REUSING PAST REPLIES TO RESPOND TO NEW EMAIL: A CASE-BASED REASONING APPROACH

I Wayan Sathya Linggawa

A thesis submitted to Auckland University of Technology
in partial fulfilment of the requirements for the degree of
Master of Computer and Information Sciences

March 2017

School of Engineering, Computer, and Mathematical Sciences

Supervisors:
Dr Muhammad Asif Naeem
Associate Professor Russel Pears
Dr Gerald Weber

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Signature of candidate

Acknowledgements

Firstly, I would like to acknowledge my supervisors, Dr Muhammad Asif Naeem and Associate Professor Russel Pears, for their guidance, support, and feedback during my thesis journey. I appreciate the trust and flexibility given to me to explore the topic I enjoyed the most. I also would like to thank Dr Gerald Weber and Dr Christof Lutteroth for their great insights in each discussion to improve my work, particularly during the prototype development.

This thesis, and my whole study journey in AUT, would not have been possible without the support from the New Zealand ASEAN Scholarship (NZAS). I would like to express my gratitude for the financial support throughout my study.

Special thanks to Dessy Ariyanti who provided useful insight in organising my experiment results; Nindita Soenarso who helped me by being my pilot test participant; Fatthy Amir for advising me on the early stage of my research journey; Shelley Lodge, Titaningtyas, and Ridho Rizki for proofreading my thesis; and the AUT Scholarship Office team (Sacha, Ruth, and Margaret), who enhanced my study journey with the kiwi experience. I would also appreciate all the support and fun shared by my fellow friends in Auckland and Indonesia.

Finally, I would like to express my profound gratitude to my parents for their tremendous supports and prayers throughout the process of writing this thesis. Thank you for always encouraging me to pursue my dream to study in New Zealand; a childhood dream that sparked from a scoop of vanilla ice cream.

Abstract

Email communication has been widely used in managing customer queries, such as complaints and inquiries. The user is expected to respond the query properly. However, with an increasing number of query emails received every day, it seems likely that the inbox appears to have more unreplied queries and leads to overwhelmed and cluttered email management called *email overload*.

This thesis presents a development in the email overload issue on managing a reply task. The system, called a smart email client, helps in replying to an email by suggesting a list of replies gleaned from the emails replied to in the past. These suggested replies are ranked according to the level of similarity. The methodology follows the Case-Based Reasoning (CBR) approach to solve the problem by reusing previously written solutions from the past replies stored in the case base. Techniques from Natural Language Processing and Information Retrieval are utilised, particularly in lexical semantics, through using WordNet. Finally, an evaluation of the retrieval algorithm shows that the effectiveness and efficiency of the algorithm is influenced by the case feature selection and various text analysis techniques such as lexical analysis, stemming, stopwords removal, and synonym expansion.

Contents

Attestation of Authorship **2**

Acknowledgements **3**

Abstract **4**

1 Introduction **9**
1.1 Thesis Contribution 12
1.2 Thesis Structure 13

2 Related Literature **14**
2.1 Email Client 14
2.2 Email Overload 17
2.3 Responding to a New Email 19
2.3.1 Automatic email answering or template generation 19
2.3.2 Predicting reply action 22
2.3.3 Reusing previously authored reply message 24
2.4 Case-Based Reasoning 26
2.4.1 CBR Cycle 27
2.4.2 Categorisation of the CBR System 32
2.4.3 Textual CBR for text-based problems 34
2.5 Related tasks and techniques in textual CBR 35
2.5.1 Information Retrieval 36
2.5.2 Natural Language Processing 38
2.5.3 Document Preprocessing 40
2.6 Summary 46

3 Design and Implementation **47**
3.1 Overview of Smart Email Client 47
3.2 System Architecture 49
3.3 Functionality Design 51
3.3.1 Email Communication 52
3.3.2 Information Retrieval (IR) 54
3.3.3 Graphical User Interface (GUI) 55

3.4 Adapting the CBR in Smart Email Client 57
3.4.1 Process Flowchart 57
3.4.2 Case Representation 58
3.4.3 Case Base Population 59
3.4.4 Case Retrieval 63
3.4.5 Case Reuse 64
3.4.6 Case Revision 65
3.4.7 Case Retain 66
3.5 Implementation Results 66
3.6 Restrictions and Limitations 74
3.6.1 Logic 74
3.6.2 User Interface 75
3.7 Summary 75

