Linguistically Informed Information Retrieval for Contact Center Automation

Lars Buitinck
M.A. thesis
Information Science
University of Groningen

September 29, 2010
Preface

This thesis is the result of research performed by the author as an internship project at the Faculty of Economics and Business, University of Groningen. Thesis supervision was performed by Gosse Bouma, Faculty of Arts, and Ashwin Ittoo, Faculty of Economics and Business.

The research was carried out as part of the project “Merging of Incoherent Field Feedback Data into Prioritized Design Information (DataFusion),” sponsored by the Dutch Ministry of Economic Affairs under the IOP-IPCR program.

Thanks go out to my Gosse and Ashwin for providing support, instructions and feedback; to Marvin Siliakus for tips on language identification methods; to prof. John Nerbonne for advice on statistical testing; and to Gerrit Duits and Rian Hagebeuk for proofreading this text.

This thesis was typeset by the author in \LaTeX, using the KOMA-Script package.
1 Introduction

Customer service departments need to handle an increasing volume of textual data in the form of electronic mail. To handle this volume, some kind of automated processing is required. The aim of the research described in this thesis is to employ techniques from the fields of information retrieval (IR) and natural language processing (NLP; Jurafsky and Martin 2009) to automate part of the customer service pipeline.

In particular, the problem to be solved is that of associating incoming customer messages, describing complaints, suggestions or requests for information, with product problems previously acknowledged (and perhaps already solved). Both types of information come in the form of textual descriptions, the first formulated by the customer, the second by a customer support department. We call the former category incidents, and the latter category known errors (KEs).

In the rest of this thesis, wherever the term ‘incident’ is used, it should be understood as the subject field of an email sent in by a customer. We reserve the term ‘message’ for the full length email, as well as any replies to this message from the customer support department and messages sent between employees inside this department.

In addition, a set of problem descriptions is available, which, like KEs, are organization-internal descriptions of acknowledged problems that are not yet acknowledged as product errors.

We thus face a problem that falls into the domain of content-based document management or information retrieval (Baeza-Yates and Ribeiro-Neto 1999; Manning et al. 2008). This points to the range of common techniques from this field.

At first, the problem at hand seems superficially similar to the problem of text classification (also known as text categorization or topic spotting; Manning and Schütze 1999, pp. 575–608, Nenkova and Bagga 2003, Sebastiani 2005) where texts are to be automatically divided into classes/categories—say, email into classes \{spam, non-spam\}, \{not-urgent, urgent, emergency\}, or \{department A, department B, \ldots\}—so that they can then be handled appropriately. Classification techniques belong to the machine learning/data mining paradigm of supervised learning, where a set of instances (texts) is preclassified manually so that the appropriate features of this set can be summarized in a classification model.

However, classification was deemed impractical for the specific task and dataset dealt with here. Most research in email classification has assumed small, fixed numbers of classes with large numbers of instances.

For example, in the overview paper of Sebastiani (2002), the largest number of classes for a given classification problem is 135; for that problem, a dataset of 21,450 documents is used. The same holds true in the setting of contact center automation (Busemann et al. 2000, Beckers et al. 2009). Busemann et al. working with a set of 4777 contact
center emails in 74 categories, consider only those 47 categories which contain at least
30 documents.

The sparseness in the data set at hand, however, is far greater than that in traditional
research on text/email classification. Besides, existing classification systems are geared
toward closed sets of classes, making their use in industrial applications with dynamic,
open-ended sets of classes (such as the set of known errors that we are considering)
impractical; for every change in the set of classes, the classifier may have to be re-
learned.

With these constraints in mind, the problem can be phrased as one of ad-hoc inform-
ation retrieval, the technology underlying web search engines such as Google and Bing
as well as specialized textual search applications such as electronic library catalogues.
In ad-hoc retrieval systems, a user enters a query, which in the simplest case is a set of
keywords, and the software searches for the query’s keywords in an index of documents.
In this case, an incident is regarded as a query and a document consists of an incident
with, when available, its associated KE.

A benefit of choosing the ad-hoc IR framework is that it is well-studied and ready-
to-use software packages are available to put it into operation. These packages are
designed to be fast, dynamic (supporting changes to indexes) and robust. This enables
rapid deployment of baseline systems and rapid development of more advanced ones
g geared toward a specific domain of use.

The rest of this thesis is structured as follows. Chapter 2 describes in detail the system
sketched above and the models and methods used to construct it. Chapter 3 character-
izes the dataset used for formal evaluation of several configurations of the IR system,
described in Chapter 4. Chapter 5 concludes, discusses the difficulties encountered,
sketches possibilities for practical implementations and makes suggestions for further
research.
2 Methods

This chapter describes first a simple information retrieval system (search engine), then a series of successive (potential) improvements to be achieved by adding linguistic knowledge to it.

The workflow of the final program is summarized in figure 2.1. This figure displays a dual pipeline of processing modules: the left pipeline displays the construction of an IR document index, while the right pipeline shows the processing steps performed on an incoming incident report. Note that modules with the same name share a single implementation, but the steps ‘Cleanup’ through ‘POS filter’ are the same; the difference between the two pipelines lie in their data sources and processing after POS filtering. In the final step, the IR search engine receives a query constructed from the incident and consults the index to produce a ranked list of relevant KEs.

Note that this diagram does not display language guessing, a technique we explore in Section 2.2.2. The reason for this shall become clear later on in this chapter.

2.1 Document model

All experiments described in this thesis were performed using the open source document retrieval toolkit Lucene which in recent years has seen extensive use in both industrial and academic settings [Yee et al., 2006]. Lucene is at its core an ad-hoc document retrieval engine, employing a combination of two IR models, the Boolean model and the vector space model [Salton et al., 1975]. The following is a formal exposition of these models, adapted from [Baeza-Yates and Ribeiro-Neto, 1999, pp. 23–30].

