,i) Two features concatenating 3f with both 3b and 3c.
(j,k,l,m,n) Features resulting from concatenation of 1a with each of 3a, 3b, 3c, 3d and 3f.

4. **Sequential feature knowledge**

CRFs enable us to use the classification of the previous character as a final feature.

Each training example is a single English word and the corresponding features for each character. An example is shown in Fig. 3. Each column contains a feature value, with the final column representing the truth value of whether the character should be deleted to form the abbreviation. The same word often appears multiple times with different truth values: once for each possible abbreviation seen in the corpus. For instance, the word "know" appears twice in the training data corresponding to the abbreviations "kno" and "no", leading to ambiguity; the features for the letter 'k' will be identical, but the truth value is 0 for the 'k' in "kno" and 1 for the 'k' in "no". The training examples may

```
v <s> <s> e r <s>v ve <s><s>v <s>ve ver N N NN vN ver Y \
   N N YNN B verB YB NB vver vY vN vN vB 0
e v <s> r y ve er <s>ve ver ery N Y NY eY ver Y \
   N N YNN M verM YM NM ever eY eN eN eM 1
r e v y </s> er ry ver ery ry</s> N Y NY rN ver Y \
   N N YNN E verE YE NE rver rY rN rN rE 1
y r e </s> </s> ry y</s> ery ry</s> y</s></s> N Y NY yY y N \
   Y N NYN B yB NB YB yy yN yY yN yB 1
```

Fig. 3. Training sequence for the word "very" with features representing the common abbreviation "v".

be given to the CRF in any order. Words already in standard form are not submitted as training examples. The CRF is not trained with characters that cross word boundaries; that is, the context features (features 1b – 1j) do not contain spaces or characters from the previous or next word, only <s> and </s>.

Generating Word Candidates. To ensure that our system is robust to the numerous and inconsistent variations in abbreviating, we want to generate multiple reasonable abbreviations for each word. The posterior probability of the tag for each character c generated by the classifier can be used to compute an abbreviation score. For each abbreviation a of an N-character word w, its abbreviation score is:

$$S_a = \prod_{i=1}^{N} sc(c_i), \tag{3}$$

where

$$sc(c_i) = \begin{cases} p(c_i) & c_i \in a \\ 1 - p(c_i) & \text{otherwise} \end{cases} \tag{4}$$

c_i is the *ith* character of word w, and $p(c_i)$ is the posterior probability of the 'Y' class.

An example output is shown for the word "know" in Fig. 4. Using the scores shown in the figure, we can see that the score for the abbreviation "no" is $0.168478 * 0.975795 * 0.805093 * 0.788493 \approx 0.1043$ while the abbreviation "ow" is much less likely, with a score of approximately 0.0006.

Reranking. For a four character word such as "know", it is reasonable to generate all 16 possible deletion-only abbreviations. However, for longer words it becomes infeasible to generate and keep all of the 2^N possibilities. In addition, it is extremely unlikely that the word "characterization" would be abbreviated with only "ar"; including pairs such as this in the list of possibilities forces the system to examine many very improbable choices when the abbreviation "ar" is encountered in testing. The normalization performance is bounded by the number of WAPs from the test set that appear in our list of possible translations. That is, it will be impossible for our system to suggest "know" as a possible normalization of the term "knw" in testing if "knw" is not one of the abbreviations retained in the previous step. For this reason, we rerank the list of abbreviations generated for each word to maximize coverage, and then choose the highest ranked words as candidates. Based on the finding in Yang et al. (2009) that there is a strong correlation between the length of the formal text and the length of the abbreviation for Chinese organizations, we use a simple algorithm to rescore the candidate abbreviations. We combine the original abbreviation score with the length based information,

```
k Y/0.831522 N/0.168478
n Y/0.975795 N/0.024205
o Y/0.805093 N/0.194907
w Y/0.201507 N/0.788493
```

Fig. 4. Sample output of posterior probabilities for deletion of each character in the word "know".

```
a b ||| a _ b ||| 0.5 0.880834 0.010989 0.164514 2.718
a b ||| a b b ||| 0.181818 0.880834 0.00732601 0.905748 2.718
a b ||| a b o u ||| 0.00714286 0.880834 0.003663 0.00346958 2.718
a b ||| a b o ||| 0.00662252 0.880834 0.003663 0.0690538 2.718
a b ||| a b u l a r y ||| 1 0.440819 0.003663 4.55505e-08 2.718
a b ||| a b ||| 0.894737 0.880834 0.934066 0.934992 2.718
a b ||| a p ||| 0.00956938 0.000874125 0.00732601 0.000887968 2.718
a b ||| a v e ||| 0.00249377 0.000970303 0.003663 3.35128e-05 2.718
a b ||| b ||| 0.000325839 0.308415 0.003663 0.968721 2.718
a b ||| i b ||| 0.0238095 0.308415 0.003663 0.077216 2.718
```

Fig. 5. An excerpt from a phrase table showing possible translations when the character sequence "ab" is found in a message.

i.e., the probability of using an M-character abbreviation for an N-character word, $P(M|N)$, which we gather from the training set. The new score S'_a is thus:

$$S'_a \approx \alpha \log P(M|N) + \beta \log S_a, \tag{5}$$

where α and β determine the weighting for each model, while S_a refers to Eq. (3)). Note that S'_a is a representation of $P(a_i|w_i)$.

In decoding, we want to find all possible w_i for a given a_i as quickly as possible. Therefore the lookup table is reversed from the generation process – for an abbreviation, we store a list of possible words and their corresponding scores (S_a from Eq. hyperlinkeq:abbr_score(5)).