4 Evaluation **76**
4.1 Evaluation Configuration 76
4.1.1 Evaluation methods 77
4.1.2 Evaluation metrics 78
4.1.3 Dataset description 80
4.1.4 Experiment configuration 83
4.2 Results 84
4.2.1 Dataset distribution 84
4.2.2 Experiment results 86
4.3 Discussion 93
4.3.1 Influence of text analysis and case feature selection 93
4.3.2 Dataset critique 98
4.4 Summary 100

5 Conclusions **102**
5.1 Conclusions 102
5.2 Future Work 104

List of Tables

2.1 Overview of various approaches in responding to a new email . . 19
2.2 Methods and techniques in the "automatic email answering" approach 20
2.3 Methods and techniques in the "predicting reply action" approach 22
2.4 Methods and techniques in the "reusing previously authored reply message" approach 24
2.5 Positioning of our approach (appears in bold) based on the CBR system categorisation by López (2013) 32
2.6 An example of a stopwords list obtained from Fox (1989) 41

3.1 Case representation from the email corpus in database 58

4.1 A query and its annotated relevant response from farmer-c mailbox 81
4.2 Datasets used in this evaluation 82
4.3 Template used to record experiment results 83
4.4 MRR score from experiments in both dataset 87
4.5 Total terms indexed from experiments in both dataset 91
4.6 Lexical analysis: words are treated equally by normalisation . . . 94
4.7 Advantages of stemming: more discovery of matching terms . . . 95
4.8 Advantages of stopwords removal: reduced number of terms indexed 96
4.9 Synonym expansion: extending the capability of exact matching . 96
4.10 Expanding terms with their corresponding synonyms in the content 97
4.11 Retrieval results using "subject"; italic indicates matching keywords 98
4.12 Summary of messages with less meaningful context 99
4.13 Given relevance judgment compared to our retrieval results; italics indicates matching keywords 100

List of Figures

2.1 The CBR Cycle adapted from Aamodt & Plaza (1994) and Watson (1999). 27
2.2 Vector Space Model as introduced by Salton et al. (1975) 44

3.1 Flowchart of the system 48
3.2 High-level architecture view of the email client 50
3.3 Proposed functionality layers in the application 51
3.4 Table definition in the email client application 53
3.5 The user interface layout forms an F-pattern 56
3.6 Look and feel of the application using JavaFX library 56
3.7 Process workflow adapting CBR methodology 57
3.8 Flowchart of indexing case base 60
3.9 Case reuse 64
3.10 Case revision 65
3.11 Case retain 66
3.12 The start screen user is asked to select a message 67
3.13 The unreplied to message screen and two options for replying. . . 68
3.14 Reply window using normal reply 69
3.15 Implementation of case retrieval in the application 70
3.16 Implementation of case reuse in the application 71
3.17 Implementation of case revision in the application 72
3.18 Implementation of case retain in the application 73

4.1 Diagram showing summary of our experiments process 79
4.2 Luke: An open source tool to read Lucene index files 85
4.3 Dataset distribution according to Zipf's Law distribution 86
4.4 Retrieval effectiveness as represented by MRR score 88
4.5 Average similarity score over top 10 results in germany-c dataset . 89
4.6 Average similarity score over top 10 results in farmed-d dataset . 89
4.7 Similarity score increase rate before (SA) and after (SE) applying synonym expansion with respect to case features body and subject 90
4.8 Index size as represented in a count of terms indexed 91
4.9 Influence of synonym expansion for term distribution in germany-c 92
4.10 Influence of synonym expansion for term distribution in farmer-d 92
4.11 Retrieval efficiency as represented in processing time elapsed . . . 93

Chapter 1

Introduction

Email is still one of the most preferred communication channels., which is reflected by the fact that in 2015 that the number of email accounts reached 4.3 billion, a figure that is expected to grow by 6-7% in the next four years (Radicati, 2015). In the same period, there were 2.5 billion email users, an annual growth of 3%. That indicates that a user may have at least one email account, with an average of 1.72 email accounts per user. In addition to the daily traffic, 112 billion email messages are sent and received daily (Radicati, 2015).

