A Unified Tagging Approach to Text Normalization

Conghui Zhu
Harbin Institute of Technology
Harbin, China
chzhu@mtlab.hit.edu.cn

Jie Tang
Department of Computer Science
Tsinghua University, China
jietang@tsinghua.edu.cn

Hang Li
Microsoft Research Asia
Beijing, China
hangli@microsoft.com

Hwee Tou Ng
Department of Computer Science
National University of Singapore, Singapore
nght@comp.nus.edu.sg

Tiejun Zhao
Harbin Institute of Technology
Harbin, China
tjzhao@mtlab.hit.edu.cn

Abstract

This paper addresses the issue of text normalization, an important yet often overlooked problem in natural language processing. By text normalization, we mean converting 'informally inputted' text into the canonical form, by eliminating 'noises' in the text *and* detecting paragraph and sentence boundaries in the text. Previously, text normalization issues were often undertaken in an ad-hoc fashion or studied separately. This paper first gives a formalization of the entire problem. It then proposes a unified tagging approach to perform the task using Conditional Random Fields (CRF). The paper shows that with the introduction of a small set of tags, most of the text normalization tasks can be performed within the approach. The accuracy of the proposed method is high, because the subtasks of normalization are interdependent and should be performed together. Experimental results on email data cleaning show that the proposed method significantly outperforms the approach of using cascaded models and that of employing independent models.

1 Introduction

More and more 'informally inputted' text data becomes available to natural language processing, such as raw text data in emails, newsgroups, forums, and blogs. Consequently, how to effectively process the data and make it suitable for natural language processing becomes a challenging issue. This is because informally inputted text data is usually very noisy and is not properly segmented. For example, it may contain extra line breaks, extra spaces, and extra punctuation marks; and it may contain words badly cased. Moreover, the boundaries between paragraphs and the boundaries between sentences are not clear.

We have examined 5,000 randomly collected emails and found that 98.4% of the emails contain noises (based on the definition in Section 5.1).

In order to perform high quality natural language processing, it is necessary to perform 'normalization' on informally inputted data first, specifically, to remove extra line breaks, segment the text into paragraphs, add missing spaces and missing punctuation marks, eliminate extra spaces and extra punctuation marks, delete unnecessary tokens, correct misused punctuation marks, restore badly cased words, correct misspelled words, and identify sentence boundaries.

Traditionally, text normalization is viewed as an engineering issue and is conducted in a more or less ad-hoc manner. For example, it is done by using rules or machine learning models at different levels. In natural language processing, several issues of text normalization were studied, but were only done separately.

This paper aims to conduct a thorough investigation on the issue. First, it gives a formalization of

Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 688–695,
Prague, Czech Republic, June 2007.

the problem; specifically, it defines the subtasks of the problem. Next, it proposes a unified approach to the whole task on the basis of tagging. Specifically, it takes the problem as that of assigning tags to the input texts, with a tag representing deletion, preservation, or replacement of a token. As the tagging model, it employs Conditional Random Fields (CRF). The unified model can achieve better performances in text normalization, because the subtasks of text normalization are often interdependent. Furthermore, there is no need to define specialized models and features to conduct different types of cleaning; all the cleaning processes have been formalized and conducted as assignments of the three types of tags.

Experimental results indicate that our method significantly outperforms the methods using cascaded models or independent models on normalization. Our experiments also indicate that with the use of the tags defined, we can conduct most of the text normalization in the unified framework.

Our contributions in this paper include: (a) formalization of the text normalization problem, (b) proposal of a unified tagging approach, and (c) empirical verification of the effectiveness of the proposed approach.

The rest of the paper is organized as follows. In Section 2, we introduce related work. In Section 3, we formalize the text normalization problem. In Section 4, we explain our approach to the problem and in Section 5 we give the experimental results. We conclude the paper in Section 6.

2 Related Work

Text normalization is usually viewed as an engineering issue and is addressed in an ad-hoc manner. Much of the previous work focuses on processing texts in clean form, not texts in informal form. Also, prior work mostly focuses on processing one type or a small number of types of errors, whereas this paper deals with many different types of errors.

Clark (2003) has investigated the problem of preprocessing noisy texts for natural language processing. He proposes identifying token boundaries and sentence boundaries, restoring cases of words, and correcting misspelled words by using a source channel model.

