Email Data Cleaning

Jie Tang
Department of Computer Science
Tsinghua University
12#109, Tsinghua University
Beijing, China, 100084
j-tang02@mails.tsinghua.edu.cn

Hang Li, Yunbo Cao
Microsoft Research Asia
5F Sigma Center
No.49 Zhichun Road, Haidian
Beijing, China, 100080.
{hangli, yucao}@microsoft.com

Zhaohui Tang
Microsoft Corporation
One Microsoft Way
Redmond,
WA, USA, 98052
zhaotang@microsoft.com

ABSTRACT
Addressed in this paper is the issue of 'email data cleaning' for text mining. Many text mining applications need take emails as input. Email data is usually noisy and thus it is necessary to clean it before mining. Several products offer email cleaning features, however, the types of noises that can be eliminated are restricted. Despite the importance of the problem, email cleaning has received little attention in the research community. A thorough and systematic investigation on the issue is thus needed. In this paper, email cleaning is formalized as a problem of *non-text filtering* and *text normalization*. In this way, email cleaning becomes independent from any specific text mining processing. A *cascaded* approach is proposed, which cleans up an email in four passes including non-text filtering, paragraph normalization, sentence normalization, and word normalization. As far as we know, non-text filtering and paragraph normalization have not been investigated previously. Methods for performing the tasks on the basis of Support Vector Machines (SVM) have also been proposed in this paper. Features in the models have been defined. Experimental results indicate that the proposed SVM based methods can significantly outperform the baseline methods for email cleaning. The proposed method has been applied to term extraction, a typical text mining processing. Experimental results show that the accuracy of term extraction can be significantly improved by using the data cleaning method.

Categories and Subject Descriptors
H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - *Information filtering, selection process*

General Terms
Algorithm, Design, Experimentation, Theory.

Keywords
Text Mining, Data Cleaning, Email Processing, Statistical Learning

KDD'05, August 21-24, 2005, Chicago, Illinois, USA.

1. INTRODUCTION
Email is one of the commonest means for communication via text. It is estimated that an average computer user receives 40 to 50 emails per day [8]. Many text mining applications need take emails as inputs, for example, email analysis, email routing, email filtering, email summarization, information extraction from email, and newsgroup analysis.

Unfortunately, Email data can be very noisy. Specifically, it may contain headers, signatures, quotations, and program codes; it may contain extra line breaks, extra spaces, and special character tokens; it may have spaces and periods mistakenly removed; and it may contain words badly cased or non-cased and words misspelled.

In order to achieve high quality email mining, it is necessary to conduct data cleaning at the first step. This is exactly the problem addressed in this paper.

Many text mining products have email data cleaning features. However, the number of noise types that can be processed is limited. In the research community, no previous study has so far sufficiently investigated the problem, to the best of our knowledge. Data cleaning work has been done mainly on structured tabular data, not unstructured text data. In natural language processing, sentence boundary detection, case restoration, spelling error correction, and word normalization have been studied, but usually as separated issues. The methodologies proposed in the previous work can be used in email data cleaning. However, they are not sufficient for removing all the noises.

Three questions arise for email data cleaning: (1) how to formalize the problem (since it involves many different factors at different levels and appears to be very complex); (2) how to solve the problem in a *principled* approach; and (3) how to make an implementation.

(1) We formalize email data cleaning as that of non-text filtering and text normalization. Specifically, email cleaning is defined as a process of eliminating irrelevant non-text data (it includes header, signature, quotation and program code filtering) and transforming relevant text data into canonical form like that in newspaper (it includes paragraph, sentence and word normalization).

(2) We propose to conduct email cleaning in a 'cascaded' fashion. In the approach, we clean up an email by running several passes: first at email body level (non-text filtering), and then at paragraph, sentence, and word levels (text normalization).

(3) It turns out that some of the tasks in the approach can be accomplished with existing methodologies, but some cannot. The tasks of email header detection, signature detection, and program code detection in non-text filtering, and paragraph ending detection in paragraph normalization do not seem to be examined previously. We view the former three tasks as 'reverse information extraction'. We propose a unified statistical learning approach to the tasks, based on SVM (Support Vector Machines). We define features for the models.

We tried to collect data from as many sources as possible for experimentation. In total, 5,459 emails from 14 different sources were gathered. Our experimental results indicate that the proposed SVM based methods perform significantly better than the baseline methods for cleaning. We also applied our method to term extraction. Experimental results indicate that our method can indeed enhance the accuracy of term extraction. We observed 38%-45% improvements on term extraction in terms of F1-measure.

The rest of the paper is organized as follows. In Section 2, we introduce related work. In Section 3, we formalize the problem of email data cleaning. In Section 4, we describe our approach to the problem and in Section 5, we explain one possible implementation. Section 6 gives our experimental results. We make concluding remarks in Section 7.

2. RELATED WORK

2.1 Data Cleaning

Data cleaning is an important area in data mining. Many research efforts have been made so far. However, most of the previous work was focusing on cleaning up of structured data and only a little work was concerned with semi-structured or non-structured data cleaning.

Email Data Cleaning

Several products have email cleaning features. For instance, eClean 2000 is a tool that can clean up emails by removing extra spaces between words, removing extra line breaks between paragraphs, removing email headers, and re-indenting forwarded mails [33]. It conducts email cleaning using rules defined by users.

WinPure ListCleaner Pro is a data cleaning product. It also has an email cleaning module [34]. It can identify inaccurate and duplicated email addresses in a list of email addresses. However, it does not conduct cleaning on email data itself.