4.2. A machine translation model

The second abbreviation model that we investigated is a machine translation (MT) method. Rather than the typical MT set-up that translates word sequences in one language to words in another, we translate character sequences in abbreviations to their standard character sequences. Formally, for an abbreviation $a : c_1(a), c_2(a), \ldots, c_m(a)$ (where $c_i(a)$ is the *ith* character in the abbreviation), we use an MT system to find the best word hypothesis:

$$\begin{aligned} \hat{w} &= \text{argmax}\, p(w|a) \\ &= \text{argmax}\, p(w)p(a|w) \\ &= \text{argmax}\, p(c_1(w), \ldots c_n(w)) \times p(c_1(a), \ldots, c_m(a)|c_1(w), \ldots c_n(w)) \end{aligned} \tag{6}$$

where $c_i(w)$ is a character in the English word, $p(c_1(w), \ldots c_n(w))$ comes from the character LM, and $p(c_1(a), \ldots, c_m(a)|c_1(w), \ldots c_n(w))$ is based on the learned phrase translation table. Note the similarity to Eq. (1).

The translation model is trained using the annotated WAPs after the following preprocessing. We remove those abbreviations which the annotators were unable to provide a translation and those marked as sound effects (*e.g.*, ahhhhh, zzzzz, blech, hrmpf, etc.). We remove all punctuation, excluding that found in emoticons (which we treat as words) and apostrophes in common contractions and possessive forms. To facilitate character-level training, we replace any spaces with underscores and insert spaces between characters.

Due to the character-level training, each hypothesis ($\hat{w}$) for an abbreviation (a) is a sequence of characters, which may or may not produce a valid word. To see why this is possible, examine the partial phrase-table[3] shown in Fig. 5; using this table, "hab" could be translated to "hib", "habulary" or "habou" as well as the correct word "have". It is also possible, and in fact very likely, for a hypothesis to appear many times in the N-best hypothesis list due to different segmentation (two characters may be generated by a single phrase mapping or by two different mappings, one for each

[3] Aside from the possible translations, the phrase table also shows five values for each word. They correspond to the inverse phrase translation probability $\phi(f|e)$, the inverse lexical weighting $lex(f|e)$, the direct phrase translation probability $\phi(e|f)$, the direct lexical weighting $lex(e|f)$ and the phrase penalty (always $exp(1) = 2.718$), where e and f are the English and foreign phrases, respectively. We do not currently make use of these values explicitly, though the MT system uses them for decoding.

character). We generate the top twenty *distinct* hypotheses for each abbreviation and then eliminate those hypotheses that do not occur in the CMU Lexicon.

We then use the scores for these hypotheses directly in Eq. (1) to choose the best standard form; the abbreviation score, $P(a_i|w_i)$, represents the likelihood that abbreviation a_i is derived from word w_i, and can be obtained from:

$$p(a_i|w_i) \propto \frac{p(w_i|a_i)}{p(w_i)} \tag{7}$$

where $p(w_i|a_i)$ is the score from the character-level MT system, and $p(w_i)$ is from the character LM used in MT decoding. We use the score from the character MT system as the likelihood score without dividing by the character-level LM contribution. This is equivalent to using both a character-level and a word-level LM during decoding.

5. Experiments

5.1. Experimental setup

With the exception of tests to establish baselines, we used a cross validation setup for our experiments. The data from four annotators is used as training data, while the data from the fifth annotator is divided in half for development and testing. For each fold we perform two tests; initially we use the first half for development and test on the second half, then the development and test portions were swapped. The results shown here are averaged over all ten tests.

The language model (LM) we use during decoding is a trigram language model generated using the SRILM toolkit (Stolcke, 2002) with Kneser-Ney discounting. To train the model we use those messages from the Edinburgh Twitter corpus (Petrovic et al., 2010) containing no out-of-vocabulary (OOV) words compared to a dictionary. We constrain SRILM to use the appropriate order LM for the amount of context given during testing.

Throughout the experiments there are two sets of abbreviations on which we perform tests. The first is made up of all single-word abbreviations (SW) in our annotated data. This data does not include those annotations where a single token was mapped to multiple words or when multiple tokens (or parts of multiple tokens) are mapped to one or more words. During preprocessing we also removed those tokens annotated as sound effects or those tokens where the annotator was unable to guess a translation even though he or she recognized that it was an abbreviation. The second test set is made up of only those abbreviations that can be formed from their annotated standard form by deletions only (DEL). This is necessary because our CRF model is designed to address this and only this type of abbreviation. For fair comparison, we also test all other systems on this deletion-only set.

5.1.1. CRF system setup

We use the open source software CRF++[4] for our experiments. Abbreviations for all words in the CMU lexicon are generated and stored in the lookup table. For an n-character word, we first generate $2n$ abbreviations using the setup under test. The weight for the length model and the set of features used are tuned simultaneously as follows. We perform both forward and backward feature selection to find a good feature set. For each feature set tested during feature selection, we rerank the generated list using various weights for the length model described in Section 4.1 and then prune the list to length $n-1$. We then calculate the lookup table's coverage of the development set and use the feature/weight combination that yields the maximum coverage for testing. During backward selection, we often found that removing even one feature decreased performance. For this reason, we also performed tests using all features.

In general, the forward selection setup yielded slightly higher coverage on the development data than the backward selection setup. For this reason, we ran tests using the optimized forward selection features and also using the set of all features. The set of all features outperformed the forward selection set by a fair margin (over 10 percentage points in some tests). For this reason, we present only the results using all 29 of the features listed in Section 4.1.

A histogram showing the number of times each feature was selected over the 10 development experiments using forward selection is shown in Fig. 6. Knowledge of the previous character's classification is very useful, and trigram context information is also important. Of the syllable related features, the actual characters in the syllable are selected most often.

[4] http://crfpp.sourceforge.net/

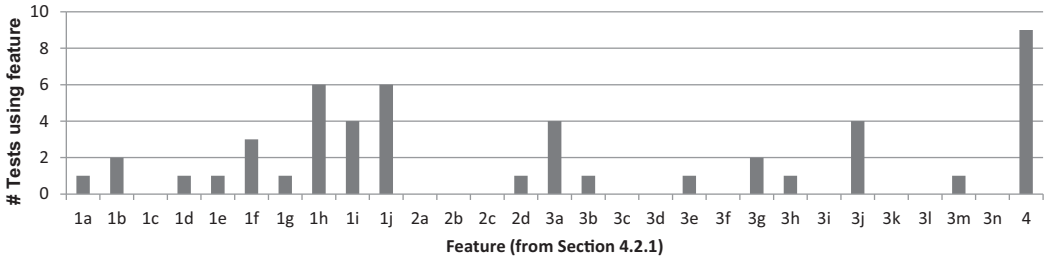

Fig. 6. Distribution of chosen features across the development tests.