Minkov et al. (2005) have investigated the problem of named entity recognition in informally inputted texts. They propose improving the performance of personal name recognition in emails using two machine-learning based methods: Conditional Random Fields and Perceptron for learning HMMs. See also (Carvalho and Cohen, 2004).

Tang et al. (2005) propose a cascaded approach for email data cleaning by employing Support Vector Machines and rules. Their method can detect email headers, signatures, program codes, and extra line breaks in emails. See also (Wong et al., 2007).

Palmer and Hearst (1997) propose using a Neural Network model to determine whether a period in a sentence is the ending mark of the sentence, an abbreviation, or both. See also (Mikheev, 2000; Mikheev, 2002).

Lita et al. (2003) propose employing a language modeling approach to address the case restoration problem. They define four classes for word casing: all letters in lower case, first letter in uppercase, all letters in upper case, and mixed case, and formalize the problem as assigning class labels to words in natural language texts. Mikheev (2002) proposes using not only local information but also global information in a document in case restoration.

Spelling error correction can be formalized as a classification problem. Golding and Roth (1996) propose using the Winnow algorithm to address the issue. The problem can also be formalized as that of data conversion using the source channel model. The source model can be built as an n-gram language model and the channel model can be constructed with confusing words measured by edit distance. Brill and Moore, Church and Gale, and Mayes et al. have developed different techniques for confusing words calculation (Brill and Moore, 2000; Church and Gale, 1991; Mays et al., 1991).

Sproat et al. (1999) have investigated normalization of non-standard words in texts, including numbers, abbreviations, dates, currency amounts, and acronyms. They propose a taxonomy of non-standard words and apply n-gram language models, decision trees, and weighted finite-state transducers to the normalization.

3 Text Normalization

In this paper we define text normalization at three levels: paragraph, sentence, and word level. The subtasks at each level are listed in Table 1. For example, at the paragraph level, there are two subtasks: extra line-break deletion and paragraph boundary detection. Similarly, there are six (three) subtasks at the sentence (word) level, as shown in Table 1. Unnecessary token deletion refers to deletion of tokens like '-----' and '====', which are not needed in natural language processing. Note that most of the subtasks conduct 'cleaning' of noises, except paragraph boundary detection and sentence boundary detection.

Level	Task	Percentages of Noises
Paragraph	Extra line break deletion	49.53
	Paragraph boundary detection	
Sentence	Extra space deletion	15.58
	Extra punctuation mark deletion	0.71
	Missing space insertion	1.55
	Missing punctuation mark insertion	3.85
	Misused punctuation mark correction	0.64
	Sentence boundary detection	
Word	Case restoration	15.04
	Unnecessary token deletion	9.69
	Misspelled word correction	3.41

Table 1. Text Normalization Subtasks

As a result of text normalization, a text is segmented into paragraphs; each paragraph is segmented into sentences with clear boundaries; and each word is converted into the canonical form. After normalization, most of the natural language processing tasks can be performed, for example, part-of-speech tagging and parsing.

We have manually cleaned up some email data (cf., Section 5) and found that nearly all the noises can be eliminated by performing the subtasks defined above. Table 1 gives the statistics.

1. i'm thinking about buying a pocket
2. pc   device for my wife this christmas,.
3. the worry that i have is that she won't
4. be able to sync it to her outlook express
5. contacts…

Figure 1. An example of informal text

I'm thinking about buying a Pocket PC device for my wife this Christmas.// The worry that I have is that she won't be able to sync it to her Outlook Express contacts.//

Figure 2. Normalized text

Figure 1 shows an example of informally inputted text data. It includes many typical noises. From line 1 to line 4, there are four extra line breaks at the end of each line. In line 2, there is an extra comma after the word 'Christmas'. The first word in each sentence and the proper nouns (e.g., 'Pocket PC' and 'Outlook Express') should be capitalized. The extra spaces between the words 'PC' and 'device' should be removed. At the end of line 2, the line break should be removed and a space is needed after the period. The text should be segmented into two sentences.

Figure 2 shows an ideal output of text normalization on the input text in Figure 1. All the noises in Figure 1 have been cleaned and paragraph and sentence endings have been identified.

We must note that dependencies (sometimes even strong dependencies) exist between different types of noises. For example, word case restoration needs help from sentence boundary detection, and vice versa. An ideal normalization method should consider processing all the tasks together.

4 A Unified Tagging Approach

4.1 Process

In this paper, we formalize text normalization as a tagging problem and employ a unified approach to perform the task (no matter whether the processing is at paragraph level, sentence level, or word level).