To the best of our knowledge, no previous work has been done on email cleaning in the research community.

Web Page Data Cleaning

Considerable efforts have been placed on the cleaning of web pages.

For instance, Yi and Liu [30] define banner ads, navigational guides, and decoration pictures as web page noises. They assign a weight to each block in a web page, where a weight represents the importance (cleanness) of a block. They use, in the weight calculation, the fact that web pages in a site tend to follow fixed layouts and those parts in a page that also appear in many other pages in the site are likely to be noises.

Lin and Ho view the problem of web page cleaning as that of discovering informative contents from web pages [16]. They first partition a page into several blocks on the basis of HTML tags. They next calculate entropy value of each block. Finally, they select the informative blocks by a predefined threshold from the page. See also [14].

Tabular Data Cleaning

Tabular data cleaning is aimed at detecting and removing duplicate information when data is consolidated from different sources. Therefore, tabular data cleaning significantly differs in nature from text data cleaning.

Tabular data cleaning has been investigated at both schema level and instance level. At schema level, the differences in data schemas can be absorbed by schema translation and schema integration. The main problem here is to resolve naming and structural conflicts [23]. At instance level, the main problem is to identify overlapping data. The problem is also referred to as object identification [10], duplicate elimination, or merge/purge problem [13]. See [25] for an overview.

Some products provide tools for tabular data cleaning. For instance, SQL Server 2005 provides a tool for tabular data cleaning called Fuzzy Grouping. The ETL tool performs data cleaning by identifying rows of similar or duplicate data and choosing a canonical row to represent the rows of the data [35].

2.2 Language Processing

Sentence boundary detection, word normalization, case restoration, spelling error correction, and other related issues have been intensively investigated in natural language processing, but usually as separated issues.

Sentence Boundary Detection

Palmer and Hearst, for instance, 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 [22]. They utilize the part of speech probabilities of the tokens surrounding the period as information for the disambiguation. See also [20].

Case Restoration

Lita et al. propose employing a language modeling approach to address the case restoration problem [17]. They define four classes for word casing: all lower case, first letter upper case, all letters upper case, and mixed case, and formalize the problem as that of assigning the class labels to words in natural language texts. They then make use of an n-gram model to calculate the probability scores of the assignments.

Mikheev proposes making use of not only local information but also global information in a document in case restoration [20]. See also [5, 9].

Spelling Error Correction

Spelling error correction can be formalized as a word sense disambiguation problem. The goal then becomes to select a correct word from a set of confusion words, e.g., {to, too, two} in a specific context. For example, Golding and Roth propose using statistical learning methods to address the issue [12].

The problem can also be formalized as that of data conversion using the noise channel model from Information Theory. 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. For example, Mayes et al., Church and Gale, Brill and Moore have developed techniques for the confusing words calculation [2, 4, 18].

Word Normalization

Sproat et al. have investigated normalization of non-standard words in texts, including numbers, abbreviations, dates, currency amounts, and acronyms [27]. They define a taxonomy of non-standard words and apply n-gram language models, decision trees, and weighted finite-state transducers to the normalization.

2.3 Information Extraction

In information extraction, given a sequence of instances, we identify and pull out a sub sequence of the input that represents information we are interested in. Hidden Markov Model [11], Maximum Entropy Model [1, 3], Maximum Entropy Markov Model [19], Support Vector Machines [7], Conditional Random Field [15], and Voted Perceptron [6] are widely used information extraction models.

Information extraction has been applied, for instance, to part-of-speech tagging [26], named entity recognition [32] and table extraction [21, 24, 29].

3. CLEANING AS FILTERING AND NORMALIZATION

Mining from emails is an important subject in text mining. A large number of applications can be considered, for example, analysis of trends in emails, automatic routing of email messages, automatic filtering of spam emails, summarization of emails, information extraction from emails, and analysis of trends in newsgroup discussions (newsgroup articles are usually emails).

```
1.  On Mon, 23 Dec 2002 13:39:42 -0500, "Brendon"
2.  <brendon@nospamitology.net> wrote:

3.  NETSVC.EXE from the NTReskit. Or use the
4.  psexec from
5.  sysinternals.com. this lets you run
6.  commands remotely - for example net stop 'service'.

7.  --
8.  ----------------------------------------
9.  Best Regards
10. Brendon
11.
12. Delighting our customers is our top priority. We welcome your comments and
13. suggestions about how we can improve the support we provide to you.
14. ----------------------------------------

15. >>-----Original Message-----
16. >>"Jack" <jehandy@verizon.net> wrote in message
17. >>news:00a201c2aab2$12154680$d5f82ecf@TK2MSFTNGXA12...
18. >> Is there a command line util that would allow me to
19. >> shutdown services on a remote machine via a batch file?

20. >>Best Regards
21. >>Jack
```

Figure 1. Example of email message

```
1. NETSVC.EXE from the NTReskit. Or use the psexec from sysinternals.com.
2. This lets you run commands remotely - for example net stop 'service'.
```

Figure 2. Cleaned email message

Unfortunately, emails are usually very noisy and simply applying text mining tools to them, which are usually not designed for mining from noisy data, may not bring good results. We examined the quality of the 5,459 emails and found that surprisingly 98.4% of the emails have this or that type of noise *for text mining* (based on the definition of clean email described below).

Figure 1 shows an example email which includes many typical noises (or errors) for text mining. Lines 1 and 2 are a header; lines from 7 to 14 are a signature; and a quotation lies from line 15 to line 21. All of them are supposed to be irrelevant to text mining. Only lines from 3 to 6 are actual text content. However, the text is not in canonical form. It is mistakenly separated by extra line breaks. The word "this" in line 5 is also not capitalized.

