Automated Email Answering by Text Pattern Matching

Eriks Sneiders

Department of Computer and Systems Sciences,
Stockholm University,
Forum 100, SE-16440 Kista, Sweden
eriks@dsv.su.se

Abstract. Answering email by standard answers is a common practice at contact centers. Our research assists this process by creating reply messages that contain one or several standard answers. Our standard answers are linked to representative text patterns that match incoming messages. The system works in three languages. The performance was evaluated on two email sets; the main advantage of our email answering technique is good correctness of the delivered replies.

Keywords: Automated email answering, automatic email response, text message answering, question answering, text patterns.

1 Introduction

It is not unusual that an email flow to a contact center (aka customer care center, customer service) contains frequently reoccurring inquiries, therefore agents who communicate with customers use predefined response templates as draft answers. Finding a predefined answer, if it exists, is a task that a computer can do. Answering a generic question (e.g., "Can I pay with my Visa card?") would be easy. More specific requests (e.g., "Please update my address in your customer database…") are less trivial. Fortunately, many companies have a web-based self-service system where a customer logs in and interacts with the system without mediation of a contact center agent. Hence, an automated answer can advise using the self-service system, where appropriate.

Katakis et al. [1] present a good introduction to email management techniques and their application domains. Most research has been done assuming personal use of email. Automated message answering at contact centers has raised less interest.

Most email answering systems pursue the text classification approach. A typical system perceives a message as a bag of words represented by a term vector with tf-idf weights. Normally the words are stemmed and stop-words removed. Further, a typical system is trained on sample documents in predefined classes. The most popular learning algorithms are Support Vector Machine and Naïve Bayes [2][3][4]; Ripper and K Nearest Neighbors have also been used [3]. After the training phase, a new message is placed into one of the predefined classes, e.g., assigned a predefined answer or a small set of candidate answers.

Malik et al. [5] map training email messages to standard answers by identifying questions that contain key phrases of length up to 3 words. After the training, these questions are matched to questions in query emails.

H. Loftsson, E. Rögnvaldsson, S. Helgadóttir (Eds.): IceTAL 2010, LNAI 6233, pp. 381–392, 2010.

Weng and Liu [6] assign a set of representative concepts with weighted terms to each class of messages. When a new message arrives, its weight is calculated with respect to each concept set considering the terms in the message and their weights.

Very few email answering systems do text generation. Marom and Zukerman [7] create new response texts by selecting the most representative sentences from previous responses to messages similar to the new incoming message. Kosseim at al. [8] follow the tradition of Information Extraction and operate a number of templates for capturing intention, concepts, named entities, and relations. When a new message arrives, the system fills the templates, performs semantic validation, and generates the answer.

This paper introduces an email answering approach that has been used for sending replies without any human intervention as well as generating draft replies for contact center agents. The system operates a database of standard answers and text patterns, linked to these answers, that record expected wordings in messages to be answered.

2 Pattern-Based Email Answering

This research stems from an earlier work in automated FAQ answering in a restricted domain: the system could answer about 70% of the queries, 9 out of 10 answers correct [9]. A natural step forward is to adapt the technique to answer larger pieces of text, such as email messages.

2.1 Question Templates

The email answering system has a few standard answers that respond to the most frequent inquiries. Each standard answer has a title that summarizes it in one sentence, which helps to avoid confusion while reading the automatic reply if the answer does not quite correspond to the original message. Furthermore, each standard answer is linked to a number of question templates that record the expected text patterns of the future inquiries to be answered by this standard answer, Fig. 1.

The syntax of our text patterns resembles that of regular expressions; the text patterns are less rigorous though. Each question template contains two patterns. The required pattern matches a piece of message text if the message fits this standard answer. The forbidden pattern must not match the message; it detects details that disqualify the answer. Please observe that the question templates are created before the actual email answering starts.