There are two steps in the method: preprocessing and tagging. In preprocessing, (A) we separate the text into paragraphs (i.e., sequences of tokens), (B) we determine tokens in the paragraphs, and (C) we assign possible tags to each token. The tokens form the basic units and the paragraphs form the sequences of units in the tagging problem. In tagging, given a sequence of units, we determine the most likely corresponding sequence of tags by using a trained tagging model. In this paper, as the tagging model, we make use of CRF.

Next we describe the steps (A)-(C) in detail and explain why our method can accomplish many of the normalization subtasks in Table 1.

(A). We separate the text into paragraphs by taking two or more consecutive line breaks as the endings of paragraphs.

(B). We identify tokens by using heuristics. There are five types of tokens: 'standard word', 'non-standard word', punctuation mark, space, and line break. Standard words are words in natural language. Non-standard words include several general 'special words' (Sproat et al., 1999), email address, IP address, URL, date, number, money, percentage, unnecessary tokens (e.g., '===' and

'###'), etc. We identify non-standard words by using regular expressions. Punctuation marks include period, question mark, and exclamation mark. Words and punctuation marks are separated into different tokens if they are joined together. Natural spaces and line breaks are also regarded as tokens.

(C). We assign tags to each token based on the type of the token. Table 2 summarizes the types of tags defined.

Token Type	Tag	Description
Line break	PRV	Preserve line break
	RPA	Replace line break by space
	DEL	Delete line break
Space	PRV	Preserve space
	DEL	Delete space
Punctuation mark	PSB	Preserve punctuation mark and view it as sentence ending
	PRV	Preserve punctuation mark without viewing it as sentence ending
	DEL	Delete punctuation mark
Word	AUC	Make all characters in uppercase
	ALC	Make all characters in lowercase
	FUC	Make the first character in uppercase
	AMC	Make characters in mixed case
Special token	PRV	Preserve the special token
	DEL	Delete the special token

Table 2. Types of tags

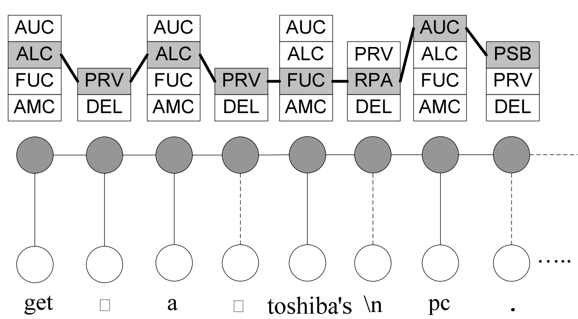

Figure 3. An example of tagging

Figure 3 shows an example of the tagging process. (The symbol '□' indicates a space). In the figure, a white circle denotes a token and a gray circle denotes a tag. Each token can be assigned several possible tags.

Using the tags, we can perform most of the text normalization processing (conducting seven types of subtasks defined in Table 1 and cleaning 90.55% of the noises).

In this paper, we do not conduct three subtasks, although we could do them in principle. These include missing space insertion, missing punctuation mark insertion, and misspelled word correction. In our email data, it corresponds to 8.81% of the noises. Adding tags for insertions would increase the search space dramatically. We did not do that due to computation consideration. Misspelled word correction can be done in the same framework easily. We did not do that in this work, because the percentage of misspelling in the data is small.

We do not conduct misused punctuation mark correction as well (e.g., correcting '.' with '?'). It consists of 0.64% of the noises in the email data. To handle it, one might need to parse the sentences.

4.2 CRF Model

We employ Conditional Random Fields (CRF) as the tagging model. CRF is a conditional probability distribution of a sequence of tags given a sequence of tokens, represented as $P(Y|X)$, where X denotes the token sequence and Y the tag sequence (Lafferty et al., 2001).

In tagging, the CRF model is used to find the sequence of tags Y^* having the highest likelihood $Y^* = \max_Y P(Y|X)$, with an efficient algorithm (the Viterbi algorithm).

In training, the CRF model is built with labeled data and by means of an iterative algorithm based on Maximum Likelihood Estimation.