To select a weight for the length model, we held the score model weight steady at 1 and varied the length model weight from 0.01 to 100. The best performing weight varied so much during the development tests that it is difficult to suggest a good weight to choose for future tests. We show histograms of the chosen weights in Fig. 7. After the forward selection setup, the length weights are well-centered around 1. The backward selection weights appear more spread out, but are still somewhat centered on a weight of 1.

The values $2n$ and $n-1$ were found empirically to be a good balance between the size of the look-up table and accuracy. These generated pairs are then reversed and stored in the look-up table used to find the hypotheses and scores during testing. Finally, we use the development set to tune the weights of the language model and the resultant score model.

5.1.2. MT system setup

We use the popular open source SMT implementation, Moses (Koehn et al., 2007), for all of our experiments. The MT system works by generating words from the abbreviations. Therefore it requires much less setup and tuning as we give the abbreviations in our test set to Moses directly. As mentioned above, we generate the top twenty *distinct* hypotheses for each abbreviation and eliminate those hypotheses that do not occur in the CMU Lexicon. Once again, we use the development set to tune the weights for the abbreviation model (AM) and the language model (LM) and use the top performing weights for evaluation on the test set. Note that Moses may generate positive score (impossible for a real log-probability). Because a log probability is expected during decoding, we replace any positive value with -0.1, indicating that this pair is very likely.

5.2. Baseline experiments

Before we test our methods, we first establish baselines with which we can compare our results. We provide two baselines: the first uses a language model (LM) for decoding without combining it with our system scores, while the second uses a state-of-the-art spelling checker, Jazzy (Idzelis, 2005).

5.2.1. Language model baseline

We use an LM for decoding in combination with the scores generated from one of our methods, so a natural question is how well the LM performs on its own before our scores are added. We do not conduct experiments without context for this baseline because without context the LM will always choose the same word (the one with the highest unigram probability).

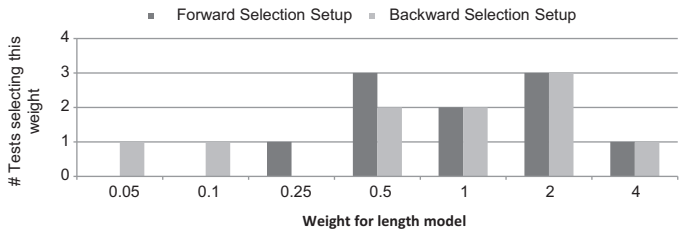

Fig. 7. Distribution of chosen length model weight across the development tests.

-5.596789 i abandon	-2.358411 abandon that	-7.9552
-5.131902 i abandoned	-2.389068 abandoned that	-7.52097
-5.032517 i abhor	-1.806325 abhor that	-6.838842
-4.038481 i accept	-1.06385 accept that	-5.102331
-3.875253 i accidentally	-3.851014 accidentally that	-7.726267
-3.138025 i ate	-1.999238 ate that	-5.137263
-1.334690 i have	-2.220222 have that	-3.554912
-1.588771 i know	-1.253628 know that	-2.842399
-1.347307 i love	-1.442284 love that	-2.789591
-2.184133 i see	-1.496268 see that	-3.680401

Fig. 8. Bigram LM baseline example for the observation "i kno that".

We thus test the LM baseline using one or two words of context on either side of the abbreviation. For a test case $w_1 A w_2$ ($w_1 w_2 A w_3 w_4$) we extract from the LM all bigrams (trigrams) that begin with the left context word(s), as well as their scores. We consider the second (third) word in the bigram (trigram) as a candidate *A'* for the translation of *A*. We then extract from the LM the score for the bigram (trigram) beginning with *A'* followed by the right context. For trigram testing, we also extract the score for the trigram $w_2 A' w_3$. As the scores produces by SRILM are log-probabilities, we merely add the scores to find a final score for the word *A'*.

In the exceptional case where no bigram (trigram) beginning with the left context exists, we back off to the unigram (bigram) model and consider all unigrams as possible candidates for *A*. We also back off when there are no bigrams (trigrams) beginning with a candidate *A'*, or in the case of *trigram* testing, when the trigram $w_2 A' w_3$ does not appear in the model.

Finally the candidates are ordered by their scores and the highest ranking candidate *A'* is chosen as the translation of the abbreviation *A*.

For clarity, an example is shown in Table 8. The first column shows an excerpt from the language model of bigrams beginning with the word "i" (there are 4646 bigrams that begin with "i" in our LM). The second column shows the bigrams beginning with the corresponding word from the first column and ending with the word "that". The rightmost column shows the final score, which is the sum of the scores in the first two columns. In this example, the word "love" has the best score and is chosen. Note that this method does not take into account any similarity between the abbreviation "kno" in the original text and the 4646 possibilities obtained from the language model and would suggest "love" for any abbreviation appearing between "i" and "that". The abbreviation models we introduce serve to address this problem, and limit the number of candidate words that need to be checked.

The results of these tests are shown in Table 7. We calculate the top-*N* accuracy for values $N = 1$, 3, 10, and 20, where a system is considered correct in top-*N* if the correct translation appears in the first *N* hypotheses given by that system. We also compute the number of correct answers eventually **found** by the system regardless of how far down the list it appears.

Using only the LM performs very poorly, as expected. The results are similar for the two test sets, which is to be expected since the LM has no knowledge of the abbreviation form, or even what abbreviation originally appeared in the context. Note that while the trigram context outperforms the bigram context in general, the bigram context has a higher **found** percentage. This is because the trigram context eliminates the correct word *C* from consideration when the trigram $W_1 W_2 C$ does not appear in the background LM, which happens somewhat frequently.

Table 7
Top-*N* accuracy (%) using only a language model for decoding.