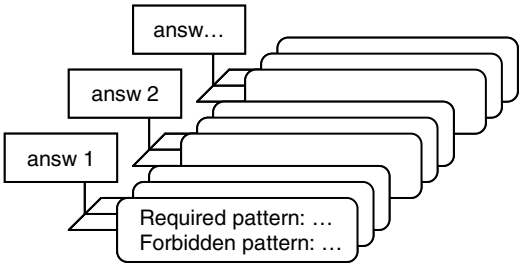

Fig. 1. Standard answers with their question templates

2.2 Steps of the Answering Process

During the email answering process, a new incoming message is split into paragraphs, each paragraph into sentences, each sentence into terms – words, numbers, and email addresses. Then the words are spell-checked (see Section 3); all spelling alternatives are equal.

After the message has been preprocessed, the system matches each question template, one by one, to each paragraph of the message, one by one. The matching is case-insensitive. If the forbidden pattern matches at least one paragraph, the system drops the current question template and starts all over again with the next question template. If the required pattern matches at least one paragraph, and the forbidden pattern does not match any, the system accepts the standard answer linked to the current question template.

One incoming message may contain several inquiries and match several question templates, and therefore be answered by several standard answers included in one reply message. The length of the message is not limited. Nonetheless, if the message matches too many standard answers, we would rather not answer it automatically.

2.3 Components of a Text Pattern

A text pattern, which matches a piece of a text message, is a collection of lexical items and rules that state how the system matches those items to terms in a paragraph of a query message. A text pattern is written in plain text following a simple syntax.

There are three types of basic lexical items: words and word stems (e.g., customer* or %paper), numbers (e.g., 29), and entity slots. An entity slot represents any item of a given concept. Currently we have entity slots for numbers and email addresses, written as <number> and <email>. The diversity of possible entity slots is limited only by the systems ability to recognize a piece of text as a concept, and isolate that piece during email text tokenization.

Lexical items are organized into synonym sets. For example, ford volvo ; car vehicle* ; repair* are three synonym sets, where at least one synonym per set, in every set, should match any term in the query paragraph.

A phrase defines the order of and the distance between synonym sets. A phrase itself is a synonym in another synonym set. Phrases can be embedded. A phrase matches terms within the boundaries of one sentence. A phrase defines three neighborhood options for matching terms:

- adjacent, e.g., [bright; light] matches "bright light" only;
- distant, e.g., [bright # light] allows unlimited number of words between "bright" and "light" within one sentence;
- optional, e. g., [bright : day ; light] matches both "bright light" and "bright day light".

An empty synonym set matches everything, therefore [bright :::; light] matches “bright” and “light” with 0-3 any lexical items in between.

Phrases match compound words. For example, [new*; paper*] matches “new paper” and “newspapers”. Because compound words are popular in some languages such as Swedish, we have special syntax for them, e.g., news%paper* is equal to [news* ; paper*].

We define reoccurring pieces of text patterns as substitutes and reuse them. For example, we define ford volvo as $car_name and use it in $car_name ; car vehicle* ; repair*.

In order to minimize words stem ambiguity, the stems are required to have at least five letters before the asterisk. In the syntax for compound words, a component must be at least four letters long, or at least two if there is another component at least five letters long. This has proved a sufficient trade-off between the ambiguity of word stems and their ability to represent concepts.

2.4 Test Beds

Before we discuss real-life examples of the text patterns, let us introduce our test beds. The email answering system was implemented at two contact centers. The first one was an insurance company that employed fully automated email answering with 11 standard answers in Swedish. The system scanned through all incoming messages. If it could answer the message, it sent a reply to the "from" and "cc" addresses of the original message. The reply informed its reader that it was computer-created and contained simple instructions how to reach a human agent if necessary. Messages that could not be answered were passed to a human agent.

The second contact center worked for a telecom service provider. It used 4 standard answers in Latvian. Here, email answering was not fully automated; the system created draft reply messages for the agents of the contact center.