Transition Features
$y_{i-1}=y', y_i=y$
$y_{i-1}=y', y_i=y, w_i=w$
$y_{i-1}=y', y_i=y, t_i=t$

State Features	
$w_i=w, y_i=y$	$t_i=t, y_i=y$
$w_{i-1}=w, y_i=y$	$t_{i-1}=t, y_i=y$
$w_{i-2}=w, y_i=y$	$t_{i-2}=t, y_i=y$
$w_{i-3}=w, y_i=y$	$t_{i-3}=t, y_i=y$
$w_{i-4}=w, y_i=y$	$t_{i-4}=t, y_i=y$
$w_{i+1}=w, y_i=y$	$t_{i+1}=t, y_i=y$
$w_{i+2}=w, y_i=y$	$t_{i+2}=t, y_i=y$
$w_{i+3}=w, y_i=y$	$t_{i+3}=t, y_i=y$
$w_{i+4}=w, y_i=y$	$t_{i+4}=t, y_i=y$
$w_{i-1}=w', w_i=w, y_i=y$	$t_{i-2}=t'', t_{i-1}=t', y_i=y$
$w_{i+1}=w', w_i=w, y_i=y$	$t_{i-1}=t', t_i=t, y_i=y$
	$t_i=t, t_{i+1}=t', y_i=y$
	$t_{i+1}=t', t_{i+2}=t'', y_i=y$
	$t_{i-2}=t'', t_{i-1}=t', t_i=t, y_i=y$
	$t_{i-1}=t'', t_i=t, t_{i+1}=t', y_i=y$
	$t_i=t, t_{i+1}=t', t_{i+2}=t'', y_i=y$

Table 3. Features used in the unified CRF model

4.3 Features

Two sets of features are defined in the CRF model: transition features and state features. Table 3 shows the features used in the model.

Suppose that at position i in token sequence x, w_i is the token, t_i the type of token (see Table 2), and y_i the possible tag. Binary features are defined as described in Table 3. For example, the transition feature $y_{i-1}=y'$, $y_i=y$ implies that if the current tag is y and the previous tag is y', then the feature value is true; otherwise false. The state feature $w_i=w$, $y_i=y$ implies that if the current token is w and the current label is y, then the feature value is true; otherwise false. In our experiments, an actual feature might be the word at position 5 is 'PC' and the current tag is AUC. In total, 4,168,723 features were used in our experiments.

4.4 Baseline Methods

We can consider two baseline methods based on previous work, namely cascaded and independent approaches. The independent approach performs text normalization with several passes on the text. All of the processes take the raw text as input and output the normalized/cleaned result independently. The cascaded approach also performs normalization in several passes on the text. Each process carries out cleaning/normalization from the output of the previous process.

4.5 Advantages

Our method offers some advantages.

(1) As indicated, the text normalization tasks are interdependent. The cascaded approach or the independent approach cannot simultaneously perform the tasks. In contrast, our method can effectively overcome the drawback by employing a unified framework and achieve more accurate performances.

(2) There are many specific types of errors one must correct in text normalization. As shown in Figure 1, there exist four types of errors with each type having several correction results. If one defines a specialized model or rule to handle each of the cases, the number of needed models will be extremely large and thus the text normalization processing will be impractical. In contrast, our method naturally formalizes all the tasks as assignments of different types of tags and trains a unified model to tackle all the problems at once.

5 Experimental Results

5.1 Experiment Setting

Data Sets

We used email data in our experiments. We randomly chose in total 5,000 posts (i.e., emails) from 12 newsgroups. DC, Ontology, NLP, and ML are from newsgroups at Google (http://groups-beta.google.com/groups). Jena is a newsgroup at Yahoo (http://groups.yahoo.com/group/jena-dev). Weka is a newsgroup at Waikato University (https://list.scms.waikato.ac.nz). Protégé and OWL are from a project at Stanford University (http://protege.stanford.edu/). Mobility, WinServer, Windows, and PSS are email collections from a company.

Five human annotators conducted normalization on the emails. A spec was created to guide the annotation process. All the errors in the emails were labeled and corrected. For disagreements in the annotation, we conducted "majority voting". For example, extra line breaks, extra spaces, and extra punctuation marks in the emails were labeled. Unnecessary tokens were deleted. Missing spaces and missing punctuation marks were added and marked. Mistakenly cased words, misspelled words, and misused punctuation marks were corrected. Furthermore, paragraph boundaries and sentence boundaries were also marked. The noises fell into the categories defined in Table 1.

Table 4 shows the statistics in the data sets. From the table, we can see that a large number of noises (41,407) exist in the emails. We can also see that the major noise types are extra line breaks, extra spaces, casing errors, and unnecessary tokens.