While these overall statistics show the importance of email, it is also interesting to see the statistics of user activities. According to the statistics presented in Radicati (2015), on average 122 emails are sent and received by a user daily. 72% of these messages are received by a user, and 28% of the messages are sent emails. Assuming that the received email messages are queries, the user is expected to manage the workflow to respond to the query properly as otherwise, it may result in a personal information management issue called *email overload* (Whittaker & Sidner, 1996).

Over the past years, managing email messages has widely been done through an application called *email client*. This application originally had the main role of accessing the user's mailbox in a mail server. However, as email communication evolved and became more complex, an email client became capable of dealing with task management applications, as well as personal archiving (Whittaker & Sidner, 1996). One of the components of task management is related to how a user can receive a message (which may come in the form of a query), and how they respond to that message, (in other words, providing the solution in reply to the query). Due to the increase in incoming emails (or queries) every day, the user needs a more efficient workflow to respond to the query. Several studies have attempted to develop an automatic email response or to generate a template (Kosseim et al., 2001; Weng & Liu, 2004; Malik et al., 2007; Sneiders, 2016b). Other approaches have focused on indicating whether or not an email needs a reply (Ayodele & Zhou, 2009; Dredze et al., 2008). The latter approaches however, still involve user control more than the earlier ones do.

Motivated by the explicit usage of previously authored information to reply to the message, another study suggests that the incoming query may have been already answered in the past (Lamontagne & Lapalme, 2003; Hewlett & Freed, 2008). Therefore, a user could benefit from reusing his/her own past reply message if the query is similar. This may improve the experience of responding to an email, since the user does not always need to start with a blank message.

From the perspective of computer science, the notion of understanding problems in the form of user query, then solving this, has triggered the implementation of Natural Language Processing (NLP) and Information Retrieval (IR). While past studies show an extensive use of rule-based approaches, such as keyword utilisation to categorise email messages, NLP focuses on understanding human language and processing it in a way that computers can understand. IR

attempts to retrieve documents based on the user's query; and therefore benefits from using NLP (Liddy, 2001).

Enhancements in retrieving documents based on the user query can be achieved by implementing additional knowledge related to that domain. Some studies suggest that providing a domain-specific ontology may improve the chance of finding more relevant results (Lamontagne & Lapalme, 2003). However, this specific knowledge might not be useful, since the context of the terms might not be relevant to the general domain. Another approach adopts language-based knowledge to accommodate lexical semantics by understanding words such as synonyms (Hliaoutakis et al., 2006; Abdalgader & Skabar, 2010). The problem-solving paradigm has also evolved over time, and recently Case-Based Reasoning (CBR) methodology has been introduced. CBR solves a problem by retrieving similar past problems from the collection (knowledge base), referred to as the case base, and reusing its solutions for the new problem. This approach is commonly known as *lazy learners*, as it performs the function of retrieving similar knowledge at the time the query is presented instead of first building training model.

The notion behind CBR methodology is a motivation to adapt this approach to solve the email overload problem. CBR has its own sub area, identified as textual CBR, where the problem is presented as textual information. The textual CBR framework benefits from the utilisation of text processing techniques, such as stemming and stop words removal, as well as specific domain knowledge elicitation (Weber et al., 2005).

Our proposed email client solution is a desktop application that has standard functionality for email communications and an enhanced capability to intelligently provide reply recommendations. The application starts performing when a user initiates the reply action to a new incoming message. It then attempts to

retrieve similar past incoming emails (known as cases), reusing the answers and treating them as a reply recommendation list. The process is finished when the user selects a result from the list and sends the email. The answered email is considered as a problem solved and is stored in the case base.

There are two main objectives that we have attempted to accomplish. The first was to build a prototype of an email client which has the basic functionality to send and receive email messages from the mail server. The second is to assist users through a reply recommendation list, based on their past replies, so they can reply to an incoming email.

There are two challenges that we have identified in this research. The first challenge is to apply the CBR methodology to the problem domain, the email reply behaviour. We discuss how the case for CBR is represented and processed in the CBR cycle.