2.5 Flexibility of a Text Pattern

The following example illustrates what a text pattern may look like:

```
[ $2_4_hjul #
    [vad ::; $kosta :::; $försäkring $2_4_hjulförsäkring $trafikförsäkring $hel_halv_försäkring]
    [vad ::; $försäkring $2_4_hjulförsäkring $hel_halv_försäkring :::::; $kosta $pris2]
    [$försäkring $2_4_hjulförsäkring $trafikförsäkring $hel_halv_försäkring # vad ::::::;
        $kosta $pris2]
]
```

A text pattern is more than a loose regular expression; it embodies a question-specific synonym dictionary and a quasi-ontology. Furthermore, a combination of words and matching rules is delicate, even small changes may influence the accuracy of detecting relevant texts. Therefore, at the current stage of our research, management of the text patterns requires a manual effort.

In order to get an impression about the potential size of a text pattern, let us see how many terms in a piece of query text (i.e., a paragraph) one pattern can match, taking the insurance case (see Section 2.4) as an example. The 11 standard answers were linked to 161 required text patterns. In average, a text pattern matched 5 terms in a paragraph if the system delivered an answer. Potentially the text patterns could match between 3 and 40 terms in a paragraph.

Fig. 2 shows the largest (top curve) and smallest (bottom curve) number of terms in a query paragraph that a text pattern could possibly match, for each of the 161 patterns lined up on the horizontal axis. On the very left side, there are text patterns that can match just over 40 query terms. In the middle of the line-up, the text patterns can match up to 15-20 query terms. The smallest text patterns on the right side can match just over 5

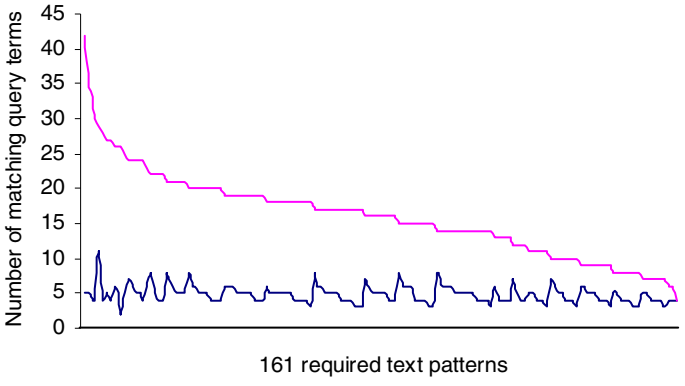

Fig. 2. Number of query terms that can match a required text pattern

terms in a query paragraph. The bottom curve oscillates like an electrocardiogram, with average 4.8 terms in a query paragraph that a text pattern must match as a minimum.

Why can one text pattern match a variable number of query terms? A synonym set may contain phrases of different length. The largest/smallest number of matching terms can be reached if the system always selects the longest/shortest synonym phrase, or a basic lexical item instead of the shortest phrase, in each synonym set. Fig. 2 suggests that a text pattern may contain a large number of parallel wordings that capture a variety of paraphrases of a meaningful statement. Furthermore, text patterns are built to match future messages, and we are uncertain what exactly these messages will look like. We are likely to compensate this uncertainty with some redundancy in the text patterns caused by guessing the future wordings. Hereby the text patterns are more complex but also more expressive than just a set of keywords and a synonym dictionary.

3 Spelling Correction

Email texts are often untidy. Tang et al. [10] inspected more than five thousand web-based newsgroup messages in English and discovered that 73.2% of them needed paragraph normalization, 85.4% needed sentence normalization, 47.1% needed upper-lower case restoration, and 7.4% of the messages contained misspellings. Dalianis [11] inspected spelling in another context of Internet-based communication and found out that about 10% of the search queries submitted to a search engine of the Swedish tax authority were misspelled. We did not count misspelled messages processed by our system, yet we know that without spelling alterations our text-pattern matching would fail to identify many pertinent messages because the matching rules, embodied in these patterns, are strict, and even one letter wrong would result in a no-match.