In the experiments, we conducted evaluations in terms of precision, recall, F1-measure, and accuracy (for definitions of the measures, see for example (van Rijsbergen, 1979; Lita et al., 2003)).

Implementation of Baseline Methods

We used the cascaded approach and the independent approach as baselines.

For the baseline methods, we defined several basic prediction subtasks: extra line break detection, extra space detection, extra punctuation mark detection, sentence boundary detection, unnecessary token detection, and case restoration. We compared the performances of our method with those of the baseline methods on the subtasks.

Data Set	Number of Email	Number of Noises	Extra Line Break	Extra Space	Extra Punc.	Missing Space	Missing Punc.	Casing Error	Spelling Error	Misused Punc.	Unnecessary Token	Number of Paragraph Boundary	Number of Sentence Boundary
DC	100	702	476	31	8	3	24	53	14	2	91	457	291
Ontology	100	2,731	2,132	24	3	10	68	205	79	15	195	677	1,132
NLP	60	861	623	12	1	3	23	135	13	2	49	244	296
ML	40	980	868	17	0	2	13	12	7	0	61	240	589
Jena	700	5,833	3,066	117	42	38	234	888	288	59	1,101	2,999	1,836
Weka	200	1,721	886	44	0	30	37	295	77	13	339	699	602
Protégé	700	3,306	1,770	127	48	151	136	552	116	9	397	1,645	1,035
OWL	300	1,232	680	43	24	47	41	152	44	3	198	578	424
Mobility	400	2,296	1,292	64	22	35	87	495	92	8	201	891	892
WinServer	400	3,487	2,029	59	26	57	142	822	121	21	210	1,232	1,151
Windows	1,000	9,293	3,416	3,056	60	116	348	1,309	291	67	630	3,581	2,742
PSS	1,000	8,965	3,348	2,880	59	153	296	1,331	276	66	556	3,411	2,590
Total	5,000	41,407	20,586	6,474	293	645	1,449	6,249	1,418	265	4,028	16,654	13,580

Table 4. Statistics on data sets

For the case restoration subtask (processing on token sequence), we employed the TrueCasing method (Lita et al., 2003). The method estimates a tri-gram language model using a large data corpus with correctly cased words and then makes use of the model in case restoration. We also employed Conditional Random Fields to perform case restoration, for comparison purposes. The CRF based casing method estimates a conditional probabilistic model using the same data and the same tags defined in TrueCasing.

For unnecessary token deletion, we used rules as follows. If a token consists of non-ASCII characters or consecutive duplicate characters, such as '===', then we identify it as an unnecessary token.

For each of the other subtasks, we exploited the classification approach. For example, in extra line break detection, we made use of a classification model to identify whether or not a line break is a paragraph ending. We employed Support Vector Machines (SVM) as the classification model (Vapnik, 1998). In the classification model we utilized the same features as those in our unified model (see Table 3 for details).

In the cascaded approach, the prediction tasks are performed in sequence, where the output of each task becomes the input of each immediately following task. The order of the prediction tasks is: (1) Extra line break detection: Is a line break a paragraph ending? It then separates the text into paragraphs using the remaining line breaks. (2) Extra space detection: Is a space an extra space? (3) Extra punctuation mark detection: Is a punctuation mark a noise? (4) Sentence boundary detection: Is a punctuation mark a sentence boundary? (5) Unnecessary token deletion: Is a token an unnecessary token? (6) Case restoration. Each of steps (1) to (4) uses a classification model (SVM), step (5) uses rules, whereas step (6) uses either a language model (TrueCasing) or a CRF model (CRF).

In the independent approach, we perform the prediction tasks independently. When there is a conflict between the outcomes of two classifiers, we adopt the result of the latter classifier, as determined by the order of classifiers in the cascaded approach.

To test how dependencies between different types of noises affect the performance of normalization, we also conducted experiments using the unified model by removing the transition features.

Implementation of Our Method

In the implementation of our method, we used the tool CRF++, available at http://chasen.org/~taku /software/CRF++/. We made use of all the default settings of the tool in the experiments.

5.2 Text Normalization Experiments

Results

We evaluated the performances of our method (Unified) and the baseline methods (Cascaded and Independent) on the 12 data sets. Table 5 shows the five-fold cross-validation results. Our method outperforms the two baseline methods.