Figure 2 shows an ideal output of cleaning on the email in Figure 1. Within it, the non-text parts (header, signature and quotation) have been removed. The text has been normalized. Specifically, the extra line breaks have been eliminated. The case of word "this" has been correctly restored.

In this paper, we formalize the email cleaning problem as that of non-text data filtering and text data normalization. By 'filtering' of an email we mean a process of removing the parts in the email which are not needed for text mining, and by 'normalization' of an email we mean a process of converting the parts necessary for text mining into texts in canonical form (like a newspaper style text).

Header, signature, quotation (in forwarded message or replied message), program code, and table are usually irrelevant for mining, and thus should be identified and removed (in a particular text mining application, however, we can retain some of them when necessary). On the other hand, text and list are needed for text mining and thus should be retained.

In a text in canonical form, paragraphs are separated by line breaks; sentences have punctuation marks (period, question mark, exclamation mark, colon, ellipsis); the first words in the sentences are capitalized; and all the words are correctly cased and spelled.

Usually natural language processing and text mining systems are designed for processing texts in canonical form. A desirable consequence of conducting cleaning in this way is that we can significantly enhance the modularity of text mining.

Here, we only consider handling emails in plain text format, i.e., non-structured data. We do not take into consideration of emails in other formats such as HTML and Rich Format Text. There are two reasons: all the other formats can be reduced to plain text (with the format information lost, however) and usually many emails for text mining (and data mining) are stored in databases as plain texts.

4. CASCADED APPROACH

We perform email cleaning in four passes of processing: non-text filtering, paragraph normalization, sentence normalization, and word normalization. Figure 3 shows the flow.

The input is an email message. In non-text filtering, we identify the existing header, signature, quotation, and program code in the email and remove the identified blocks. In paragraph normalization, we identify extra line breaks and remove them. In

sentence normalization, we figure out whether a period, a question mark, or an exclamation mark is a real sentence-ending. If so, we take it as a sentence boundary. Moreover, we remove non-words including non-ASCII words, tokens containing many special symbols, and lengthy tokens, and take their locations as sentence boundaries as well (a sentence obtained in this way is not necessarily a natural sentence). In word normalization, we conduct case restoration on badly cased words.

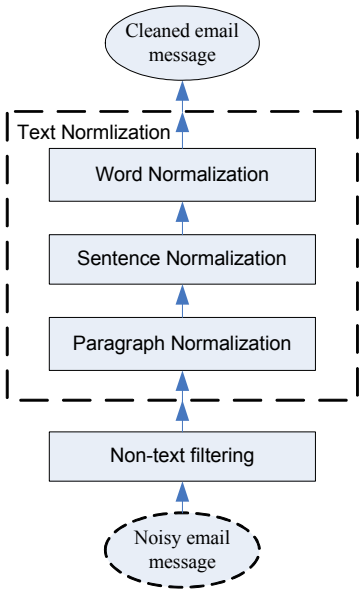

Figure 3. Flow of email data cleaning

We note that it is reasonable to conduct cleaning as described above. Removing noisy blocks first are preferable, because such blocks are not needed in the other processing. Normalizing text from paragraph to sentence and then to word is desirable, because there are dependencies between the processes. Word normalization (e.g., case restoration) needs sentence beginning information. Paragraph normalization (e.g., paragraph ending information) helps sentence normalization.

We should also filter out other noisy blocks like tables. In this paper, we confine ourselves to the removal of the noisy blocks described above (header, signature, and program code), because we have observed only a few other block types (tables) available in our data. (0.6% of emails in the 14 data sets have other types). We should also conduct spelling error correction in word normalization. However, we will leave this to future work, because spelling errors are less common than casing errors in emails. (93.6% of the word level errors are casing errors.)

5. IMPLEMENTATION

We consider one implementation of the cascaded approach. We employ a unified machine learning approach in non-text filtering and paragraph normalization. Furthermore, we utilize rules in the sentence normalization and word normalization. The former two issues have not been investigated previously and are the main focus of our work. The latter two issues have been intensively studied in the literature as explained.

5.1 Outline

The input is an email message. The implementation carries out cleaning in the following steps.

(1) Preprocessing. It uses patterns to recognize 'special words', including email address, IP address, URL, date, file directory, Date (e.g. 02-16-2005), number (e.g. 5.42), money (e.g. $100), percentage (e.g. 92.86%), words containing special symbols (e.g. C#, .NET, .doc, Dr.). It also uses patterns to recognize bullets in list items (e.g.: (1), b), etc.)

(2) Non-text filtering. It detects the header and signature (if there exist) in the email by using a classification model. It then eliminates the identified blocks. It next detects program codes (if there exist) in the email with the same approach and removes the identified blocks. Finally, it filters out quotations using hard-coded rules. It views lines starting with special characters (e.g. >, |, >>) as quotations. After this step, only relevant text data remains. The step relies on header detection, signature detection, and program code detection.

(3) Paragraph normalization. It identifies whether or not each line break is a paragraph ending by using a classification model. If not, it removes the line break. It also forcibly removes consecutive (redundant) line breaks between paragraphs into a single line-break. As a result, the text is segmented into paragraphs. The step is mainly based on paragraph ending detection.