Furthermore, our system faces three challenges. First of all, spelling alternatives are sought among words, word stems, and phrases from the text patterns rather than in off-the-shelf tools.

The second challenge is multiple languages. The system works with texts in English, Swedish (Germanic/Indo-European languages), and Latvian (Baltic/Indo-European language).

The third challenge is substitution of language-specific character sets with the ISO-8859-1 or ASCII character sets present on virtually all computers. Of the three languages that our system works with, Latvian emails are most exposed to character set substitution. The Latvian alphabet has 33 letters, of which 11 are not included in the ISO-8859-1 character set and may get replaced with English letters. For example, "ā" becomes "a" or "aa", "ķ" becomes "k" or "kj" or "kk", etc. There are no rules, everything goes as long as people grasp the text. In this case we deal with a deliberately altered syntax, a pidgin language, rather than misspellings. The cure, however, is the same – spelling correction.

Our spelling correction approach has three building blocks: (i) detection of correctly spelled words, (ii) spelling modification with respect to standalone words and stems from the text patterns, and (iii) spelling modification with respect to phrases from the text patterns. The same algorithms are applied to texts in all three languages, except the system does not detect correct spelling in Latvian emails.

Detection of correct spelling and common misspellings. Correctly spelled words are detected by matching them to words, word stems, and phrases from the text patterns, as well as word lists available in public domain or donated by colleagues. Likewise, the system matches query words to a list of common misspellings and finds correct spelling for commonly misspelled words.

Detection of correctly spelled and frequently misspelled words is not performed to messages in Latvian because of frequent occurrence of pidgin language in them, where a misspelling of one word may be a homograph of correctly spelled another word. For example, "kāpa" and "kapa" are correctly spelled different words, yet "kapa" may be inappropriately used instead of "kāpa". Thus spelling correction is closely related to the problem of word sense disambiguation [12], which lies outside the scope of this paper. Our system assumes that every word in Latvian emails is potentially misspelled.

Words and word stems as spelling alternatives. About 80% of misspellings fall into four categories: one letter too many, one letter missing, a letter replaced by another one, or adjacent letters are transposed [13]. Our spelling correction is based on detecting these four typical mistakes. We tested also basic Soundex for English; it generated too many irrelevant options. There exist more advanced spelling error correction models, such as [14]. Still, advances in spelling correction lie outside the scope of this paper.

Our detection of replaced letters considers also sequences of letters, and looks for:

- phonetic similarity, e.g., "f" ~ "ph", "ea" ~ "ee", "k" ~ "ck", "s" ~ "c", etc.
- visual similarity, e.g., "ä" ~ "a", "š" ~ "s", etc.
- conventions in pidgin languages, e.g., "ä" ~ "ae", "š" ~ "sh" ~ "sch", etc.

If a standalone word stem (not a part of a phrase) is considered as a spelling alternative, the corrected word must be no more than three letters longer than the stem in order to hinder replacement of a compound word with the stem of its first component.

Phrases as spelling alternatives. Phrases from the text patterns, simple ones without other embedded phrases, are the main spelling alternatives for misspelled compound words. The system tests a phrase as a spelling alternative of a query word according to the following principle:

- the system applies synonyms from a synonym set to a query word
- in case if a stem fits the query word, the system takes the remainder of the query word, left behind the fitting stem, as a new query word
    - the system takes the next synonym set of the phrase as the current synonym set and repeatedly applies it to the new (remainder) query word, dropping the first letter of the new query word after each iteration in order to detect the beginning of the next component of a compound word
- if the remainder does not fit the current synonym set, the system takes the next query word and applies the current synonym set

Let us consider an example and test [fågel* fåglar* ; bur] as a spelling alternative of "faogelsbuur". First, the system discovers that fågel* fits "faogel". Then it takes the remainder "sbuur" and repeatedly applies bur, after each iteration dropping the first letter of the remainder, until bur fits "buur". Let us now try "faoglarnas buur". The system discovers that fåglar* fits "faoglar", that bur does not fit the remainder "nas", but it does fit the next query word "buur".