	Top-1	Top-3	Top-10	Found
Bigrams (SW)	13.95	24.86	38.98	91.69
Trigrams (SW)	18.32	28.51	39.22	75.54
Bigrams (DEL)	13.15	24.82	39.31	90.7
Trigrams (DEL)	17.32	26.82	37.50	74.51

Table 8
Top-*N* accuracy (%) using Jazzy spell checker.

	Top-1	Top-3	Top-10	Top-20	Found
SW	37.91	39.47	44.25	44.53	44.60
DEL	34.68	36.72	43.40	43.51	43.60

Table 9
Comparison of systems without LM information.

	CRF (DEL)	MT (DEL)	MT (SW)
Top-1	37.27	54.35	55.90
Top-3	43.75	74.15	72.81
Top-10	48.86	79.69	77.07
Found	51.18	80.72	77.74

5.2.2. Jazzy spell checker

Similar to Liu et al. (2011), we wish to compare our work to the state-of-the-art spell checking algorithm, Jazzy (Idzelis, 2005). Jazzy is based on the Aspell algorithm and integrates a phonetic matching algorithm (Double Metaphone) and Ispell's near miss strategy enabling the interchanging of adjacent letters, as well as the insertion, deletion and substitution of letters.

Rather than using the small dictionary included with the Jazzy source, we used an Aspell dictionary[5]. Initial tests using both dictionaries indicated better performance using the larger Aspell dictionary. We attempted to use the Aspell phonetic dictionary[5] as well, however it degraded performance significantly compared to Jazzy's default setup. We modify Jazzy's configuration file to tell it to suggest corrections for words containing digits; by default it ignores these tokens, which are common in this domain.

Jazzy's predictions do not change based on context, so we perform only unigram-based tests. When Jazzy believes a word to be misspelled, it returns a ranked list of words. We treat this as the *N*-best list and perform top-*N* accuracy evaluations as before. As we only submit abbreviations that need to be translated, if Jazzy does not provide any suggestions we automatically mark it as incorrect. The results for Jazzy on our two data sets are shown in Table 8. We find the results on our data to be lower than those obtained by Liu et al. (2011) (on their own test set) by a fair margin.

5.3. Single system experiments

To perform experiments, we must first generate scores using each system. We describe the methods of generating the scores used in testing and follow that with a presentation of the results.

5.3.1. Comparison of abbreviation models alone

The first question we ask is how well the scores generated by each system perform on their own, without any information from a language model. Without a language model, our system cannot make use of context, so we submit only an abbreviation for each test case. We calculate top-*N* accuracy for $N = 1$, 3 and 10 as well as the total number found regardless of position. Results for each system are presented in Table 9. For comparison, we also tested the MT system on the DEL test set.

By examining the "found" row in the table, we can see that the MT model produces a higher coverage of the test set, in other words, it has higher *recall*. This implies that (with proper reranking) it has the potential to obtain higher results in a final system. We also see that it performs significantly better at each step in the *N*-best list. We can create an analog for *precision* by examining the ratio of the top-1 score to the "found" score: for CRFs we find $37.27/51.18 = 72.8\%$ of the abbreviations found had the highest ranked hypothesis being correct. For the MT system, this value is only 67.3%. CRFs thus produces a better initial model in terms of ranking.

[5] en_US dictionary and en_phonet.dat from Ubuntu, package version 6.0-0-6ubuntu1

Table 10
Comparison of systems using both abbreviation and LM scores.

		CRF (DEL)	MT (DEL)	MT (SW)
Top-1	Unigram	45.48	62.09	64.19
	Bigram	46.81	69.59	69.69
	Trigram	47.44	70.58	70.44
Top-3	Unigram	49.84	79.45	76.91
	Bigram	50.39	76.39	75.17
	Trigram	50.39	77.57	75.87
Top-10	Unigram	51.17	80.71	77.73
	Bigram	51.09	80.71	77.73
	Trigram	51.09	80.71	77.73
Found		51.17	80.71	77.73

5.3.2. Incorporation of an LM for decoding

Next, we tested each system when incorporating LM information using Eq. (2). We first generate hypotheses for the abbreviation using the system under test and then decode using the LM to find the final score for each hypothesis. The hypotheses are ranked by score and the highest scoring word is suggested. We use SRILM to generate the LM scores automatically. We tested using no context (*unigram*) and one or two words of context on either side (*bigram* and *trigram*, respectively) in order to see how context affects our results. For these tests, abbreviated context words are replaced with their annotated standard form to yield the best chance of decoding.

During development, we first run the tests using only trigram context on the development set and optimize the top-1 accuracy. We set $\alpha = 1$ and vary β from 0.01 to 100. In both systems, we find that lower values of β generally produce better results. The final values for β are 0.1 and 0.05 for CRF and MT respectively. The results of these experiments are shown in Table 10.

Once again, we see that the MT system outperforms the CRF system by a large margin at all places in the top-N. We also note that, using the same *precision* analogue as before, the MT system (on DEL using a trigram context) reaches only 87.4%, whereas the CRF system is able to reach 92.7%. This shows that there is more work that can be done to obtain better ranking of results from the MT system in the future.

Regarding the language model, we find that increasing the order of the LM leads to an increase in performance. We see a consistent jump in accuracy when increasing the order from unigram to the bigram, with a smaller increase from bigram to trigram.

5.3.3. Decoding an entire message

We also test our systems at the message level, where the entire message is given to the system for decoding. When performing tests, we replace any abbreviation not in the current test set (SW or DEL) with its standard form. This yields original word error rates (WERs) of 9.49% for DEL and 11.6% for SW. Note that the original error rate of the data set is even higher due to the inclusion of multi-word abbreviations and onomatopoeia.

It is common for abbreviations to appear in the context of other abbreviations, which poses a more difficult problem for decoding. We use the SRILM lattice-tool to generate the 20 best sentences. The lattice-tool is typically used to combine a language model with an acoustic model; our abbreviation model replaces the acoustic model during decoding. The decoding is easier in our case because we force a strict one-to-one alignment, which is not usually present in speech-to-text. We use the same LM as in previous tests and constrain decoding to use order 1, 2 or 3 to test the impact of LM order during decoding.