Both models assume documents are represented as vectors of terms extracted from the document/query and their associated weights. Formally, if \( D = \{d_1, \ldots, d_N\} \) is a set of

---

\(^1\) Specifically, the Lucene Java implementation version 3.0.1, developed by the Apache Software Foundation and available at [http://lucene.apache.org/java](http://lucene.apache.org/java)

---
documents and $V = \{t_1, \ldots, t_T\}$ is the set of all terms occurring in $D$, then a document is represented by a $T$-dimensional vector $\vec{d} = (w_{t_1,d}, \ldots, w_{t_T,d})$ where $w_{t,d}$ is the weight of term $t$ in document $d$. The exact definition of $w$ varies per model.

Queries are represented by similar vectors $\vec{q}$, mapping query components to weights.

Both models define a similarity score function $\text{sim}(d, q)$ between a document $d$ and a query $q$. The similarity score is interpreted as a measure of the relevance of $d$ to $q$. Given a query, IR systems such as Lucene sort all documents in a collection (the index) by descending similarity and return these to the user (usually up to a certain cutoff rank and often only if a threshold similarity value is reached).

### 2.1.1 Boolean model

The Boolean model restricts weights to binary values $w_{t,d} \in \{0, 1\}$, thus interpreting documents as sets of terms. Queries in this model are Boolean/set-theoretic expressions describing sets of documents. In the general case, the disjunctive normal form (DNF) of a query can be interpreted as a weight vector over conjunctions of literals (Baeza-Yates and Ribeiro-Neto [1999] p. 26), but for simplicity all experiments described here use disjunctions of atomic term queries. The Boolean model’s similarity function thus becomes:

$$w_{t,d} = \begin{cases} 1 & \text{if term } t \text{ occurs in } d \\ 0 & \text{otherwise} \end{cases}$$

$$\text{sim}_b(d, q) = \bigvee_{t \in V} \vec{d}_t \times \vec{q}_t$$

The Boolean model in Lucene functions as a pre-selection step for vector space retrieval; any document scoring zero according to Boolean similarity is considered irrelevant and not retrieved.

### 2.1.2 Vector space model

In the vector space model (VSM), similarity is based on the cosine of the angle between vectors $\vec{d}$ and $\vec{q}$, defined as

$$\text{sim}(d, q) = \frac{\vec{d} \cdot \vec{q}}{|\vec{d}| \times |\vec{q}|} = \frac{\sum_{t \in V} w_{t,d} \times w_{t,q}}{\sqrt{\sum_{t \in V} w_{t,d}^2} \times \sqrt{\sum_{t \in V} w_{t,q}^2}}$$

---

2 This was achieved by by-passing Lucene’s QueryParser class, which implements a complex query language. Preliminary experiments showed the parser’s special handling of quotation marks and Boolean keywords to have detrimental effects on retrieval performance. Lucene’s ‘escaping’ of special characters was found to be buggy.
Similarity in this model is real-valued, expressing a degree of similarity between a
document and a query in terms of textual content. Weights are assigned using the tf-idf
formula, which in its simplest form is the product of a term’s frequency in a document
(the term frequency, \( tf \)) and the (logarithm of the) inverse of its frequency across the
document collection (\( idf \)). The latter term is intended to penalize frequently occurring
terms, as these are likely to carry little information. Let \( T_D \) denote the set of all terms
in all documents in a collection \( D \) and \( #(t, d) \) the raw frequency (occurrence count) of
the term \( t \) in a document \( d \). The tf-idf weighting formula is then:

\[
\begin{align*}
tf_{t,d} &= \frac{ #(t, d)}{\max_{t' \in T_D} #(t', d)} \\
idf_t &= \log \frac{|D|}{#(d \in D : t \in d)} \\
w_{t,d} &= tf_{t,d} \times idf_t
\end{align*}
\]

Many variants of tf-idf have been developed for practical IR ([Salton and Buckley 1988]
and [Manning et al. 2008] pp. 126–132 list dozens of variants). We use the default tf-idf
variant provided by Lucene, which uses the following definitions of \( tf \) and \( idf \):

\[
\begin{align*}
tf_{t,d} &= \sqrt{ #(t, d)} \\
idf_t &= \left(1 + \log \frac{|D|}{#(d \in D : t \in d) + 1} \right)^2
\end{align*}
\]

The resulting cosine score is multiplied by a normalization factor based on query and
document length. All computations are done with limited precision to optimize index
size and computing time, so results are really approximations of the above. Note that we
do not employ Lucene features such as multi-field indexing (called multi-zone indexing
by [Manning et al. 2008]) or boosting (which adds extra weight to selected terms).

### 2.1.3 Construction of documents

The above exposition assumed definitions of the concepts \textit{query} and \textit{document} (index
record). However, when choosing how to construct these two kinds of objects from inci-
cidents, problems and KEs, we again face several options. Most information retrieval
research works with files, emails, webpages, etc., which have a straightforward inter-
pretation as indexable documents. Our KE summaries, however, are very short (on the
order of ten tokens; see Section 3.3), making it implausible that all the relevant keywords
appear in any KE summary.

Queries are always built from a single incident with textual data from the associated
problem (if any) appended term-by-term.

Three different setups for documents were considered:

3 As described in the Lucene API documentation, \texttt{http://tinyurl.com/lucene-tfidf}
1. KE summary text plus problem description;

2. one incident with associated KE summaries;

3. KE summary plus problem description and the text of all associated incidents;

4. KE summary plus problem description and the full text of all associated incidents and messages.

As we shall see in Chapter 4, these three setups vary widely in their retrieval performance characteristics.