Order of the rules. The system conducts spelling correction in the following manner:

1. Correctly spelled words are detected, commonly misspelled words are corrected, for texts in English and Swedish.
2. Phrases from the text patterns are tested as spelling alternatives, longer phrases first. Corrected compound words are not further processed applying shorter phrases and standalone word stems.
3. Standalone words and word stems from the text patterns are tested as spelling alternatives.

When a word or a word stem is considered as a spelling alternative, the four typical mistakes are tested in the following order: (1) replaced letters, (2) adjacent letters transposed, (3) one letter too much or missing, tested simultaneously.

4 Performance Measurements

Before the email answering system can start operating, it is "trained" to recognize email texts that fit a given standard answer. We say "trained", in quotes, because this is not training as understood in machine learning. We use some text filtering, clustering, and aggregation tools in order to group messages. Then the messages are analyzed and text patterns created. Today, the text patterns are crafted manually; increased automation of this process is further research and lies outside the scope of this paper.

"Training" messages. Section 2.4 introduced our test beds – one contact center for a Swedish insurance company, another one for a Latvian telecom service provider.

In the insurance case, we had access to 5148 messages before the performance test; many of these messages had been analyzed in order to manually create and adjust the text patterns.

In the telecom case, we had access to two sets of messages. Initially we had 4782 messages, of which 706 messages corresponded to the 4 standard answers that the system included in its automated replies. During the operation of the system, but before the performance test, we acquired 3768 more messages that were analyzed in order to manually adjust the text patterns.

Because the systems were running in production settings, the "training" phase was the entire period the systems had been in operation. While doing the "training", we paid more attention to correctness of the replies rather than recall. We rather process a message manually than increase the risk of an incorrect reply.

Test messages. The data for the performance measurements came from the systems' logs. We had no prior access to these messages, we could not have used them in order to "train" the systems.

In the insurance case, 3526 consecutive messages were analyzed. In the telecom case, 1314 consecutive messages were analyzed. The correspondence between query messages and their automated replies was judged by humans, third party observers.

4.1 Precision, Recall, and Correctness

We applied two evaluation criteria. The first criterion was precision and recall, calculated for each query message separately. The second criterion was correctness of the replies actually given. We distinguished between:

- a fully correct reply;
- a correct partial reply where the system did not possess answers to all the questions in the query (recall = 1);
- a correct partial reply where the system did not find all the answers (recall < 1);
- an incorrect reply (usually right concepts, wrong relations);
- a technically correct reply where the system properly identified the question or the problem statement, but the message further implied that the author could not use the standard solution proposed in the reply.

Table 1 shows precision and recall calculated for each query message separately. Let us explain the insurance case. From the 3526 inspected messages, 395 messages got recall 1, 20 messages got recall 0.5, 179 messages got recall 0. In total, 594 messages (395+20+179) had relevant answers in the system's database and some recall value. The average recall – the sum of all recall values divided by the number of messages that had any recall value – was 0.682.

Precision values where calculated only for the query messages with a non-zero recall (precision is not defined otherwise), which was 415 messages (395+20). Most messages – 404 – got precision value 1. The average precision was 0.987.

Table 2 shows correctness of the replies, i.e., how many replies actually delivered were somewhat correct or incorrect. The messages sent to manual processing do not show up in this table.

Table 1. Precision and recall of the automatic replies

Precision value	Num queries with the precision value	Recall value	Num queries with the recall value
Insurance case			
1	404	1	395
0.67	1	0.5	20
0.5	9	0	179
0.33	1		
Average: 0.987 (for 415 queries)	Total: 415	Average: 0.682 (for 594 queries)	Total: 594
Telecom case			
1	119	1	120
0.5	5	0.5	4
		0	37
Average: 0.98 (for 124 queries)	Total: 124	Average: 0.758 (for 161 query)	Total: 161