The second challenge is to build a retrieval algorithm capable of finding similar cases beyond the exact matching. We attempt to explore recent studies in this area, including a strategy that uses the most useful case feature in the email message, and the use of a semantic component in the text analysis (Hliaoutakis et al., 2006; Abdalgader & Skabar, 2010).

1.1 Thesis Contribution

This thesis makes the following contributions:

1. A comprehensive literature review of current developments in the area of email response processes.

2. A Java-based prototype of a smart email client with reply recommendation system, which implements the CBR approach.

3. The development and evaluation of a retrieval algorithm, with improvements using text processing techniques and semantics analysis using word synonyms.

1.2 Thesis Structure

The structure of this thesis is organised as follows. **Chapter 2** begins with a discussion of past research in this field and provides background information about the study, such as potential areas in Natural Language Processing and Information Retrieval. It also describes the CBR approach using the 4R's (retrieve, reuse, revise, and retain) model. In addition, various text processing techniques and algorithms that could be used in the development are examined.

Chapter 3 describes the design thinking process for the proposed solution, along with the justification. This includes the components and tools utilised in the development of the application. Moreover, the implementation phase, which adapts the CBR methodology, is also described. It starts with building the case base, performing a retrieval strategy, reusing and revising potential solutions, then retaining the solved problem in the case base.

Chapter 4 discusses the various evaluations of the CBR system. These evaluations include examining the effects of text processing techniques, identifying which case feature is the most useful for retrieval of similar solutions, and analysing the dataset with respect to the results. The datasets used in this evaluation were obtained from a publicly available email dataset: the Enron corpus.

Finally, **Chapter 5** concludes the thesis, as well as describing some directions for further work.

Chapter 2

Related Literature

This chapter provides a basic understanding of the problem investigated in this thesis. We examine current developments in email communication and identify gaps in it. We also discuss the related areas that have been studied in order to solve the problems. Finally, we specifically identify potential techniques in order to build our solution.

2.1 Email Client

The mail system can be distinguished into two subsystems (Partridge, 2008). The first has the role of moving email messages from the sending user to the receiving user. This subsystem is called the message handling system (MHS) and is built on a set of message transfer agents (MTA). The other subsystem, the user agent (UA) or mail user agent (MUA), has roles related to user activity, such as receiving, managing, and creating email messages, as well as sending them via MHS. Therefore, considering these roles, the MUA can be further referenced as an email client.

An email client is an application that enables the configuration of one or more remote mailboxes, allowing the user to receive and send email through a mailbox. It has been widely developed as a desktop application and is available off-the-shelf. The email client also has been implemented in mobile phones alongside the native operating system. Moreover, most email clients have been developed to address the limitations of a web-based mailbox interface by storing the messages offline.

Recently there has been an increase in the use of remote mailbox service providers to support email communications such as Gmail[1], Yahoo[2], and Outlook[3]. In addition to providing remote mailbox storage, they also have a web-based interface to access the email in the mailbox. While this service requires the user to be connected to the internet, the problem arises when the internet connection is not presented. This is one of the issues that has motivated the development of a non web-based email client.

As mentioned previously, the main use of an email client is to compose the message and retrieve it from the mailbox. Two methods are used for message retrieval: Post Office Protocol (POP) and Internet Message Access Protocol (IMAP) (Partridge, 2008). The first method allows the user to download the message then delete it from the mail server after it has been successfully stored offline, while the latter implements enhancements through a flagging capability on the messages. This flag allows a message to be marked read or unread. Typically, the email client will have a POP or IMAP configuration, based on the mailbox service of the user.

To compose a message, a typical email client usually has an interface to input and edit the text content. The message consists of three parts: headers, the body,

[1]https://www.google.com/gmail/
[2]https://mail.yahoo.com/
[3]https://outlook.live.com/

and non-textual contents and attachments commonly known as Multi-Purpose Internet Mail Extensions (MIME). This formatting accords to the standard internet message format, RFC5322[4]. The headers include information about the origin and destination of the message, such as *To, CC, BCC, From,* and *Reply-to*. There is also a configuration for the date and time of the message in RFC5322, which is not comprehensively explained here. It is important to carefully consider the format of the email message, since it ensures the message will be delivered to the mail server.