(4) Sentence normalization. It determines whether each punctuation mark (i.e., period, exclamation mark, and question mark) is sentence ending by utilizing rules. If there is no space after an identified sentence ending, it adds a space there. It also removes redundant symbols (including space, exclamation mark, question mark, and period) at the sentence ending. Furthermore, it eliminates noisy tokens (e.g. non-ASCII characters, tokens containing many special symbols, and lengthy tokens) and views the positions as sentence endings (this is because a sentence can rarely be across such tokens). As a result, each paragraph is segmented into sentences.

(5) Word normalization. It conducts case restoration on badly cased words using rules and a dictionary.

5.2 Classification Model

We make use of Support Vector Machines (SVM) as the classification model [28].

Let us first consider a two class classification problem. Let $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ be a training data set, in which x_i denotes an instance (a feature vector) and $y_i \in \{-1,+1\}$ denotes a classification label. In learning, one attempts to find an optimal separating hyper-plane that maximally separates the two classes of training instances (more precisely, maximizes the margin between the two classes of instances). The hyper-plane corresponds to a classifier (linear SVM). It is theoretically guaranteed that the linear classifier obtained in this way has small generalization errors. Linear SVM can be further extended into non-linear SVMs by using kernel functions such as Gaussian and polynomial kernels.

We use SVM-light, which is available at http://svmlight.joachims.org/. We choose polynomial kernel, because our preliminary experimental results show that it works

best for our current task. We use the default values for the parameters in SVM-light. When there are more than two classes, we adopt the "one class versus all others" approach, i.e., take one class as positive and the other classes as negative.

5.3 Header and Signature Detection

5.3.1 Processing

Header detection and signature detection are similar problems. We view both of them as 'reverse information extraction'. Hereafter, we take header as example in our explanation. The learning based header detection consists of two stages: training and detection.

In detection, we identify whether or not a line is the start line of a header, and whether or not a line is the end line of a header using two SVM models. We next view the lines between the identified start line and the end line as header.

In training, we construct the two SVM models that can detect the start line and the end line, respectively. In the SVM models, we view a line in an email as an instance. For each instance, we define a set of features and assign a label. The label represents whether the line is start, end, or neither. We use the labeled data to train the SVM models in advance.

It seems reasonable to take lines as instances for non-text filtering. We randomly picked up 104,538 lines from the 5,459 emails and found that 98.37% of the lines are either text or non-text (header, signature, program code, etc). It is really rare to have a mix of text and non-text in a line.

The key issue here is how to define *features* for effectively performing the cleaning task.

5.3.2 Features in Header Detection Models

The features are used in both the header-start and header-end SVM models.

Position Feature: The feature represents whether the current line is the first line in the email.

Positive Word Features: The features represent whether or not the current line begins with words like "From:", "Re:", "In article", and "In message", contains words such as "original message" and "Fwd:", or ends with words like "wrote:" and "said:".

Negative Word Features: The features respectively represent whether or not the current line contains words like "Hi", "dear", "thank you", and "best regards". The words are usually used in greeting and should not be included in a header.

Number of Words Feature: The feature stands for the number of words in the current line.

Person Name Feature: The feature represents whether or not the current line contains a person name (first name or last name).

Ending Character Features: The features respectively represent whether or not the current line ends with colon, semicolon, quotation mark, question mark, exclamation mark or suspension points. (The first line of a header is likely to end with characters like quotation mark, but is less likely to end with characters like colon or semicolon.)

Special Pattern Features: In the preprocessing step, the special words have already been recognized. Each of the features represents whether or not the current line contains one type of special words. Positive types include email address and date. Negative types include money and percentage.

Number of Line Breaks Feature: The feature represents how many line breaks exist before the current line.

The features above are also defined similarly for the previous line and the next line.

5.3.3 Features in Signature Detection Model

The features are used in both the signature-start and signature-end SVM models.

Position Features: The two features are defined to represent whether or not the current line is the first line or the last line in the email.

Positive Word Features: The features represents whether or not the current line contains positive words like "Best Regards", "Thanks", "Sincerely" and "Good luck".

Number of Words Features: One of the two features stands for the number of words in the current line. The first line of a signature usually contains a few words, such as the author's name or words like "Best Regards", "Thanks". The other feature stands for the number of words in a dictionary.

Person Name Feature: The feature represents whether or not the current line contains a person name (first name or last name). A signature is likely to begin with the author's name.

Ending Character Features: The features respectively represent whether or not the current line ends with a punctuation mark like colon, semicolon, quotation mark, question mark, exclamation mark and suspension points. (A signature is less likely to end with punctuation marks like colon or semicolon.)

Special Symbol Pattern Features: The features respectively indicate whether the line contains consecutive special symbols such as: "--------", "=======", "******". Such patterns can be frequently found in signatures.

Case Features: The features represent the cases of the tokens in the current line. They indicate whether the tokens are all in upper-case, all in lower-case, all capitalized or only the first token is capitalized.

Number of Line Breaks Feature: The feature represents how many line breaks exist before the current line.

The features above are also defined similarly for the previous line and the next line.

5.4 Program Code Detection

5.4.1 Processing

Program code detection is similar to header and signature detection. It can also be viewed as a 'reverse information extraction' problem. The detection is performed by identifying the start line and the end line of a program code using SVMs. A recognized program code is then removed. Again, utilizing effective features in the SVM models is the key to a successful detection.

5.4.2 Features in Program Code Detection Model

The following features are used in both the code-start and code-end models.

Position Feature: The feature represents the position of the current line.

Declaration Keyword Feature: The feature represents whether or not the current line starts with one of the keywords, including "string", "char", "double", "dim", "typedef struct", "#include", "import", "#define", "#undef", "#ifdef", and "#endif".