Table 2. Correctness of the automatic replies

Fully correct	Technically correct	Partial, recall=1	Partial, recall<1	Incorrect	Total
Insurance case					
371	28	24	20	41	484
76.65%	5.79%	4.96%	4.13%	8.47%	100%
Telecom case					
111	6	9	4	16	146
76.03%	4.11%	6.16%	2.74%	10.96%	100%

A sharp-eyed reader has probably noticed that numbers in Table 1 and Table 2 do not match. Let us take the insurance case in order to explain these numbers. 371 fully answered plus 24+20 partially answered queries make 415 total precision queries. Incorrectly answered queries have either zero-recall or do not have any recall and precision values at all. Technically correct replies were judged as incorrect when precision and recall where calculated, because these queries should not have been answered. Therefore technically correct replies did not get any recall and precision values.

Both implementations of the system show surprisingly similar correctness and average precision percentages, despite different languages and knowledge domains. Correctness of the replies is good. From all the answered queries,

- around 76% were answered fully,
- around 85% were answered fully or partially,
- around 90% got the issues identified, fully or partially (fully correct, technically correct, partial replies).

4.2 Performance Figures in Context

The performance figures are easier to grasp if observed in the context of related systems, mentioned in the introduction, whose performance measurement methods are similar to those of ours.

Message classification. Busemann et al. [2] managed to choose the right category in 56.23% cases using Support Vector Machine; Weng and Liu [6] reached top performance, i.e., the highest F-value, at 62.77% recall and 77.52% precision by finding representative terms in query messages, and weighing these terms with respect to message classes.

Mapping a message to a standard answer. Malik et al. [5] made a human-equivalent selection of answer templates in 61% cases, human-equivalent or incomplete selection in 73.4% by first extracting a set of representative questions for each standard answer (training), and then mapping these questions to questions in query messages (test), calculating taxonomy-distance between words.

Information extraction and answer text generation. Kosseim et al. [8] delivered 66.4% correct answers by applying Information Extraction templates and performing semantic validation of the extracted information.

These figures give us an intuitive insight into viable quality of email answering. In order to make any formal claims which system performs better, we have to test the systems using the same input data and applying the same conditions.

5 Advantages and Limitations

Following are the advantages of our text pattern-based automated email answering.

The system demonstrates a good correctness of the replies and a superior ability to identify questions and problem statements relevant to standard answers, if the message is answered.

The text patterns operate in isolated narrow single-answer knowledge domains where they are self-sufficient. Therefore the system can start operation with rendering one standard answer, and the number of standard answers can gradually rise to as many as needed.

Because each text pattern is autonomous, the system can easily handle several questions in one query message and put several standard answers into one reply message.

Because the text patterns are self-sufficient, the system has a low technological threshold or barrier that impedes its deployment, i.e., the system operates without components such as part-of-speech taggers, stemmers, generic or domain ontologies. We do need some tool support to analyze training messages, however, as stated in further research.

Because of the low technological threshold, our approach is affordable for rare domains, small businesses, and small languages where open-source linguistic and knowledge-representation tools, as well as technologically advanced manpower are not readily available.

We have tested the system for English, Swedish, and Latvian, and consider it portable to most European languages.

Our approach is most advantageous in settings where correctness of the replies is crucial, where we want to maximize the end-users' experience, where a list of ten candidate answers is not an option. For example, in fully automated email answering without any human mediation. Especially advantageous our approach is for email flows with a high ratio of reoccurring inquiries, such as the email flow in [4] where 9 standard answers cover 72% of all messages.

Still, our automated email answering approach has at least two *limitations*. First, it is designed for narrow and stable domains only. It should not be considered for text classification tasks in arbitrary text collections. Second, at the current stage of our research, development of the text patterns is manual, which lessens the practical value of our technique until at least partial automation of this process is achieved.