In addition to message formatting, an email client submits the message to the mail server by using Simple Mail Transfer Protocol (SMTP). The standard for SMTP specification is also available in RFC5321[5]. As with message retrieval, the email client needs to know the SMTP configuration of the mailbox being used.

As daily tasks become more complex, the main function of the email client has shifted from managing email communications to personal task management. Features such as contact management and personal calendars have been widely implemented in modern email services (Grevet et al., 2014). One study also attempted to enhance the capability of an email client by providing more visualisation, text analysis, and attention management functionality (Rohall et al., 2004). These implementations were motivated by an issue called *email overload* (Whittaker & Sidner, 1996), because users were expected to manage responses to the high volume of email received daily. In addition, these emails may appear in the form of a complaint, for instance, which needs an immediate response (Coussement & Van den Poel, 2008).

[4]https://tools.ietf.org/html/rfc5322
[5]https://tools.ietf.org/html/rfc5321

2.2 Email Overload

The term *email overload* was introduced by Whittaker & Sidner (1996) as a result of observing the cluttered user email phenomenon in 1996. Initially, email aimed to establish asynchronous communication where a specific action is taken after the recipient receives the message. However, Whittaker & Sidner (1996) argue that the actual use is beyond communication; it also acts as a task manager and a personal archive. The first role is related to how email can provide information related to the task as well as showing the current status of the task. The second role is about how the message is organised by the user so it can be easily retrieved when needed.

Whittaker & Sidner (1996) also explain in their study that people tend to receive a large volume of incoming messages, as email communication acts as a source of office tasks. This phenomenon can leave the task unfinished, the message partially read, or the correspondence partly composed and not ready to be sent. It could be useful therefore, to focus the study on assisting the user to deal with those tasks.

A decade later, Fisher et al. (2006) conducted a study on the email overload problem by reinvestigating the work of Whittaker & Sidner (1996). They identified that in the 10-year time frame, email functionality had not changed much but people's email archives had grown significantly. People also tended to use more folders to organise their emails, while in contrast, some cleaned their inboxes daily. In the end, they concluded that in the coming years, email overload would still occur due to the fact that the variety of content retrieved would become more extensive (e.g document transfer or RSS feeds). Although the current study has shown significant findings in terms of revisiting the previous research, the scope of the experiment is limited to one technology company.

Therefore, the study would be more useful if it could distinguish the various type of email users from different backgrounds since they would have different behaviours.

Further, research conducted by Grevet et al. (2014) continued the study of the email overload issue, particularly related to Google's Gmail service. They argue that the issue still arises because the volume of incoming email is increasing. In addition, email overload has taken a new form by affecting both personal and work environments. Promotional mail and billing are examples of personal email, while work emails deal with the status of work, such as to read or to do.

A study conducted by Grevet et al. (2014) makes a strong point by investigating the personal email environment, while both Whittaker & Sidner (1996) and Fisher et al. (2006) mainly focused on work email. As a result, the author has identified that one indication of email overload arises from the large volume of unwanted emails, which are further defined as 'spam' (Grevet et al., 2014). In addition, the various backgrounds of the users in the research could provide different personal experiences when utilising their personal email.

In previously explained studies, it seems that one of the keys to reducing the email overload issue might be to encourage the user to respond to emails that needs action as soon as possible, thus helping the user to be better organised. In addition, ease of access in managing repeat inquiries such as customer complaints in the customer service domain (Coussement & Van den Poel, 2008), could be another interesting objective. According to Whittaker & Sidner (1996), the concerns relate to task management and asynchronous communication. Therefore, an intuitive solution could be developed by focusing on these two functionalities.

2.3 Responding to a New Email

The previous section revealed that one of the challenges in email communication relates to defining a strategy on how to respond to the high volume of incoming email. This usually arises in customer-related domains such as help desks, event management or customer service (Coussement & Van den Poel, 2008). Most of the time, users may have already replied to a similar inquiry. However, they could be spending time browsing through their previous sent emails in order to obtain the respective reply (or solution), a time-consuming and frustrating process. Several studies have attempted to assist users in managing email responses, as summarised in Table 2.1.