2.2 Natural language processing

2.2.1 N-gram indexing

One way to tackle the problem of multilingualism in our dataset is to use as terms not the words appearing in the text, but their character level \( n \)-grams. The hope is that this approach will capture common substrings of related words as index terms. For example, the English problem, Dutch probleem and Italian problema are very similar at the \( n \)-gram level. The same way, \( n \)-gram indexing may catch spelling errors and functions as a ‘poor man’s stemmer’ (see section 2.2.3 below).

A different way to exploit \( n \)-grams is to index token-level \( n \)-grams, sometimes called *shingles*. This permits capturing fixed expressions such as product names: an index search for a “FooWidget 4” will usually find any documents containing “FooWidget” and “4”, but when using shingle indexing, it will give extra weight to the exact occurrence of the phrase “FooWidget 4”.

2.2.2 Language guessing

While KE and problem descriptions are all entirely in English, incident reports may be written in any of a number of languages. A language field was supplied with these data but was found to be incorrect in many ways, due to the language field being tied to the customer’s email address rather than (manual or automatic) per-email language tagging. Many emails solicited replies and/or forwarded messages in different languages, leading to a number of multilingual messages.

Because of multilinguality, an obvious approach to adding linguistic knowledge is to first classify documents by language, then apply language-specific NLP techniques. For this purpose, a language guesser (also known as a language identifier or text classifier) is needed. The most commonly used language identification algorithm is an \( n \)-gram-based statistical method due to \textit{Cavnar and Trenkle} (1994). Several implementations of this algorithm are freely available.

Another simple method for language identification, based on counting the occurrences of very frequent words, is presented by \textit{Grefenstette} (1995). This algorithm was rejected.

\(^4\) All product names were anonymized.
on the grounds that Grefenstette reports the method of Cavnar and Trenkle to yield higher accuracy. His remark on the higher computing time required by the \textit{n}-gram method is no longer very relevant on today’s machines. More advanced methods of language identification are reviewed by Hughes et al. (2006), who note that “all of the previous work assumes collections of minimally tens, and frequently hundreds of thousands of words of gold standard data for language identification” and that handling multilingual documents is an outstanding issue.

An experiment with the Java Text Categorizing Library (JTCL)\footnote{an implementation of the Cavnar and Trenkle algorithm,} confirms these findings. A random sample of 639 incidents containing 5122 words were manually categorized by language. Languages were found to be distributed as in Table 2.1.

<table>
<thead>
<tr>
<th>Language</th>
<th>Number of incidents</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>554</td>
<td>86.7</td>
</tr>
<tr>
<td>Danish</td>
<td>43</td>
<td>6.7</td>
</tr>
<tr>
<td>German</td>
<td>22</td>
<td>3.4</td>
</tr>
<tr>
<td>Dutch</td>
<td>16</td>
<td>2.5</td>
</tr>
<tr>
<td>French</td>
<td>4</td>
<td>0.6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>639</strong></td>
<td><strong>100.0</strong></td>
</tr>
</tbody>
</table>

Table 2.1: Language distribution for incident subjects

Note that the distribution is heavily skewed. The language models supplied with JTCL achieved 77.0\% accuracy on this set of incidents. Since a baseline method of assuming that every incident is written in English scores better at 86.7\% accuracy, an attempt was made to construct new, domain-specific language models. LMs constructed from 3770 incidents (73.6\%, evenly distributed per language) and tested on the remaining 1352 achieved an accuracy of only 37.9\%. The construction of proper language models was deemed too time-consuming and the prospect of success too low.

Probable cause of the low performance of language guessing is the abundance of technical terms, mostly product names, regardless of language, in combination with the very short length of incident subjects. Together, this means that quasi-English terms and numbers make up much of the “\textit{n}-gram mass” of incidents, confusing the comparison routine in the Cavnar and Trenkle algorithm.

The distribution of languages for messages, as reported by the language guesser, is slightly different; see Table 2.2. On messages, the language guesser seems upon manual inspection to perform significantly better than incidents.

### 2.2.3 Part of speech tagging and lemmatization

Part of speech (POS) tagging is the automatic assignment of word classes (noun, verb, etc.) to words. We employ the Stanford Log-linear POS tagger (Toutanova and Manning, \url{http://textcat.sourceforge.net}).
<table>
<thead>
<tr>
<th>Language</th>
<th>Number of messages</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>6559</td>
<td>73.5</td>
</tr>
<tr>
<td>Danish</td>
<td>1104</td>
<td>12.3</td>
</tr>
<tr>
<td>Dutch</td>
<td>589</td>
<td>6.6</td>
</tr>
<tr>
<td>German</td>
<td>354</td>
<td>4.0</td>
</tr>
<tr>
<td>French</td>
<td>161</td>
<td>1.8</td>
</tr>
<tr>
<td>Italian</td>
<td>120</td>
<td>1.3</td>
</tr>
<tr>
<td>Spanish</td>
<td>55</td>
<td>0.6</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>8942</strong></td>
<td><strong>100.0</strong></td>
</tr>
</tbody>
</table>

Table 2.2: Language guesser results on incident messages

2000) [Toutanova et al., 2003], based on the maximum-entropy (MaxEnt) model, and use its results to filter away some word classes, in the expectation that these are unlikely to be informative.

Specifically, we use the `left3words-wsj-0-18` model delivered with the Stanford tagger, a model of English employing the Penn Treebank tagset [Marcus et al., 1993], and filter away words tagged POS tags that indicate function words (as opposed to content words). The full list of POS tags filtered away is listed in table 2.3.