6 Conclusions and Further Research

Our automated email answering system maps incoming messages to standard answers by matching text patterns linked to the standard answers. The technique was designed for narrow and stable domains, such as an email flow at a contact center. The main advantage of our technique is good correctness of the delivered replies. Performance evaluation on two email collections in two languages showed that about 85% of the messages could be answered fully or partially; about 90% had their questions and problem statements correctly identified. We consider the technique applicable to the majority of the European languages.

Currently we are working on performance comparison between our system and Support Vector Machine / Naïve Bayes on different datasets, which will be published separately.

The performance of the system depends on how good its text patterns are, therefore our further research focuses on the tools that help us craft these patterns. This includes: (i) learning from the past experiences, i.e., analysis of the existing email flow, and (ii) adjustment of the acquired learnings to meet future queries.

The research continues in three directions. First, we need to know what makes a good text pattern, what features we should strive to develop. Second, we need a good key phrase extraction tool that creates the foundation for new text patterns given a sample of messages. Third, we need a machine learning tool that adjusts existing text patterns to new training messages.

References

1. Katakis, I., Tsoumakas, G., Vlahavas, I.: Email Mining: Emerging Techniques for Email Management. In: Vakali, A., Pallis, G. (eds.) Web Data Management Practices: Emerging Techniques and Technologies, pp. 219–240. Idea Group Publishing, USA (2006)

2. Busemann, S., Schmeier, S., Arens, R.G.: Message classification in the call center. In: Proc. Sixth Conference on Applied Natural Language Processing, pp. 158–165. ACL (2000)
3. Lapalme, G., Kosseim, L.: Mercure: Towards an automatic e-mail follow-up system. IEEE Computational Intelligence Bulletin 2(1), 14–18 (2003)
4. Scheffer, T.: Email answering assistance by semi-supervised text classification. In: Intelligent Data Analysis, vol. 8(5), pp. 481–493. IOS Press, Amsterdam (2004)
5. Malik, R., Subramaniam, V., Kaushik, S.: Automatically Selecting Answer Templates to Respond to Customer Emails. In: Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, pp. 1659–1664 (2007)
6. Weng, S.S., Liu, C.K.: Using text classification and multiple concepts to answer e-mails. Expert Systems with Applications 26(4), 529–543 (2004)
7. Marom, Y., Zukerman, I.: Towards a Framework for Collating Help-desk Responses from Multiple Documents. In: Proceedings of the IJCAI Workshop on Knowledge and Reasoning for Answering Questions, Edinburgh, Scotland, pp. 32–39 (2005)
8. Kosseim, L., Beauregard, S., Lapalme, G.: Using information extraction and natural language generation to answer e-mail. Data & Knowledge Engineering 38, 85–100 (2001)
9. Sneiders, E.: Automated FAQ Answering with Question-Specific Knowledge Representation for Web Self-Service. In: Bello, L.L., Iannizzotto, G. (eds.) Proc. 2nd International Conference on Human System Interaction, pp. 298–305. IEEE, Los Alamitos (2009)
10. Tang, J., Li, H., Cao, Y., Tang, Z.: Email Data Cleaning. In: Proc. Eleventh ACM SIGKDD Internationl Conference on Knowledge Discovery in Data Mining, Chicago, Illinois, USA, pp. 489–498. ACM, New York (2005)
11. Dalianis, H.: Evaluating a Spelling Support in a Search Engine. In: Andersson, B., Bergholtz, M., Johannesson, P. (eds.) NLDB 2002. LNCS, vol. 2553, pp. 183–190. Springer, Heidelberg (2002)
12. Golding, A.R., Roth, D.: A Winnow-Based Approach to Context-Sensitive Spelling Correction. In: Machine Learning, vol. 34, pp. 107–130. Springer, Netherlands (1999)
13. Mays, E., Damerau, F.J., Mercer, R.L.: Context Based Spelling Correction. Information Processing & Management 27(5), 517–522 (1991)
14. Brill, E., Moore, R.C.: An Improved Error Model for Noisy Channel Spelling Correction. In: Proc. 38th Annual Meeting on Association for Computational Linguistics, Hong Kong, pp. 286–293. ACL (2000)