There are many more metrics when performing sentence-level decoding. We first report top-N accuracy for abbreviations only to see what degradation of performance occurs due to the added difficulty of decoding abbreviations when there are other abbreviated words nearby. An abbreviation is correct in top-N if the word is correctly translated in at least one of the N-best sentences. The results are shown in Table 11.

Note that when using the above metric a score of 100% in top-N would not guarantee that any single top-N sentence is 100% correct. One abbreviation may be correct in sentence $i < N$ but incorrect in sentence $j < N$, where the opposite is true of a second abbreviation. In addition, it is possible that our system may falsely "correct" a word that was already correct in the original sentence, changing it to another English word.

Table 11
Message decoding results on abbreviation accuracy.

		CRF (DEL)	MT (DEL)	MT (SW)
Top-1	Unigram	50.37	59.81	62.63
	Bigram	51.69	64.87	66.56
	Trigram	52.48	66.34	67.90
Top-3	Unigram	51.85	71.44	70.77
	Bigram	53.08	75.18	73.67
	Trigram	53.92	75.34	73.80
Top-10	Unigram	52.84	77.45	75.03
	Bigram	54.67	79.56	76.81
	Trigram	55.35	79.37	76.81
Found	Unigram	53.20	79.05	76.24
	Bigram	55.47	80.28	77.38
	Trigram	55.67	79.88	77.25

Table 12
Sentence-level top-*N* accuracy (%) using lattice decoding.

		CRF (DEL)	MT (DEL)	MT (SW)
Top-1	Unigram	0.05	31.47	31.36
	Bigram	1.54	37.16	36.55
	Trigram	4.68	38.64	37.98
Top-3	Unigram	0.11	43.09	40.83
	Bigram	2.25	48.60	45.54
	Trigram	6.81	49.49	46.49
Top-10	Unigram	0.41	50.50	46.45
	Bigram	3.49	54.83	50.93
	Trigram	9.78	58.56	54.37
Found	Unigram	0.53	53.23	49.18
	Bigram	4.09	59.45	55.92
	Trigram	11.32	66.56	62.49

Thus, it is important that we also look at the top-*N* sentences to determine how often we are able to fully correct a message. In Table 12, a system is marked as correct in top-*N* if one of the top-*N* decoded sentences exactly matches the reference sentence. This means that the system corrected all abbreviations in the original sentence but did not mistakenly change any words that were already in their proper standard form.

We see that the sentence accuracy is quite poor, especially in the case of the CRF system. By examining Table 13, we see that the major culprit here is over generation by the CRF model; words that were already in their standard form are frequently changed to other English words. As an example of this type of error, we show the 1-best hypothesis the CRF model for the sentence "`when i get there u can have ur hug [username]`" (*when i get there you can have your hug [username]*):

```
when it gets there you can have your huge [username]
```

Table 13
False positive (FP) rate (%) and final WER in top-1 sentences.

	CRF (DEL)		**MT (DEL)**		**MT (SW)**	
	FP	WER	FP	WER	FP	WER
Unigram	52.33	52.1	4.14	7.6	4.07	7.9
Bigram	38.64	39.6	3.61	6.6	3.50	7.0
Trigram	27.82	29.7	3.56	6.4	3.44	6.8

Table 14
System generated hypotheses but only using LM scores during decoding.

		CRF (DEL)	MT (DEL)	MT (SW)
Top-1	Unigram	45.56	60.36	62.76
	Bigram	46.15	69.04	69.42
	Trigram	47.52	70.34	70.35
Top-3	Unigram	49.84	79.40	76.82
	Bigram	50.39	76.39	75.22
	Trigram	50.39	77.29	75.76
Top-10	Unigram	51.17	80.71	77.73
	Bigram	51.09	80.71	77.73
	Trigram	50.39	80.71	77.73
Found		51.17	80.71	77.73

The system corrected both abbreviations, but also changed three words that it should not have. Looking further down the *N*-best list, "i" and "hug" are never left as-is, and "get" does not appear until hypothesis number 10. The word "get" is expanded to "gets", "getting", "great", and "ghetto" in various instances.

It is clear from the table that the MT system performs far better, especially as we look farther down the top-*N* list. The MT system is able to achieve an overall decrease in WER for both datasets (recall the original WER was 9.49 and 11.6 for DEL and SW, respectively), but we still feel that it generates too many false positives. We have done some work to combat this problem using heuristics; this is addressed in Section 6. Another source of error for the MT system stems from repetitions. There are more examples of smaller numbers of repetitions in the training data (for example, "yesss" appears more times than "yesssssssss"). Longer repetitions are more likely to be singletons in the data. In addition, our use of a 5-gram character model prevents the system from learning about the entirety of a long span of repetitions and its context. For these reasons, the MT system will reduce nine "s" characters to three or four, rather than one. To combat this, we have since taken the approach used in Liu et al. (2011) involving the preprocessing of repetitions down to only two consecutive identical characters before training.

Neologisms and proper names are a problem for both systems. If a term doesn't appear in the language model used for decoding, then it cannot be predicted. This results in errors such as the slang word "posers" being changed to "posters", the term "fbers" being changed to "fibers" because "facebookers" is not in the lexicon, or the name "terese" being changed to "trees". This is a much more difficult problem to solve because it requires some real-world knowledge or a continually updating language model. Finally the system still occasionally has trouble with homophones (replacing "too" with "two", for example) or abbreviations that could reasonably correspond to multiple words, such as "n" for "and" or "in". In these cases, we expect better reranking and better language models to help.

5.3.4. Score models for pruning only

Because the optimal weights found in Section 5.3.2 were so low, we wanted to see what contribution our model scores have toward reranking the candidates to find the best choice of translation. The system under test is used to generate the possible hypotheses for translation; however, we only use the language model for decoding instead of combining the LM score with our system score (*e.g.*, setting $\beta = 0$ in Eq. (2)). In this way, our system is only used to prune the number of words the LM must consider but does not contribute to the score used to rank these hypotheses.