<table>
<thead>
<tr>
<th>Tag(s)</th>
<th>Part of speech</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>Conjunction</td>
<td>‘and’, ‘or’</td>
</tr>
<tr>
<td>DT, WDT</td>
<td>Determiner</td>
<td>‘the’, ‘a’, ‘which’</td>
</tr>
<tr>
<td>MD</td>
<td>Modal verb</td>
<td>‘can’, ‘will’</td>
</tr>
<tr>
<td>PRP, PRP$, WP, WP$</td>
<td>Pronoun</td>
<td>‘he’, ‘who’, ‘her’, ‘whose’</td>
</tr>
<tr>
<td>UH</td>
<td>Interjection</td>
<td>‘uhm’</td>
</tr>
<tr>
<td>WRB</td>
<td>Wh-adverb</td>
<td>‘how’, ‘where’</td>
</tr>
<tr>
<td>$, “ ”, various others</td>
<td>Punctuation</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: Parts-of-speech eliminated before indexing/retrieval

Content words, i.e. nouns, (non-modal) verbs, adjectives, etc. remain. Note that the ‘foreign word’ category (FW), used in the Penn Treebank tagset to denote non-English words, is kept.

Lemmatizing is the process of retrieving the lemma or root form of a word. A simple approximation to lemmatization known as stemming, where finite-state heuristics such as affix stripping or replacement are applied to the surface form of a word, has been tried extensively in previous information retrieval research; however, is generally considered too brittle, at least in the context of English-language IR [Jurafsky and Martin, 2009, p. 102, p. 806; Baeza-Yates and Ribeiro-Neto, 1999, p. 168]. In our multi-lingual setup, we also risk incorrectly conflating non-English terms with English terms or with each other. Thus, despite the availability in Lucene of an English language stemmer based
on that of Porter (1980), stemming was disregarded in favor of full lemmatization.

The lemmatizer we employ is the one integrated into the Stanford POS tagger: which is based on the work of Minnen et al. (2001): a finite-state transducer that takes into account both the POS tag and the surface form of the input word. It should be noted that the lemmatizer leaves (properly detected) non-English words as they are. It may perform erroneous transformations on undetected ‘foreign’ words, though.

When applying lemmatization, we keep both the original (inflected) forms occurring in queries and documents and their lemmas. This prevents degraded performance in the face of erroneous lemmatization (esp. in the case of non-English words). Queries and documents are thus both approximately doubled in length.

2.2.4 Spelling correction

The use of automated spelling correction in information retrieval applications dates back to at least the work of Damerau (1964) and in recent years has been applied by major internet search engines including Google, Yahoo! and Bing. The central idea is to check whether a query term is a valid word or not, and if it is not, replace it with a valid word that is in some formal sense similar to the original term.

In the following, we will assume that a term is invalid/misspelt if it occurs in a query but does not occur in the index. The reason to prefer this assumption over the usual assumption in spell checking that a word is misspelt if it does not occur in some external dictionary, is three-fold: first, the incident data is multi-lingual, so several dictionaries would have to be combined to cover all incident reports; second, the data contains many domain-specific terms (such as product names), which would have to be added to the dictionary; third, one of the techniques tried relies on word occurrence statistics, which general-purpose dictionaries do not commonly provide (and in any case, not for domain-specific terms).

Lucene built-in spell correction

The first automated spelling corrector is the one supplied with Lucene. This program relies on a dictionary, compiled from all the terms in an index. Given a misspelt word, it retrieves all words in the dictionary ordered first increasing by distance from the misspelt word, then by decreasing document frequency. Three distance measures are supplied: Levenshtein distance (Sankoff and Kruskal, 1999; Navarro, 2001), Jaro–Winkler distance (Winkler, 1999), and n-gram distance, a generalization of LCS distance (Kondrak, 2005). Each of the distance metrics is normalized by dividing by the length of the longer of the two input strings: 

$$d' = \frac{d(x,y)}{\max(|x|,|y|)}.$$ 

Probabilistic spell correction

The second method of spelling correction is a relaxed version of the ‘noisy channel’ (probabilistic) model of Kernighan et al. (1990), suggested by Norvig (2007). Here, all strings within a finite Damerau distance (a variant of Levenshtein distance that includes
a limited notion of transposition; Damerau [1964] of the misspelt word are constructed and weighted according to posterior probability in a Bayesian framework.

To correct a word \( w \) in this framework, we restrict Damerau distance to a fixed \( k \), then compute sets of candidate corrections \( C_i \) for \( 1 \leq i \leq k \) and search for the candidate \( c \in \bigcup_{1 \leq i \leq k} C_i \) that maximizes \( P(w|c)P(c) \). Following Norvig, we define \( P(w|c) \) by stipulating that corrections at distance \( i \) are infinitely more probable than corrections at distance \( i + 1 \), rather than computing a full error model (the data for which is not available). \( P(c) \), the language model for the spelling corrector, is estimated by simple frequency counts of indexed tokens.

Simplifying the error model saves computing time, since if a correction at distance \( i \) occurs, the set of candidates at distances \( j > i \) need not be computed. In general, assuming constant-time hash table lookup for known terms and their probabilities/frequencies, the running time of this algorithm depends only on the length \( m \) of the input term. By contrast, the Lucene built-in spell checkers compute a distance measure between the input term and each of the \( n \) terms in a dictionary; given that these distance measures require \( O(m^2) \) time to compute (for roughly equal-length input strings), their total time consumption is \( O(nm^2) \). (While our implementation follows that of Norvig, this ‘shortcut’ also allows the use of potentially faster methods such as the finite-state Levenshtein transducers of Hassan et al. [2008].)
3 Dataset

All experiments described in this thesis were performed on the same dataset, collected from a corporate customer service department. This dataset differs from those generally used in academic research, primarily because it is very sparse (each ‘class’ has a low number of instances) and noisy.