Statement Keyword Features: The four features represent

-whether or not the current line contains patterns like "i++";

-whether or not the current line contains keywords like "if", "else if", "switch", and "case";

-whether or not the current line contains keywords like "while", "do{", "for", and "foreach";

-whether or not the current line contains keywords like "goto", "continue;", "next;", "break;", "last;" and "return;".

Equation Pattern Features: The four features are defined for equations as follows:

-whether or not the current line contains an equation pattern like "=", "<=" and "<<=";

-whether or not the current line contains an equation pattern like "a=b+/*-c;";

-whether or not the current line contains an equation pattern like "a=B(bb,cc);";

-whether or not the current line contains an equation pattern like "a=b;".

Function Pattern Feature: The feature represents whether or not the current line contains function pattern, e.g., pattern covering "fread(pbBuffer,1, LOCK_SIZE, hSrcFile);".

Function Definition Features: The two features represent whether or not the current line starts with "sub" or "function", and whether or not it starts with "end function" or "end sub".

Bracket Features: The four features represent whether or not the line starts with or ends with "{" and whether or not the line starts with or ends with "}".

Percentage of Real Words Feature: The feature represents the percentage of 'real' words in the line that can be found in a dictionary.

Ending Character Features: Program code lines usually end with a semicolon ";", but seldom end with a question mark "?" or an exclamation mark "!". The two features are defined to represent whether the current line ends with a semicolon and whether the line ends with a question mark or an exclamation mark.

Number of Line Breaks Feature: The feature represents how many line breaks exist before the current line.

The features above are also defined similarly for the previous line and the next line.

5.5 Paragraph Ending Detection

5.5.1 Processing

A text may contain many line breaks. We identify whether each of line break is a paragraph ending or an extra-line-break. We view this problem as that of classification and employ a SVM model to perform the task. If a line break is recognized as an extra-line-break, then we remove it; otherwise, we retain it. In this way, we segment the text into normalized paragraphs.

In the SVM model, we view a line as an instance. For each instance, we define a set of features and assign a label. The label represents whether the line break in the line is unnecessary. We use the labeled data to train the SVM model in advance. The lines having extra line breaks are positive instances, and the other lines are negative instances.

5.5.2 Features in Paragraph Ending Detection Model

The following features are defined in the paragraph-ending model.

Position Features: The two features represent whether or not the current line is the first line and whether or not it is the last line.

Greeting Word Features: The features respectively represent whether or not the line contains greeting words like "Hi" and "Dear". (In such case, the line break should not be removed).

Ending Character Features: The features respectively represent whether or not the current line ends with a punctuation mark like colon, semicolon, quotation mark, question mark, exclamation mark and suspension points.

Case Features: The two features represent whether the current line ends with a word in lower case letters and whether or not the next line starts with a word in lower case letters.

Bullet Features: The features represent whether or not the next line is one kind of bullet of a list item like "1." and "a)". (In such cases, the line break should be retained)

Number of Line Breaks Feature: The feature represents how many line breaks exist after the current line.

The features above are also defined similarly for the next line.

6. EXPERIMENTAL RESULTS

6.1 Data Sets and Evaluation Measures

6.1.1 Data sets

We tried to collect emails for experimentation from as many sources as possible. We randomly chose in total 5,459 emails from 14 sources and created 14 data sets. DC, Ontology, NLP and ML and J2EE 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 from 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, PSS and BR are email collections or newsgroups at Microsoft.

Human annotators conducted annotation on all the emails. Specifically, headers, signatures, quotations, program codes, etc, in the emails were labeled. Paragraph boundaries were identified.

Sentence boundaries were marked. Incorrectly-cased words were modified and spelling errors were corrected.

Table 1 shows the statistics on the data sets. The columns respectively represent data set, number of emails, and percentages of emails containing headers, signatures, program codes, and text *only*.

Table 1. Statistics on data sets (%)

No	Data Set	Number	Header	Signature	Code	Text Only
1	DC	100	100.0	87.0	15.0	0.0
2	Ontology	100	100.0	77.0	2.0	0.0
3	NLP	60	100.0	88.3	0.0	0.0
4	ML	40	100.0	97.5	5.0	0.0
5	Jena	700	99.6	97.0	38.0	0.0
6	Weka	200	99.5	97.5	17.0	0.5
7	Protégé	500	28.0	82.2	3.2	16.8
8	OWL	500	38.4	93.2	4.2	4.8
9	Mobility	400	44.0	74.5	0.0	18.3
10	WinServer	400	44.9	67.2	1.25	22.1
11	Windows	1000	47.6	65.3	0.7	21.8
12	PSS	1000	49.2	66.8	1.0	20.8
13	BR	310	49.5	64.3	0.0	24.4
14	J2EE	255	100.0	56.1	9.4	0
	Average	5459	58.5	76.0	7.2	13.8

From table 1, we see that a large portion of emails contain headers and signatures. In three of the data sets, more than 15% of emails contain program codes.

We also made statistics on other types of errors. In summary, 73.2% of the emails need paragraph normalization, 85.4% of the emails need sentence normalization, and 47.1% of the emails need case restoration. Only 7.4% of the emails contain at least one spelling error. *Only 1.6% of the emails are absolutely clean.* We omit the details due to space limitation.