The results from these tests are shown in Table 14. Although the differences are slight, the changes we see from Table 10 are all in the downward direction. There is still a large improvement over the LM-only baseline. The abbreviation model is thus necessary for the system.

We then test this method at the sentence level. As the trends we have seen before still hold here with regard to the top-*N* list, in the interest of space we present results only for the top-1 hypotheses generated by decoding. Sentence level results are shown in Table 15.

It is interesting to note that the effects of this method are manifested quite differently in the two models. We see that the CRF model takes a significant hit in performance for normalizing the abbreviations compared to when the score model is used. However, we also see that this method significantly decreases the false positive rate, allowing the top-1 sentence accuracy and final WER to improve. The MT model, on the other hand, does not significantly decrease performance on the abbreviations, but performs worse in terms of sentence accuracy and false positive rate.

Table 15
Sentence-level decoding results when our systems are used for pruning only.

		CRF (DEL)	MT (DEL)	MT (SW)
Message Tests (Abbrs.)	Unigram	38.63	59.85	62.36
	Bigram	46.95	67.26	68.15
	Trigram	48.78	68.73	69.22
Message Tests (Sentences)	Unigram	5.98	6.57	7.52
	Bigram	18.25	24.89	25.10
	Trigram	23.53	31.89	31.64
False Positives [WER]	Orig. WER	9.49	9.49	11.6
	Unigram	13.97 [18.6]	19.82 [21.7]	19.72 [21.8]
	Bigram	6.35 [10.8]	8.39 [10.7]	8.10 [10.9]
	Trigram	4.92 [9.3]	6.53 [8.8]	6.33 [9.2]

5.3.5. Discussion

A consistent improvement is found when using a bigram LM for decoding rather than the unigram LM when all other conditions are equal. The improvement of trigram LM over bigram LM is much less, and occasionally we find a slight decrease in performance. For this reason, we feel that we need not test LMs of order greater than three.

Overall, we can see that the MT system always performs better than the CRF system, even when tested on solely the deletion abbreviations for which the CRF was designed. This is probably because of the background character-level language model used by Moses. Recall that we created that LM by using a large amount of data from the Edinburgh Twitter Corpus. In this way, the MT system has access to a much larger source of (unlabeled, non-parallel) data than the CRF model has.

When examining the corrected abbreviations and errors produced by each system, we find that there are slight differences between the two methods. For this reason, we attempt to combine the two systems into one system in hopes of achieving a higher performance than both single systems.

5.4. System combinations

For a given test case (i.e., an abbreviation a), each system $i \in 1, 2$ generates a set of word hypotheses, $\mathbf{H}_1$ and $\mathbf{H}_2$ respectively. Each hypothesis w has score for system i: $S_i(w, a)$. For the combined system, we need to determine the word hypotheses (set $\mathbf{H}_3$) and their scores $S_3(w, a) = f(S_1(w, a), S_2(w, a))$. The following describes the methods we used to combine the information from the two systems.

1 **Weighted average.** In this method, the word candidates are the union of the two systems: $\mathbf{H}_3 = \mathbf{H}_1 \cup \mathbf{H}_2$. For word candidates that appear in only one system, we keep its scores; for a word appearing in both systems, we take the weighted average of the two scores, that is:

$$S_3(w,a) = \begin{cases} S_1(w,a) & \text{if } w \in H_1 - H_2 \\ S_2(w,a) & \text{if } w \in H_2 - H_1 \\ \alpha S_1(w,a) + \beta S_2(w,a) & \text{if } w \in H_1 \cap H_2 \end{cases}$$

where α and β are determined empirically on the development set.

2 **Take highest.** The set H_3 is similar to the method above in that we form it using the superset of $\mathbf{H}_1$ and $\mathbf{H}_2$. The difference is that for the words that fall in the intersection, we take the highest score: $S_3(w, a) = max(S_1(w, a), S_2(w, a))$ if $w \in \mathrm{H}_{\mathbf{1}} \cap \mathrm{H}_{\mathbf{2}}$.

3 **System preference.** The set H_3 is once again the superset, but to determine the score for a word in the intersection, we define a preferred system such that its scores are always used. Let I be the preferred system (1 or 2), then $S_3(w, a) = S_I(w, a)$ if $w \in \mathrm{H}_{\mathbf{1}} \cap \mathrm{H}_{\mathbf{2}}$.

Table 16
Results for system combinations.

		System under test					
		MT	1	2	3a	3b	4
Context tests	Unigram	64.19	66.09	64.09	63.23	63.30	41.26
	Bigram	69.69	70.64	70.44	70.26	70.16	64.07
	Trigram	70.44	71.61	71.39	71.32	71.32	64.91
	Found	77.73	80.10	80.10	80.10	80.10	69.16
Message tests (Abbrs.)	Unigram	62.63	64.34	64.53	63.98	59.26	52.60
	Bigram	66.56	67.90	67.56	67.43	65.42	54.84
	Trigram	67.90	69.40	69.09	69.09	68.01	55.50
Incorrect changes	Unigram	4.07	10.01	12.34	17.26	10.31	4.07
	Bigram	3.50	4.88	5.00	7.44	4.87	3.56
	Trigram	3.44	4.40	4.43	5.92	4.40	3.47
Final WER	Unigram	7.9	13.0	23.9	19.4	13.8	9.1
	Bigram	7.0	8.0	8.2	10.4	8.3	8.4
	Trigram	6.8	7.4	7.5	8.8	7.6	8.3
Message tests (Sents.)	Unigram	31.36	17.02	13.42	8.59	15.04	24.79
	Bigram	36.55	32.43	32.15	26.65	31.36	27.64
	Trigram	37.98	36.00	35.92	32.19	35.48	28.39

4 **Case-specific knowledge.** This is specifically for cases where one of the systems being tested is the deletion-based system. In this example, let $\mathbf{H}_1$ be the set generated by the deletion-based system and $\mathbf{H}_2$ be generated by some other system. For each candidate, we use the knowledge of whether it is actually a deletion abbreviation or not. For those that are deletion abbreviations, if $\mathbf{H}_1 \neq \emptyset$, we use the prediction from H_1 and S_1. If $\mathbf{H}_1 = \emptyset$ or the abbreviation is formed by a method other than deletions, we use the prediction from $\mathbf{H}_2$ and $\mathbf{S}_2$.