3.1 Overview

As described in Chapter 1, our dataset consists primarily of known error (KE) and incident descriptions. In addition, a set of problem descriptions is available, which, like KEs, are organization-internal descriptions of perceived problems that are not yet acknowledged as product errors. Both KE descriptions and problem descriptions are always entirely in English.

Incident reports are the subject fields of emails sent in by customers, backed by the full-length email messages, replies to these messages from the customer support department and messages sent between employees inside this department. These messages may be written in any of a number of languages; see Section 2.2.2 for details. When we refer to incidents, we only refer to the aforementioned subject lines, whereas the term message is reserved for the emails that describe incident reports, responses to them, etc.

The dataset as delivered contains 74740 incidents, 3715 problems, 1285 KEs and 176911 messages. The following section describes cleaning/preprocessing measures taken to discard information in the dataset that was deemed irrelevant to either the algorithms employed or the evaluation procedure.

3.2 Data cleaning

Since incidents are the subjects of e-mail messages, they may contain ‘meta tokens’ such as Re:, Fwd:, etc. These were removed from the subject field to prevent the retrieval

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incident</td>
<td>012345-678901—FooWidget 4—FooWidget keyboard is broken</td>
</tr>
<tr>
<td>Message</td>
<td>012345-678901—Dear sir/madam, When I rebooted my FooWidget...</td>
</tr>
<tr>
<td>Problem</td>
<td>123456—FooWidget 4—Keyboard unresponsive after update</td>
</tr>
<tr>
<td>Known error</td>
<td>1234—FooWidget 4: Firmware update 4.123 breaks keyboard driver</td>
</tr>
</tbody>
</table>

Table 3.1: (Anonymized) sample data
engine from, e.g., associating replies with replies to other messages. In some cases, the last token of a subject field was found to be truncated (and marked with an ellipsis to show truncation). These truncated tokens were also discarded.

Incidents that had no associated KE, some 94% of the set, were discarded; such incidents would have a detrimental effect on formal evaluation, putting a 5% upper limit on such measures as precision. (In a live application, of course, all incidents and KEs would be stored.)

KEs were delivered as records containing 17 fields. Of these, only three were deemed relevant to information retrieval needs: the unique identification code, textual summary, and (when applicable) link to a problem. KEs that were not associated with any incidents were removed.

Messages were submitted to the further cleanup: an attempt was made to purge HTML tags, URLs occurring in free text, several fixed expressions apparently inserted by email software, message parts that indicate the presence and type of attachments and irregular spacing and punctuation (e.g., ellipsis denoted by more than three dot characters). In addition, messages not linked to an incident considered as relevant by the conditions set out above, were deleted.

### 3.3 Statistical characteristics

The dataset before cleaning is summarized in Table 3.3. Token counts are given only for the variable-length description fields, (so not for fields containing only product names or links to other tables) and were estimated using the GNU `wc(1)` program.

<table>
<thead>
<tr>
<th>Type</th>
<th>Total # documents</th>
<th>(tokens)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incident</td>
<td>74740</td>
<td>224223</td>
</tr>
<tr>
<td>Problems</td>
<td>3715</td>
<td>40902</td>
</tr>
<tr>
<td>Known error</td>
<td>1285</td>
<td>15464</td>
</tr>
<tr>
<td>Message</td>
<td>176662</td>
<td>11813800</td>
</tr>
</tbody>
</table>

Table 3.2: Statistical summary of data before cleaning

After deleting records that are not linked to other tables (e.g., incidents with no KE link field), the dataset can be summarized as in Table 3.3. Displayed in the two right columns are the token counts before and after stripping HTML tags, email meta-tags, etc.

The data are thus very sparse: the number of KEs is large in comparison to the number of incidents, and our texts (except for messages) are very short, on the order of ten tokens each. Figure 3.1 shows the uneven distribution of incidents per KE, which further aggravates the difficulty of automatically learning incident–KE relations. At one extreme, there is one KE with 136 related incidents; at the other, there are 221 KEs with only one incident and a total of 661 KEs (73.6%) with 10 or less incidents related to them.
<table>
<thead>
<tr>
<th>Type</th>
<th># documents</th>
<th># tokens before</th>
<th># tokens after</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incident</td>
<td>2830</td>
<td>22305</td>
<td>22156</td>
</tr>
<tr>
<td>Problem</td>
<td>867</td>
<td>9816</td>
<td>9014</td>
</tr>
<tr>
<td>Known error</td>
<td>898</td>
<td>10739</td>
<td>9731</td>
</tr>
<tr>
<td>Message</td>
<td>8935</td>
<td>763855</td>
<td>723474</td>
</tr>
</tbody>
</table>

Table 3.3: Portion of the dataset used for experiments

Figure 3.1: Rank-frequency plot displaying the number of incidents per known error
4 Experimental results

We provide formal evaluation of several configurations of the information retrieval system sketched in chapter 2 on the data described in 3, starting with a simple vector-space system and incrementally adding linguistic sophistication.

Each evaluation was carried out using the same procedure: a ten-fold cross validation was performed on the part of the incident set that is already (manually) linked to KEs. In each run, each of the incidents selected for testing is used to query an index constructed from the remaining incidents and their associated KEs. A cutoff value $K$ (set at 1, 5 and 10) is used to limit the number of KEs retrieved, to simulate a human operator scanning through the top-$K$ hits. Two common information retrieval metrics, recall and MRR, are computed in the cross validation. For reference, their harmonic mean is given as a single-value summary of the system’s performance.

4.1 Evaluation metrics

The mainstay of formal information retrieval evaluation are the precision, recall and $F_1$ measures. Recall is defined as the number of documents retrieved that satisfy the user’s information need (the true positives), divided by the total number of such documents in the indexed collection (the answer set).