6.1.2 Evaluation measures

In all the experiments on detection and extraction, we conducted evaluations in terms of precision, recall and F1-measure. The evaluation measures are defined as follows:

Precision: $P = A / (A + B)$

Recall: $R = A / (A + C)$

F1-measure: $F1 = 2PR / (P + R)$

where, A, B, C and D denote number of instances.

Table 2. Contingence table on results of detection and extraction

	Is Target	Is Not Target
Found	A	B
Non Found	C	D

In all evaluations, we view a correction or extraction made by humans as a 'target'. If a method can find the target, we say that it makes a correct decision; otherwise, we say that it makes a mistake. Precision, recall, and F1-measure are calculated on the basis of the result. For header, signature, quotation, program code, and paragraph ending detections, we conduct evaluation at line level. For the other tasks, we perform evaluation at word level.

6.1.3 Baseline methods

For header detection and paragraph ending detection, we used eClean [33] as baselines. There were default rules in eClean for header detection, and we made some extensions on the rules based on the features of our SVM models for header detection.

For signature detection, we used as a baseline the most useful features in our SVM models for signature detection. For program code detection, it was hard to find a baseline. (There is no feature in eClean for signature and program code detection.)

The rule for header detection is as follow. If a line begins with a pattern of "From:", "Newsgroups:", "To:", "Sent:", "Cc:", "Subject:", "sender:", "news:", "In article", "In Message", or "Re:", contains a pattern of "---Original Message---" or "Fwd:", or ends with a pattern of "wrote:", "said:", or "wrote in message", then identify the line as a line in the header.

The rule for signature detection is as follow. If a line is in the last five lines of the email, and begins with words of "Best Regards", "Regard", "Best Wishes", "Thanks", "Hope this help", "Cheers", "Sincerely", "Yours", or "Gratefully", then identify the line and all the following lines as signature.

The rule for paragraph ending detection is as follows. If a line does not end with a punctuation mark of "!" and ".", and the first word of the next line is neither capitalized nor a number, then remove the line break between the two lines.

6.2 Data Cleaning

6.2.1 Experiment

We evaluated the performances of our non-text filtering methods (header, signature, quotation and program code filtering) and our text normalization methods (paragraph normalization, sentence normalization, and word normalization) on the *first 12* data sets and used the remaining two data sets for other experiment.

We conducted the experiment in the following way. First, we conducted header and signature filtering. We also performed quotation filtering. After that, we conducted program code filtering. Then, we conducted paragraph normalization. Finally, we conducted sentence normalization and word normalization. In each step, we evaluated the data cleaning results in terms of precision, recall, and F1-measure. We also made comparisons with the baseline methods as described above.

Table 3 shows the five-fold *cross-validation* results on the 12 data sets. In the table, Header, Signature, Quotation, Program, Paragraph, Sentence, and Word denote the cleaning steps described above.

We see that our methods can achieve high performances in all the tasks. For Header, Signature, and Paragraph, our methods significantly outperform the baselines. We conducted sign tests on

Table 3. Performances of non-text filtering and text normalization (%)

Detection Task		Precision	Recall	F1-Measure
Header	Our Method	98.97	96.57	97.76
	Baseline	99.81	60.55	75.37
Signature	Our Method	91.35	88.47	89.88
	Baseline	88.54	23.68	37.36
Quotation		98.18	92.01	95.00
Program		92.97	72.17	81.26
Paragraph	Our Method	85.53	97.65	91.19
	Baseline	63.55	98.13	77.15
Sentence		94.93	93.91	94.42
Word		93.23	89.51	91.33

the results. The *p* values are much smaller than 0.01, indicating that the improvements are statistically significant.

For Header, Signature, and Paragraph, both precision and recall of our methods are high. For Program, precision of our method is high, but recall needs further improvement.

6.2.2 Discussions

(1) **High precision.** Precisions of our methods range from 85.53% to 98.97%. It indicates that the use of features and rules described above is very effective for email data cleaning.

(2) **Improvements over baseline methods.** The rule-based header detection baseline suffers from low recall (only 60.55%), although its precision reaches 99.81%. This is due to a low coverage of the rules. Usually it is difficult to identify headers by using rules. For signature detection, recall of the baseline is also low (only 23.68%) due to a similar reason. For paragraph ending (conversely extra-line-break) detection, the baseline of using rules cannot work well either. This is because the task is hard to be performed with rules.

(3) **Error analysis.** We conducted error analysis on the results of our method.

For header detection, there were mainly three types of errors. More than 43% of the errors were from headers having specific patterns. About 40% of the errors occurred when there were extra line breaks in headers. Furthermore, there were about 5% errors from headers containing non-ASCII characters.

For signature detection, about 38% of the errors were from signature start line identification and 62% of the errors were from end line identification. Sometimes, signatures are hard to detect. The characteristics of signatures can vary largely depending on authors. Signatures are sometimes similar to the main texts.

In program code detection, recall is only 72.17%. This is because it is hard to find general patterns for the task.

For paragraph detection, 61% of the errors were due to incorrect elimination of necessary line breaks and 39% of the errors were results of overlooking unnecessary line breaks. About 69% of the former errors happened before lists. This is because sometimes bullets in lists were missing. As for the latter errors, about 51.6% were due to errors of program code detection at the previous step.

For sentence ending detection, about 33% of the errors were misclassification of periods in acronyms. 22% of the errors were failures in recovering missing periods.

For word normalization, the errors fell into three categories. First, more than half of the errors were due to out of vocabulary words. Second, one fourth of the errors were due to ambiguities of words. For example, “Outlook 2003” vs. “a pleasant outlook”. We need implement a more sophisticated mechanism for dealing with the problem. Third, the erroneous results of sentence ending detection at the previous step also affected the results of word normalization.