Table 2.1: Overview of various approaches in responding to a new email

Approach	**Author**
Automatic email answering or template generation	Kosseim et al. (2001), Weng & Liu (2004), Malik et al. (2007), Sneiders et al. (2016)
Predict incoming message whether or not it needs a reply	Ayodele & Zhou (2009), Dredze et al. (2008)
Reuse past replies to respond to a new incoming message	Lamontagne & Lapalme (2003), Hewlett & Freed (2008)

2.3.1 Automatic email answering or template generation

One method of managing an email reply is by implementing an automated email answering. The extensive review by Sneiders (2016a) shows that the main approaches for this method are by using machine learning techniques. A summary of various past studies is shown in Table 2.2.

Kosseim et al. (2001) conducted a study that arose from the typical task of answering an email inquiry: recognising the content (usually a problem) and

Table 2.2: Methods and techniques in the "automatic email answering" approach

Author	**Method**	**Technique**
Kosseim et al. (2001)	Information Extraction	Text processing, Rule-based extraction, Decision tree
Weng & Liu (2004)	Classification, Clustering	Text processing, Term weighting TF-IDF, Cosine similarity
Malik et al. (2007)	Clustering	Text processing
Sneiders et al. (2016)	Pattern Matching	Text processing, Rule-based matching

generating the reply response. Since the domain is customer support, they identified the particular categories that are specifically available in that domain, such as a "how-to question", "suggestions", "problem reports" and others. To analyse the message text, an information extraction process is undertaken to identify specific information and re-present it in the structured format. Tokenisation and other lexical analyses are performed to group part-of-speech and phrases. The text then fills the prepared templates used in the discourse analysis. Response formulation is prepared afterwards and typically has a structured content, such as beginning with a salutation (and the name of the customer) and ending with a formal closing.

Malik et al. (2007) proposed a similar concept of preparing predefined templates to answer customer queries. This idea arises from an understanding that in a set of query-response email pairs, the association between the question and answer can be identified. This association would be useful to map similar future questions to its answer templates. The system is trained to identify earlier a large number of email message pairs in the archive in order to learn to classify them into standard answers. In mapping new email queries to the old ones in the archive, this system employs WordNet language reference. The system performs at 61% capability compared to a human-performed task.

Weng & Liu (2004) identified an approach that focused on providing a set of reply templates based on the incoming email content. Concepts, as a collection of terms, were extracted from the body of the email and indexed. The main contribution of this approach was finding out that indexing only the concepts could make the search for the associated reply template faster. The best matched reply templates were then given to the users. However, the knowledge base for a reply template needs to be defined before the system can associate them with an incoming email. In addition, the recommendation only considers a template calculation score, which came from the terms that appear more frequently.

Extensive observation from Sneiders et al. (2016) suggests that an inquiry in an incoming email can be separated into two parts: the context or description of the problem, and a request to resolve. These two elements are important in delivering the most appropriate reply generation. In this study, instead of generating answer templates, the authors designed their own reply pattern, which was manually designed and consists of syntax and regular expression. In an experiment, we demonstrated text-pattern matching to the inquiry, before assigning it a standard answer. Also, we identified that the highest misclassification occurs because of the unique wording of the inquiry, which may not match a manually prepared text pattern. Other useful findings relate to missing synonyms, no subject in the message, and misspellings.

While providing an intensive review of automatic email answering, Sneiders (2016a) also argues that business nowadays does not commonly practise automated email answering. This happens because of the fear that they will lose contact with their clients, leading to lost opportunities. This perspective aligns with our study; we realise that users still want control over the actions they perform. Therefore, the solution we propose should be able to put user control one step above the automated tasks.

In addition, we can see from the recent studies that most attempted to develop solutions based on the rule-based approach. This approach has the advantage of utilising set of predefined rules to build a strong knowledge base for performing the automation process. However, the main drawback of the rule-based approach is that it requires an extensive knowledge base before it generates results. In some newly established recommendations or automation systems, a robust knowledge base might not be present. Thus, the solution is expected to work well, even though the knowledge base has less or minimal information.