We employ recall, but not the precision measure, as the latter has little value for our dataset: 64.0% of incidents is linked to only a single KE, so with a cutoff $> 1$, precision can never reach 100% for these incidents. Due to the nature of the dataset, any user of the methods described will simply have to sift through some irrelevant KEs in most cases. The best we can do is present the user with a ranked list of KEs that are potentially relevant, and make sure the ones that actually are relevant are ranked as high as possible.

Another problem with precision is that, even if it is normalized by taking into account the number of relevant KEs per incident, it still gives no clue as to how high the relevant KEs are ranked. For this reason, we use the mean reciprocal rank or MRR statistic instead.

MRR is commonly used in Question Answering (e.g. at the TREC conferences; Radev et al. 2002; Gillard et al. 2006) and is most suitable for tasks where one correct answer/hit is required. Although in our data, one incident may be linked to several KEs, this is only the case for 478 out of 2830 incidents, or 20.3%, justifying the use of MRR in conjunction with recall. The MRR is defined as the arithmetic mean of the reciprocal ranks of the

---

1 Assuming that the search engine retrieves more than one result for at least one query. This is not guaranteed, but is very likely due to term overlap.

2 Also called First Answer Reciprocal Rank (FARR), by Radev et al.
correct KEs retrieved for a query set $Q$:

$$\text{MRR}_K = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \text{RR}_K(q_i),$$

where the reciprocal rank is set to zero if the correct KE does not occur before the cutoff $K$:

$$\text{RR}_K(q) = \begin{cases} 
\frac{1}{(\text{rank of first correct KE for } q)} & \text{if first correct KE occurs among first } K \\
0 & \text{if no correct KE occurs among first } K
\end{cases}$$

Thus, a hit at rank one is awarded a score of 1, a hit at two gets $\frac{1}{2}$, etc. It follows that, if a hit is promoted from second to first place, RR goes up by $\frac{1}{2}$, while a one-rank-promotion at lower ranks always yields a lower increase in RR and MRR (e.g., a promotion from third to second place gives an RR improvement of $\frac{1}{2} - \frac{1}{3} = \frac{1}{6}$). The MRR thus reflects a strong preference for ‘hits’ at rank one.

MRR is geared towards for tasks where one correct answer/hit is required and gives a pessimistic view of our systems’ performance. Therefore, we use as our main evaluation measure the harmonic mean of MRR and recall, which we shall call $H$:

$$H = \frac{2 \times \text{MRR} \times \text{Recall}}{\text{MRR} + \text{Recall}}$$

This definition is analogous to that of Van Rijsbergen’s (1979) $F_1$ measure. Like $F_1$, the $H$ measure could be extended into an $H_\beta$ measure with a parameter $\beta$ to define the relative weight of MRR and recall, but for simplicity’s sake we have chosen to give both equal weight.

### 4.1.1 Cross validation

All results presented below are the means of 10-fold stratified cross validation, were the incident set is divided into ten partitions. In each ‘fold’ an index is recreated from nine of these partitions and queried using the remaining partition. The reason for using cross validation rather than a single evaluation run is that incidents not only serve as queries but are also used to enrich the index with extra information.

The partitioning was done once by a simple, deterministic algorithm, so the partitions are equal across experiments.

### 4.1.2 Statistical significance

Where the addition of a technique to a previous system shows only a small increase in performance, statistical significance tests are performed using the statistics package R$^3$. More precisely, we perform paired, one-tailed $t$-tests on the ten values $H$ produced by ten-fold cross validation for each of the cutoffs 1, 5 and 10. “Not significant” must be read to mean “not significant at a threshold $\alpha = 0.05$.”

---

$^3$ [http://r-project.org](http://r-project.org)
4.2 IR setups without NLP

The first question we face when applying ad hoc document retrieval to the problem of incident–KE linking is how to construct indexable documents from our data. In this section, various options are compared with regard to their retrieval performance. We shall show which of the basic setups performs best; all further experiments shall be performed only with this setup.

As a baseline, we first consider a setup were only KE summaries are used to construct index documents, and only incident report subjects and problem descriptions are used as queries; incident messages are entirely ignored. Doing so, we find the following values for the metrics explained above.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.3578 ± 0.0118</td>
<td>0.4836 ± 0.0142</td>
<td>0.5482 ± 0.0176</td>
</tr>
<tr>
<td>MRR</td>
<td>0.3848 ± 0.0137</td>
<td>0.4338 ± 0.0112</td>
<td>0.4426 ± 0.0111</td>
</tr>
<tr>
<td>H</td>
<td>0.3708</td>
<td>0.4573</td>
<td>0.4898</td>
</tr>
</tbody>
</table>

Performance of this setup is rather poor; recall exceeds .5 when a cutoff of ten KEs is specified, but MRR lags behind, indicating that a user may have to visually examine many KE summaries before a correct KE is found.