Table 6 shows the overall performances of text normalization by our method and the two baseline methods. We see that our method outperforms the two baseline methods. It can also be seen that the performance of the unified method decreases when removing the transition features (Unified w/o Transition Features).

We conducted sign tests for each subtask on the results, which indicate that all the improvements of Unified over Cascaded and Independent are statistically significant (p << 0.01).

Detection Task		Prec.	Rec.	F1	Acc.
Extra Line Break	Independent	95.16	91.52	93.30	93.81
	Cascaded	95.16	91.52	93.30	93.81
	Unified	93.87	93.63	**93.75**	**94.53**
Extra Space	Independent	91.85	94.64	93.22	99.87
	Cascaded	94.54	94.56	94.55	99.89
	Unified	95.17	93.98	**94.57**	**99.90**
Extra Punctuation Mark	Independent	88.63	82.69	85.56	99.66
	Cascaded	87.17	85.37	86.26	99.66
	Unified	90.94	84.84	**87.78**	**99.71**
Sentence Boundary	Independent	98.46	99.62	99.04	98.36
	Cascaded	98.55	99.20	98.87	98.08
	Unified	98.76	99.61	**99.18**	**98.61**
Unnecessary Token	Independent	72.51	100.0	84.06	84.27
	Cascaded	72.51	100.0	84.06	84.27
	Unified	98.06	95.47	**96.75**	**96.18**
Case Restoration (TrueCasing)	Independent	27.32	87.44	41.63	96.22
	Cascaded	28.04	88.21	42.55	96.35
Case Restoration (CRF)	Independent	84.96	62.79	72.21	99.01
	Cascaded	85.85	63.99	73.33	99.07
	Unified	86.65	67.09	**75.63**	**99.21**

Table 5. Performances of text normalization (%)

Text Normalization	Prec.	Rec.	F1	Acc.
Independent (TrueCasing)	69.54	91.33	78.96	97.90
Independent (CRF)	85.05	92.52	88.63	98.91
Cascaded (TrueCasing)	70.29	92.07	79.72	97.88
Cascaded (CRF)	85.06	92.70	88.72	98.92
Unified w/o Transition Features	86.03	93.45	89.59	99.01
Unified	**86.46**	**93.92**	**90.04**	**99.05**

Table 6. Performances of text normalization (%)

Discussions

Our method outperforms the independent method and the cascaded method in all the subtasks, especially in the subtasks that have strong dependencies with each other, for example, sentence boundary detection, extra punctuation mark detection, and case restoration.

The cascaded method suffered from ignorance of the dependencies between the subtasks. For example, there were 3,314 cases in which sentence boundary detection needs to use the results of extra line break detection, extra punctuation mark detection, and case restoration. However, in the cascaded method, sentence boundary detection is conducted after extra punctuation mark detection and before case restoration, and thus it cannot leverage the results of case restoration. Furthermore, errors of extra punctuation mark detection can lead to errors in sentence boundary detection.

The independent method also cannot make use of dependencies across different subtasks, because it conducts all the subtasks from the raw input data. This is why for detection of extra space, extra punctuation mark, and casing error, the independent method cannot perform as well as our method.

Our method benefits from the ability of modeling dependencies between subtasks. We see from Table 6 that by leveraging the dependencies, our method can outperform the method without using dependencies (Unified w/o Transition Features) by 0.62% in terms of F1-measure.

Here we use the example in Figure 1 to show the advantage of our method compared with the independent and the cascaded methods. With normalization by the independent method, we obtain:

I’m thinking about buying a pocket PC   device for my wife this Christmas, The worry that I have is that she won’t be able to sync it to her outlook express contacts.//

With normalization by the cascaded method, we obtain:

I’m thinking about buying a pocket PC device for my wife this Christmas, the worry that I have is that she won’t be able to sync it to her outlook express contacts.//

With normalization by our method, we obtain:

I’m thinking about buying a Pocket PC device for my wife this Christmas.// The worry that I have is that she won’t be able to sync it to her Outlook Express contacts.//

The independent method can correctly deal with some of the errors. For instance, it can capitalize the first word in the first and the third line, remove extra periods in the fifth line, and remove the four extra line breaks. However, it mistakenly removes the period in the second line and it cannot restore the cases of some words, for example ‘pocket’ and ‘outlook express’.