Since the CRF system performs worse than MT, this test may seem counter intuitive. Remember, however, that the CRF method has high precision despite its low recall. Therefore, it may achieve poor results not because it produces incorrect translations, but rather because it has no guess at all for an abbreviation and thus leaves it as-is (which is also considered incorrect). For this reason, we hypothesized that when the CRF system has a guess it will be correct, and when it has no guess the MT system can help.

We performed the small context and message level tests on the combinations above using the methods described in Section 5.3 and show results in Table 16. We show only the top-1 results and the upper bounds in the interest of space. The system numbers correspond to their numbers in the list above; 3a refers to preferring the CRF system and 3b refers to preferring the MT system. For comparison, the various MT results are repeated here.

During development, we first run the tests using only trigram context on the development set and optimize the top-1 accuracy. As before, we set $\alpha = 1$ and vary β from 0.01 to 100. Once again, lower weights for β give better performance. Generally, 0.1 was selected as the best weight, although 0.5 and 0.1 were occasionally chosen as well.

The averaging system results are shown for the best performing system (MT system weight = 1 and deletion system weight = 0.25). This set-up performed slightly better than the MT system during context tests. While that advantage holds for abbreviations during full message decoding, the higher false positive rate causes an overall decrease in performance.

Excepting the case-specific method, the combinations perform well on abbreviations but suffer from a high false positive rate during sentence decoding. The oracle method performs somewhat worse on abbreviations, but surprisingly performs competitively with respect to other metrics.

5.5. Comparison with past work

Although there has been very little work on this task until quite recently, multiple studies have been done using the small 303-term dataset first used by Choudhury et al. (2007). For this reason, we run our two top performing systems (MT alone, and the average weight combination) on this small data set. We also use the Jazzy spell-checker as

Table 17
System comparison on the 303-term test set from Choudhury et al. (2007).

	Top-1	Top-3	Top-10	Top-20
Jazzy (Idzelis, 2005)	49.86	53.13	54.78	55.44
Choudhury et al. (2007)	59.9	–	84.3	88.7
Cook and Stevenson (2009)	59.4	–	83.8	87.8
Liu et al. (2011)[a]	58.09	70.96	–	–
Liu et al. (2011)[b]	62.05	75.91	–	–
MT	60.39	74.58	75.57	75.57
Average	62.70	77.55	79.53	79.53

[a] LetterTran.
[b] LetterTran +Jazzy.

a baseline. Because this dataset has no context information we are unable to perform the LM-only baseline or decode using a LM with order greater than 1.

The CRF scores used in the average combination use all features and are rescored using a length model trained on all our data. Our score models are combined with the unigram language model in order to form a prediction. The AM scores are given a weight of 0.1 and we fix the LM weight at 1.

Table 17 lists system comparisons. The results shown for other systems (except Jazzy) are those reported in their respective papers; we did not re-implement their systems. We see that both our systems perform comparably on this dataset to Liu et al. (2011) when combined with Jazzy. The averaging system has a slight advantage, but with only 303 items it is probably not significant. Although we outperform both Choudhury et al. (2007) and Cook and Stevenson (2009) in top-1, they outperform both of our systems at top-10 and top-20. One of our goals is thus to obtain better coverage.

6. Conclusions and future work

In this paper, we have provided an extensive comparison of two abbreviation models for normalizing abbreviations found in informal text. Both models yield improvements over two baselines – using a language model alone for decoding and a state-of-the-art spell-checking algorithm – even when using the score models with no context. With context and an LM, we significantly outperform both baselines. Our MT model vastly outperforms our CRF model, even on the deletion-type abbreviations for which the CRF model was designed. It will be interesting future work to use our data to compare our MT model to the extended CRF model used by Liu et al. (2011).

Increasing the order of the language model yields a fairly large increase in performance from a unigram to a bigram model, with a small increase in performance when moving to trigram models. When combining the abbreviation and language models, giving the language model much higher priority gives the best performance on abbreviations. This explains why using our systems to prune the hypotheses checked by the language model does not cause a large decrease in performance on abbreviations. However, the pruning only method leads to worse overall results due to false positives.

We also tested combinations of the two models to create a single system that performs better than either model alone. During tests on abbreviations with context words in standard form, we see a slight increase in performance when taking a weighted average of the two models (with MT weighted higher than CRF). However, a higher false positive rate means this does not translate to overall improvement at the message level.

False positives are a major area for future work. One heuristic is to leave a token as-is if we find it in a dictionary. Preliminary tests using this heuristic improved precision but greatly decreased recall because many abbreviations are themselves words, (*e.g.*. “cat” for “category” or “no” for “number”). Additionally, it is difficult to find an appropriate dictionary. The dictionaries we tried are either missing some common words or contain many acronyms and chat slang, defeating the purpose of the heuristic. Alternatively, we are considering using the dictionary not as a definitive source of whether a token should be expanded, but rather if the word is found we do not expand it *unless* the LM score, AM score, or their combination is “high enough”. This work is still in progress, but preliminary results look promising.

In addition, the MT model still needs some improvement. The correct translations did not appear in the 20-best list generated for over 20% of the abbreviations in this data. We hope that optimizing the parameters in Moses or moving

to a factored model will help decrease this percentage. We also have the potential to improve our system by reranking the results that Moses generates using a length model ot other metric as with the CRF system.

Finally, we believe that the machine translation method is mainly language independent and are expanding our work to languages other than English, with preliminary results on Spanish *tweets*. We are also investigating its use for deromanization of non-Latin-script languages that have been informally transliterated for use in social media applications.