In the next setup, problem data is included with both documents (KEs) and queries (incidents), corresponding to setup 1 in Section 2.1.3. Note that not all incidents have an associated problem. It can be seen quickly that including problem data has a positive effect on performance.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.3867 ± 0.0201</td>
<td>0.4993 ± 0.0157</td>
<td>0.5605 ± 0.0128</td>
</tr>
<tr>
<td>MRR</td>
<td>0.4170 ± 0.0196</td>
<td>0.4618 ± 0.0168</td>
<td>0.4704 ± 0.0167</td>
</tr>
<tr>
<td>H</td>
<td>0.4013</td>
<td>0.4798</td>
<td>0.5115</td>
</tr>
</tbody>
</table>

We then consider a setup in which incidents are used as index records: each record/document is the text of one incident subject with any linked problem and KE summaries appended to it. This setup, number 2 in Section 2.1.3, is closest to classical document clustering, where inter-document ‘distance’ is measured by tf-idf.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.4088 ± 0.0194</td>
<td>0.5495 ± 0.0236</td>
<td>0.5991 ± 0.0202</td>
</tr>
<tr>
<td>MRR</td>
<td>0.4399 ± 0.0215</td>
<td>0.4981 ± 0.0214</td>
<td>0.5065 ± 0.0199</td>
</tr>
<tr>
<td>H</td>
<td>0.4238</td>
<td>0.5225</td>
<td>0.5489</td>
</tr>
</tbody>
</table>

\[ \text{See Baeza-Yates and Ribeiro-Neto (1999, p. 29) for an explanation of the similarity between ad hoc IR and document clustering.} \]
Performance of the setup where a document is constructed from a KE with all associated incidents and problem data (Section 2.1.3, number 3) is shown in the table below. As a side note, execution time for this setup is still low: running three 10-fold cross validations takes 30 seconds wall-clock time; this includes ten index constructions.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.4766 ± 0.0182</td>
<td>0.6191 ± 0.0126</td>
<td>0.6667 ± 0.0100</td>
</tr>
<tr>
<td>MRR</td>
<td>0.5155 ± 0.0193</td>
<td>0.5701 ± 0.0156</td>
<td>0.5767 ± 0.0153</td>
</tr>
<tr>
<td>H</td>
<td>0.4953</td>
<td>0.5936</td>
<td>0.6184</td>
</tr>
</tbody>
</table>

There is one more setup to test: setup 4 from Section 2.1.3, which corresponds to the previous setup, but with incident messages also included. Results for this setup are summarized in the table below.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.4646 ± 0.0085</td>
<td>0.5737 ± 0.0158</td>
<td>0.6130 ± 0.0168</td>
</tr>
<tr>
<td>MRR</td>
<td>0.4979 ± 0.0101</td>
<td>0.5417 ± 0.0124</td>
<td>0.5479 ± 0.0127</td>
</tr>
<tr>
<td>H</td>
<td>0.4807</td>
<td>0.5572</td>
<td>0.5786</td>
</tr>
</tbody>
</table>

Surprisingly, we find that including messages degrades retrieval performance. This goes against common wisdom in the machine learning and IR communities, that adding more data will generally improve systems’ performance. The reason for this seems to be that incident messages are extremely noisy: they are often multilingual, include many irrelevant terms, grammatical errors and some quasi-HTML tags that were very hard to consistently filter out.

We conclude that setup 3 consistently outperforms each of the other options and is thus suitable as a starting point for NLP experiments, discussed in the next section.

### 4.3 Evaluation of NLP techniques

The first NLP technique to be evaluated is the simplest: the character-level $n$-gram indexing described in Section 2.2.1. Adding this indexing method, with $3 \leq n \leq 6$, to the best-performing setup from the previous section results in the following figures:

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.4139 ± 0.0247</td>
<td>0.5477 ± 0.0178</td>
<td>0.6075 ± 0.0162</td>
</tr>
<tr>
<td>MRR</td>
<td>0.4459 ± 0.0285</td>
<td>0.4988 ± 0.0226</td>
<td>0.5067 ± 0.0222</td>
</tr>
<tr>
<td>H</td>
<td>0.4293</td>
<td>0.5221</td>
<td>0.5526</td>
</tr>
</tbody>
</table>

---

5 I/O is kept at a minimum by constructing the index in RAM. Wall-clock time includes loading time for the Jython interpreter, the OpenJDK Runtime Environment v1.6.0 and Lucene. Test machine specifications are: Intel Core 2 Duo processor at 2.33 GHz, 2 GB RAM, Ubuntu 10.04.
It is immediately clear that \( n \)-gram indexing has a detrimental effect on performance. If, instead of character-level \( n \)-grams, we index token-level bigrams, we find the following results:

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.4736 ± 0.0182</td>
<td>0.6047 ± 0.0192</td>
<td>0.6601 ± 0.0186</td>
</tr>
<tr>
<td>MRR</td>
<td>0.5092 ± 0.0201</td>
<td>0.5604 ± 0.0203</td>
<td>0.5678 ± 0.0191</td>
</tr>
<tr>
<td>( H )</td>
<td>0.4907</td>
<td>0.5817</td>
<td>0.6105</td>
</tr>
</tbody>
</table>

The previous table shows a minor decrease in performance. Stronger NLP techniques are thus called for.

With the part-of-speech tagging, POS-based filtering of non-informational terms and lemmatization described in Section 2.2.3 turned on, both recall and MRR can be seen to slightly, though consistently, increase at all three cutoff levels. The following table displays the exact figures.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.4936 ± 0.0202</td>
<td>0.6278 ± 0.0204</td>
<td>0.6761 ± 0.0255</td>
</tr>
<tr>
<td>MRR</td>
<td>0.5336 ± 0.0235</td>
<td>0.5851 ± 0.0228</td>
<td>0.5918 ± 0.0232</td>
</tr>
<tr>
<td>( H )</td>
<td>0.5128</td>
<td>0.6057</td>
<td>0.6311</td>
</tr>
</tbody>
</table>

A small further increase in performance can be achieved by combining lemmatization and token-level bigram indexing; see the following table. Again, the increase is consistent across summary figures recall, MRR and \( H \), but not significant when compared to the previous experiment.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.4961 ± 0.0206</td>
<td>0.6262 ± 0.0168</td>
<td>0.6715 ± 0.0215</td>
</tr>
<tr>
<td>MRR</td>
<td>0.5353 ± 0.0240</td>
<td>0.5855 ± 0.0213</td>
<td>0.5917 ± 0.0217</td>
</tr>
<tr>
<td>( H )</td>
<td>0.5150</td>
<td>0.6052</td>
<td>0.6291</td>
</tr>
</tbody>
</table>

Retrieval performance with POS-filtering, lemmatization and noisy-channel spell checking (see section 2.2.4) enabled are summarized in the following table.