In the cascaded method, each process carries out cleaning/normalization from the output of the previous process and thus can make use of the cleaned/normalized results from the previous process. However, errors in the previous processes will also propagate to the later processes. For example, the cascaded method mistakenly removes the period in the second line. The error allows case restoration to make the error of keeping the word ‘the’ in lower case.

TrueCasing-based methods for case restoration suffer from low precision (27.32% by Independent and 28.04% by Cascaded), although their recalls are high (87.44% and 88.21% respectively). There are two reasons: 1) About 10% of the errors in Cascaded are due to errors of sentence boundary detection and extra line break detection in previous steps; 2) The two baselines tend to restore cases of words to the forms having higher probabilities in the data set and cannot take advantage of the dependencies with the other normalization subtasks. For example, 'outlook' was restored to first letter capitalized in both 'Outlook Express' and 'a pleasant outlook'. Our method can take advantage of the dependencies with other subtasks and thus correct 85.01% of the errors that the two baseline methods cannot handle. Cascaded and Independent methods employing CRF for case restoration improve the accuracies somewhat. However, they are still inferior to our method.

Although we have conducted error analysis on the results given by our method, we omit the details here due to space limitation and will report them in a future expanded version of this paper.

We also compared the speed of our method with those of the independent and cascaded methods. We tested the three methods on a computer with two 2.8G Dual-Core CPUs and three Gigabyte memory. On average, it needs about 5 hours for training the normalization models using our method and 25 seconds for tagging in the cross-validation experiments. The independent and the cascaded methods (with TrueCasing) require less time for training (about 2 minutes and 3 minutes respectively) and for tagging (several seconds). This indicates that the efficiency of our method still needs improvement.

6 Conclusion

In this paper, we have investigated the problem of text normalization, an important issue for natural language processing. We have first defined the problem as a task consisting of noise elimination and boundary detection subtasks. We have then proposed a unified tagging approach to perform the task, specifically to treat text normalization as assigning tags representing deletion, preservation, or replacement of the tokens in the text. Experiments show that our approach significantly outperforms the two baseline methods for text normalization.

References

E. Brill and R. C. Moore. 2000. An Improved Error Model for Noisy Channel Spelling Correction, *Proc. of ACL 2000.*

V. R. Carvalho and W. W. Cohen. 2004. Learning to Extract Signature and Reply Lines from Email, *Proc. of CEAS 2004.*

K. Church and W. Gale. 1991. Probability Scoring for Spelling Correction, *Statistics and Computing*, Vol. 1.

A. Clark. 2003. Pre-processing Very Noisy Text, *Proc. of Workshop on Shallow Processing of Large Corpora*.

A. R. Golding and D. Roth. 1996. Applying Winnow to Context-Sensitive Spelling Correction, *Proc. of ICML'1996*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data, *Proc. of ICML 2001*.

L. V. Lita, A. Ittycheriah, S. Roukos, and N. Kambhatla. 2003. tRuEcasIng, *Proc. of ACL 2003*.

E. Mays, F. J. Damerau, and R. L. Mercer. 1991. Context Based Spelling Correction, *Information Processing and Management*, Vol. 27, 1991.

A. Mikheev. 2000. Document Centered Approach to Text Normalization, *Proc. SIGIR 2000.*

A. Mikheev. 2002. Periods, Capitalized Words, etc. *Computational Linguistics*, Vol. 28, 2002.

E. Minkov, R. C. Wang, and W. W. Cohen. 2005. Extracting Personal Names from Email: Applying Named Entity Recognition to Informal Text, *Proc. of EMNLP/HLT-2005*.

D. D. Palmer and M. A. Hearst. 1997. Adaptive Multilingual Sentence Boundary Disambiguation, *Computational Linguistics*, Vol. 23.

C.J. van Rijsbergen. 1979. *Information Retrieval*. Butterworths, London.

R. Sproat, A. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards. 1999. Normalization of non-standard words, *WS'99 Final Report.* http://www.clsp.jhu.edu/ws99/projects/normal/.

J. Tang, H. Li, Y. Cao, and Z. Tang. 2005. Email data cleaning, *Proc. of SIGKDD'2005.*

V. Vapnik. 1998. *Statistical Learning Theory*, Springer.

W. Wong, W. Liu, and M. Bennamoun. 2007. Enhanced Integrated Scoring for Cleaning Dirty Texts, *Proc. of IJCAI-2007 Workshop on Analytics for Noisy Unstructured Text Data.*