6.3 Term Extraction

6.3.1 Experiment

To evaluate the effectiveness of our method, we applied it to term extraction. Term extraction is a task in which base noun phrases are extracted from documents. The extracted terms can be used as features of documents and thus term extraction is a fundamental processing in text mining.

We evaluated term extraction results before and after conducting email data cleaning.

The two data sets BR and J2EE were used in the experiment. Terms in the two data sets were also manually labeled. In term extraction, we employed a tool based on technologies proposed in [30]. The tool first conducts part of speech tagging with a Hidden Markov Model. It then extracts base noun phrases as terms using a Finite State Machine.

We carried out data cleaning in the same way as that in the experiment in Section 6.2. The SVM models were trained with the first 12 data sets. After each step of data cleaning, we performed term extraction. Table 4 shows the results of term extraction. In the table, Original Data denotes the results of extraction from the original noisy data and Header, Signature, Quotation, Program, and Paragraph denote the results of extraction after the cleaning steps described above. Base denotes the baseline method for cleaning and SVM denotes our proposed method.

We see that significant improvements can be obtained on term extraction precision (+74.2% on BR and +42.4% on J2EE). At each cleaning step, we observe improvements on precision. The results indicate that our method of data cleaning works well and each of the steps in our method is needed.

From the results we see that our method is significantly better than the baseline. We conducted sign tests on the results. The *p*

values on the two data sets are much smaller than 0.01, indicating that the improvements are statistically significant.

Table 5 shows a comparison between the term extraction results on manually cleaned data and those on the automatically cleaned data using our method. The former results, thus, are upper bounds for our method. We see that the results obtained with our method are close to the upper bounds. We again confirm the effectiveness of our data cleaning method.

Table 4. Performances of term extraction on original data and cleaned data (%)

Data Set	Term Extraction	Precision		Recall		F1-Measure	
		Base	SVM	Base	SVM	Base	SVM
BR	Original Data	50.0		85.2		63.0	
	+Header	60.1 +20.0	64.1 +28.0	85.2 +0.0	85.2 +0.0	70.5 +11.8	73.1 +16.0
	+Signature	61.9 +23.7	79.8 +59.5	84.8 -0.4	84.3 -1.0	71.6 +13.6	82.0 +30.1
	+Quotation	61.9 +23.7	79.8 +59.5	84.8 -0.4	84.3 -1.0	71.6 +13.6	82.0 +30.1
	+Program	61.9 +23.7	79.8 +59.5	84.8 -0.4	84.3 -1.0	71.6 +13.6	82.0 +30.1
	+Paragraph	67.5 +34.8	87.2 +74.2	91.3 +7.2	90.6 +6.4	77.6 +23.1	88.9 +41.0
J2EE	Original Data	38.9		68.7		49.7	
	+Header	42.6 +9.5	43.3 +11.4	68.3 -0.6	68.3 -0.6	52.5 +5.6	53.0 +6.7
	+Quotation	48.0 +23.3	48.6 +25.0	68.3 -0.6	68.3 -0.6	56.4 +13.4	56.8 +14.4
	+Signature	48.6 +25.0	51.2 +32.9	67.9 -1.2	67.7 -1.5	56.7 +14.1	58.6 +18.0
	+Program Code	48.6 +25.0	52.9 +36.0	67.9 -1.2	67.6 -1.6	56.7 +14.1	59.3 +19.5
	+Paragraph	50.3 +29.4	55.4 +42.4	70.2 +2.2	70.3 +2.3	58.6 +18.0	62.0 +24.7

Table 5. Performances of term extraction on clean data and data cleaned by our method (%)

Data set	Term Extraction	Precision	Recall	F1-Measure
BR	Clean	88.71	91.87	90.26
	Our Method	87.16	90.63	88.86
J2EE	Clean	61.68	71.05	66.03
	Our Method	55.40	70.27	61.96

6.3.2 Discussions

(1) The improvements on precision are significant (ranging from +42.4% to +74.2%). The results are consistent with those obtained in the experiment in Section 6.2.

(2) Non-text filtering (header, signature, quotation and program code detection) makes major contributions to the improvements. The improvements on precision are +59.5% and +36.0%.

(3) Non-text filtering induces small drops in recall sometimes. This is because filtering has certain errors and can mistakenly discard some text data.

(4) Paragraph normalization (paragraph ending detection) is useful. It can concatenate broken lines and makes possible the extraction of noun phrases that lie across two broken lines. As a result, it improves not only precision (+14.7% and +6.4%), but also recall (+6.4% and +2.3%).

(5) Term extraction after cleaning with our method outperforms that with the baseline methods. On BR, the improvement by our methods is +41.0% and the improvement by the baseline is only +23.1%. On J2EE, the improvement by our methods is 24.7% and that by the baseline is 18.0%.

(6) It is difficult to accurately extract terms from J2EE, even after manual cleaning. This is because the text contains many out-of-vocabulary words and thus it is hard to extract terms from the data.

7. CONCLUSION

In this paper, we have investigated the problem of email data cleaning. We have defined the problem as that of non-text filtering and text normalization. We have proposed a cascaded approach to the task. Using Support Vector Machines, we have been able to make an implementation of the approach. Experimental results show that our approach can significantly outperform baseline methods for email data cleaning. When applying it to term extraction from emails, we observe a significant improvement on extraction accuracy.

As future work, we plan to make further improvement on the accuracy of each step. We also want to apply the cleaning method to other text mining applications.