<table>
<thead>
<tr>
<th>Cutoff</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>0.4954 ± 0.0219</td>
<td>0.6306 ± 0.0242</td>
<td>0.6773 ± 0.0277</td>
</tr>
<tr>
<td>MRR</td>
<td>0.5353 ± 0.0261</td>
<td>0.5876 ± 0.0256</td>
<td>0.5939 ± 0.0257</td>
</tr>
<tr>
<td>( H )</td>
<td>0.5146</td>
<td>0.6083</td>
<td>0.6329</td>
</tr>
</tbody>
</table>

Since the spell checker is configured to correct ‘misspelt’ terms to lemmata of ‘existing’ terms, no evaluation was performed without POS tagging. Also, no formal evaluation of
the other spell correction techniques is presented, but preliminary experiments suggested that retrieval performance degraded considerably when any of them was turned on. The maximum edit distance was fixed at one; raising it to two makes no difference whatsoever. Again, retrieval performance can be seen to improve slightly, but not enough to be statistically significant.
5 Conclusion

We have seen how a corporate contact center’s incoming email messages may be integrated with known information about product defects by the use of standard document retrieval models and algorithms. In addition, we have seen how, and to what extent, techniques from the field of natural language processing (NLP) may be used to improve the robustness of the resulting software system.

The first problem to be solved was how to coerce contact center data into the query/document objects that information retrieval systems are expected to process. Exploiting the linking of data was shown to be best way to achieve this, though, counterintuitively, the full text of email messages to and from the contact center had to be omitted to achieve optimal results.

As regards NLP, POS tagging, term filtering based on POS (rather than the traditional stoplist) and lemmatization were shown to have positive effects, even in the face of multilingual data and tools designed only for English text. This, however, points to another important conclusion: language guessing, sometimes considered a ‘solved’ problem of NLP, broke down very quickly on our dataset, limiting access to more sophisticated, language-specific methods (or even to language-specific stemming).

Lastly, an automated spelling correction routine was shown to provide small retrieval performance benefits. However, these results were not proven statistically significant.

5.1 Notes for practical implementation

5.1.1 Interface

The only user interfaces to the software developed in the course of this research were a command-line interface for experimenting and measuring, and a web-based prototype application. The latter is connected to the Lucene search engine software and allows custom queries to be executed, but cannot be used to store new information in the search index. A practical implementation would include links to the incoming email system to obtain indexable information and would add these to the index, as well as a user interface integrated with the email UI. This interface should automatically search for related incidents/KEs upon the arrival of a new incident report, allow the user to check whether linkage has been performed correctly (and perhaps issue a custom query if linkage is incorrect) and then update the index appropriately. Fast index updating routines are included in Lucene; however, as in the current setup updating the index means altering a document (by appending the text of a new incident report), some form of locking will have to be implemented.
5.1.2 A note on the spellchecker

The spellchecker described in section 2.2.4, which maps unseen terms to the Damerau–closest known terms, was shown to have some favorable results on retrieval performance. It should be noted, however, that data sparsity is one of the reasons that this approach actually works: as more terms come in, the number of term \textit{types} tends to increase, so the probability of an actually misspelled word being labelled as unseen decreases. Particularly, since we are dealing with an ever-growing index of incidents, a misspelled term may be stored in the index, leading its next occurrence in a query to likely ‘trigger’ its retrieval.

One way of dealing with this problem is to only store spellchecked versions of incident reports. However, this would require manual intervention as the spellchecker will reject all \textit{hapax legomena} and replace them by Damerau–close terms, if such exist.\footnote{As an example, consider the introduction of a FooWidget+ product as an upgrade to an existing FooWidget. Any occurrence of the name FooWidget+ in an incident report will lead the spellchecker to erroneously replace it with FooWidget.} A method for tackling this problem would be to consider not only unseen terms, but also terms with a small corpus frequency \textit{k} as eligible for spelling correction. It may be wise to periodically increment \textit{k}, though not very often.

5.2 Suggestions for further research

The main problem faced in this research is the extremely noisy nature of the data. This noise is expected to increase when the methods developed are taken into production, as incident reports not linked to any KE have been filtered out for the sake of formal evaluation; in production use, all incident reports will have to be taken into account, including those that do not actually describe a customer problem. Further reduction of this noise may be possible by filtering out these ‘non-incidents’: a method for doing so, using techniques similar to those proposed here, is described by Coussement and Van den Poel\footnote{Coussement and Van den Poel (2008).}.

As stated above, language guessing did not work for our problem, which involves multi-language documents and very short documents. The former is acknowledged by Hughes et al.\footnote{Hughes et al. (2006).}, who states that there is no known working solution for this problem yet. One can envision such a solution based on any of the sequence tagging models that have been developed in NLP over the past decades (HMMs, CRFs, etc.), but developing a general solution is likely to require much research.

In the current approach, part-of-speech tagging is only used to provide a ‘smart’ stop word filter, that responds to words of a certain POS rather than simple base forms. Instead, parts-of-speech can each be given a weight (in Lucene terms, a ‘boost’) with which to multiply the \textit{tf–idf} weight. Such weights can either be determined manually or automatically learned; Tiedemann\footnote{Tiedemann (2006, 2007).} describes such a weight-learning scheme in the context of a Lucene-based question answering system based on genetic algorithms.
Bibliography