Acknowledgements

Thanks to Justin Schneider and Duc Le for their work in implementing the message selection procedure for annotations. Thanks also to Paul Cook for providing his abbreviation type labels for the SMS test set so that we could perform comparison experiments.

This work is partly supported by DARPA under Contract No. HR0011-12-C-0016. Any opinions expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

Aw, A., Zhang, M., Xian, J., Su, J., 2006. A phrase-based statistical model for SMS text normalization. In: Proceedings of COLING/ACL, Sydney, Australia, pp. 33–40.

Bangalore, S., Murdock, V., Riccardi, G., 2002. Bootstrapping bilingual data using consensus translation for a multilingual instant messaging system. In: 19th International Conference on Computational Linguistics, Taipei, Taiwan, pp. 1–7.

Bartlett, S., Kondrak, G., Cherry, C., 2008. Automatic syllabification with structured SVMs for letter-to-phoneme conversion. In: Proceedings of ACL, Columbus, OH, pp. 568–576.

Beaufort, R., Roekhaut, S., Cougnon, L., Fairon, C., 2010. A hybrid rule/model-based finite-state framework for normalizing SMS messages. In: Proceedings of ACL, Uppsala, Sweden, pp. 770–779.

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

Contractor, D., Faruquie, T.A., Subramaniam, L.V., 2010. Unsupervised cleansing of noisy text. In: Proceedings of COLING: Posters, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 189–196.

Cook, P., Stevenson, S., 2009. An unsupervised model for text message normalization. In: NAACL HLT Workshop on Computational Approaches to Linguistic Creativity, Boulder, CO, pp. 71–78.

Fairon, C., Paumier, S., 2006. A translated corpus of 30,000 French SMS. In: Proceedings of LREC, Genoa, Italy.

Han, B., Baldwin, T., 2011. Lexical normalisation of short text messages: Makn sens a #twitter. In: Proceedings of ACL.

Han, B., Cook, P., Baldwin, T., 2012. Automatically constructing and normalisation dicitonary for microblogs. In: Proceedings of the Joint Conference on EMNLP and CoNLL, Jeju Island, Korea.

Henríquez, Q.C.A., Hernández, H.A., 2009. A ngram-based statistical machine translation approach for text normalization on chat-speak style communications. In: Content Analysis for the Web 2.0, Madrid, Spain.

How, Y., Kan, M.Y., 2005. Optimizing predictive text entry for short message service on mobile phones, in: Human Computer Interfaces International (HCII). Available from http://www.comp.nus.edu.sg/ kanmy/papers/hcii05.pdf.

Idzelis, M., 2005. Jazzy: The java open source spell checker. http://jazzy.sourceforge.net/

Kobus, C., Yvon, F., Damnati, G., 2008. Normalizing SMS: are two metaphors better than one? In: Proceedings of COLING, Manchester, UK, pp. 441–448.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., Herbst, E., 2007. Moses: open source toolkit for statistical machine translation. In: Proceedings of ACL Interactive Poster and Demonstration Sessions, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 177–180.

Kukich, K., 1992. Technique for automatically correcting words in text. ACM Computing Surveys 24, 377–439.

Li, C., Liu, Y., 2012. Normalization of text messages using character- and phone-based machine translation approaches. In: Proceedings of InterSpeech, Portland, Oregon, USA.

Liu, F., Weng, F., Jiang, X., 2012. A broad-coveraged normalization system for social media language. In: Proceedings of ACL, Jeju Island, Korea.

Liu, F., Weng, F., Wang, B., Liu, Y., 2011. Insertion, deletion, or substitution? Normalizing text messages without pre-categorization nor supervision. In: Proceedings of ACL-HLT.

Pakhomov, S., 2002. Semi-supervised maximum entropy based approach to acronym and abbreviation normalization in medical texts. In: Proceedings of ACL, Association for Computational Linguistics, Philadelphia, Pennsylvania, pp. 160–167.

Pennell, D., Liu, Y., 2010. Normalization of text messages for text-to-speech. In: Proceedings of ICASSP, Dallas, Texas, USA, pp. 4842–4845.

Pennell, D., Liu, Y., 2011. A character-level machine translation approach for normalization of SMS abbreviations. In: Proceedings of IJCNLP, Chiang Mai, Thailand.

Pennell, D., Liu, Y., 2011. Toward text message normalization: modeling abbreviation generation. In: Proceedings of ICASSP, Prague, Czech Republic.

Petrovic, S., Osborne, M., Lavrenko, V., 2010. The Edinburgh Twitter corpus. In: NAACL-HLT Workshop on Computational Linguistics in a World of Social Media, Los Angeles, California, USA, pp. 25–26.

Raghunathan, K., Krawczyk, S., 2009. CS224N: Investigating SMS Text Normalization using Statistical Machine Translation. Technical Report.

Sproat, R., Black, A., Chen, S., Kumar, S., Ostendorf, M., Richards, C., 2001. Normalization of non-standard words. Computer Speech and Language 15, 287–333.

Stolcke, A., 2002. SRILM-an extensible language modeling toolkit. In: Proceedings of ICLSP, pp. 901–904.

Toutanova, K., Moore, R.C., 2002. Pronunciation modeling for improved spelling correction. In: Proceedings of ACL, Philadelphia, Pennsylvania, pp. 144–151.

Whitelaw, C., Hutchinson, B., Chung, G., Ellis, G., 2009. Using the web for language independent spellchecking and autocorrection, in. In: Proceedings of EMNLP, pp. 890–899.

Willis, T., Pain, H., Trewin, S., Clark, S., 2002. Informing flexible abbreviation expansion for users with motor disabilities. Computers Helping People with Special Needs, 359–371.

Wong, W., Liu, W., Bennamoun, M., 2006. Integrated scoring for spelling error correction, abbreviation expansion and case restoration in dirty text. In: Proceedings of the 5th Australasian Conference on Data Mining and Analytics, Sydney, Australia, pp. 83–89.

Yang, D., Pan, Y., Furui, S., 2009. Automatic Chinese abbreviation generation using conditional random field. In: Proceedings of NAACL-HLT, Boulder, Colorado, pp. 273–276.