8. REFERENCES

[1] A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. A Maximum Entropy Approach to Natural Language Processing. Computational Linguistics, 22:39-71, 1996.

[2] E. Brill and R. C. Moore. An Improved Error Model for Noisy Channel Spelling Correction. In Proc. of the 38th Annual Meeting of the ACL, 2000, pages 286–293.

[3] H. L. Chieu and H. T. Ng. A Maximum Entropy Approach to Information Extraction from Semi-Structured and Free Text. In Proc. of Eighteenth National Conference on Artificial Intelligence, 2002.

[4] K. Church and W. Gale. Probability Scoring for Spelling Correction. In Statistics and Computing, Vol. 1, 1991, pages 93-103.

[5] A. Clark. Pre-processing Very Noisy Text. In Proc. of Workshop on Shallow Processing of Large Corpora. Corpus Linguistics 2003, Lancaster. 2003.

[6] M. Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In Proc. of Conference on Empirical Methods in NLP, 2002.

[7] C. Cortes and V. Vapnik. Support-Vector Networks. Machine Learning 20:273-297, 1995

[8] N. Ducheneaut and V. Bellotti. E-mail as Habitat: An Exploration of Embedded Personal Information Management. Interactions, 8, pages 30-38.

[9] W. A. Gale, K. W. Church, and D. Yarowsky. 1994. Discrimination Decisions for 100,000-Dimensional Spaces. Current Issues in Computational Linguistics: In Honour of Don Walker. Kluwer Academic Publishers, pages 429-450.

[10] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Declaratively Cleaning Your Data Using AJAX. In Journees Bases de Donnees, Oct. 2000. http://caravel.inria.fr/~galharda/BDA.ps.

[11] Z. Ghahramani and M. I. Jordan. Factorial Hidden Markov Models. Machine Learning, 29:245-273. 1997

[12] A. R. Golding and D. Roth. Applying Winnow to Context-Sensitive Spelling Correction. In Proc. of the 13th International Conference on Machine Learning (ICML 1996).

[13] M. A. Hernández and S. J. Stolfo. Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. Publisher Kluwer Academic Publishers Hingham, MA, USA. Vol. 2, Issue 1, January 1998, pages 9-37.

[14] N. Kushmerick. Learning to Remove Internet Advertisement. In Proc. of the Third International Conference on Autonomous Agents (Agents'99). ACM Press. Seattle, WA, USA. 1999, pages 175-181.

[15] J. Lafferty, A. McCallum, and F. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In Proc. of ICML 01, 2001.

[16] S. H. Lin and J. M. Ho. Discovering Informative Content Blocks from Web Documents. In Proc. of SIGKDD-02, 2002.

[17] L. V. Lita, A. Ittycheriah, S. Roukos, and N. Kambhatla. tRuEcasIng. In Proc. of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003), July 7-12, Sapporo, Japan.

[18] E. Mays, F. J. Damerau, and R. L. Mercer. Context Based Spelling Correction. Information Processing and Management: an International Journal, v.27 n.5, pages 517-522, 1991

[19] A. McCallum, D. Freitag, and F. Pereira. Maximum Entropy Markov Models for Information Extraction and Segmentation, In Proc. of the ICML Coference, 2000.

[20] A. Mikheev. Periods, Capitalized Words, etc. Computational Linguistics, 28(3):289-318, 2002.

[21] H. T. Ng, C. Y. Lim, J. L. T. Koo. Learning to Recognize Tables in Free Text. In Proc. of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics (ACL'99). 1999, pages 443-450.

[22] D. D. Palmer and M. A. Hearst. Adaptive Multilingual Sentence Boundary Disambiguation. Computational Linguistics. MIT Press Cambridge, MA, USA. Vol. 23, Issue 2 (June 1997), pages 241–267.

[23] C. Parent, and S. Spaccapietra. Issues and Approaches of Database Integration. Comm. ACM 41(5):166-178, 1998.

[24] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table Extraction Using Conditional Random Fields. In Proc. of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2003.

[25] E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches, IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 23 No. 4, December 2000

[26] A. Ratnaparkhi. Unsupervised Statistical Models for Prepositional Phrase Attachment. In Proc. of COLINGACL98. Montreal, Canada, 1998.

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

[28] V. Vapnik. Statistical Learning Theroy. Springer Verlage, New York, 1998.

[29] Y. Wang and J. Hu. A Machine Learning based Approach for Table Detection on the Web. In Proc. of The Eleventh International World Wide Web Conference (WWW2002), Honolulu, Hawaii, USA, May 2002

[30] E. Xun, C. Huang, and M. Zhou. A Unified Statistical Model for the Identification of English baseNP. In Proc. of The 38th Annual Meeting of the Association for Computational Linguistics (ACL), Hong Kong, 3 - 6 October 2000

[31] L. Yi and B. Liu, and X. Li. Eliminating Noisy Information in Web Pages for Data Mining. In Proc. of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2003), Washington, DC, USA, August, 2003, pages 24-27.

[32] L. Zhang, Y. Pan, and T. Zhang. Recognising and Using Named Entities: Focused Named Entity Recognition Using Machine Learning. In Proc. of SIGIR'04, 2004.

[33] J. Decker. eClean 2000, http://www.jd-software.com/eClean2000/index.html.

[34] ListCleaner Pro. http://www.WinPure.com/EmailCleaner.

[35] Fuzzy Lookup and Fuzzy Grouping in Data Transformation Services for SQL Server 2005. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql90/html/FzDTSSQL05.asp