2.3.2 Predicting reply action

Automatic template generation may be useful for an efficient reply to a homogeneous query. Often, the user needs to be assisted in a way that means they can have more control over the replying process. In this case, some of the studies shown in Table 2.3 suggest that by identifying the structure and content of an email message, a prediction of whether a message needs a reply or not can be made.

Table 2.3: Methods and techniques in the "predicting reply action" approach

Author	**Method**	**Technique**
Ayodele & Zhou (2009)	Clustering	Text processing, Rule-based prediction
Dredze et al. (2008)	Classification	Text processing, Term weighting TF-IDF, Rule-based prediction

Ayodele & Zhou (2009) conducted a study on generating reply prediction using the machine learning technique. This study was part of the email management system that the author has previously used (Ayodele & Zhou, 2008). The main idea is to generate a prediction about whether or not an incoming

email message requires a reply in a form of labelling. The prediction appears as the result of a scoring mechanism calculation based on the email subject and content extraction. Content extraction is examined based on several indicators, such as interrogative words, question marks, domain recognition of the sender's email address, attachments, and most used phrases. Then, each email is given a label of "definitely needs reply" or "needs no reply" and shown to the users according to the score.

Ayodele & Zhou (2009) also show that their solution could assist the users to notice which email is prioritised to be replied to. However, it is not clear in their study how they perform the text processing, other than finding the interrogative words. Also, the authors did not explicitly state whether they considered the body content of the email.

Dredze et al. (2008) developed a prototype for predicting whether a reply is necessary for an email message. This work differs from that of Ayodele & Zhou (2009), since it includes a prediction for the attachment too. The reply prediction provides information on whether a message requires a reply by establishing a rule-based system using reply predictors. The predictors include the user profile (e.g send/received statistics and address book), as well as the presence of the words "reply" or urgent, and of question marks in a message. Afterwards, the attachment predictors benefit from the discovery of the words "attach", "attaching", "attachment", or "attached".

Dredze et al. (2008) also argue that the users have control over the system, since they are being assisted when taking the action of replying to the most important email first. When the reply message contains the keywords mentioned previously, they could be alerted to attach documents instead of missing this. The prototype is usable as it has been implemented in the Mozilla Thunderbird email client as an extension.

While previous studies have demonstrated the usefulness of features for user control, as in providing labels and alerts, the system employs a rule-based approach under the hood. Therefore, collecting sufficient training data to build the knowledge for classifying the email is necessary to optimise the classification process.

2.3.3 Reusing previously authored reply message

Some of the studies shown in Table 2.4 extend beyond automatic reply and template generation, and predicting the reply. They realise that a new incoming email forming a query (or problem) may have been replied to in the past. Therefore, the exploration and utilisation of past messages are identified, based on the notion that similar problems may have similar answers.

Table 2.4: Methods and techniques in the "reusing previously authored reply message" approach

Author	**Method**	**Technique**
Lamontagne & Lapalme (2003)	Case Based Reasoning	Text processing, Rule-based extraction
Hewlett & Freed (2008)	Classification	Text processing

Sneiders (2016b) demonstrates the use of the term co-occurrences to increase automation in a company's website contact page. The idea is to use previously published information when the user types the message, so when it is available, the user will be directed to the pages that contain the information. He argued that his approach could save the user's time and the company's resources as the solution may already be there for the user. While this approach is novel when matching email-style messages to web pages or documents, it is different to our approach, which matches an email query with a collection of past email messages, not web pages.

Lamontagne & Lapalme (2003) developed a solution to deal with customer email messages sent to a company. The goal is to adapt previous email messages in order to reply to a new request. To achieve this, the system compares the new request message with a collection of past messages in order to find the most similar one. When the most appropriate past message is found, based on the calculation the system attempts to reuse the corresponding answer message to reply to the request.

The study conducted by Lamontagne & Lapalme (2003) was also part of the Mercure project (Lapalme & Kosseim, 2003). This study is domain-specific, notably in terms of investor relation messages. Thus, the authors explain that no domain specific resources are present. They did not benefit from the existing linguistic resources such as WordNet. However, by expanding the query into term co-occurrences, they benefitted from a slight increase in the result.

Although providing reply predictions and templates could minimise the users' efforts to reply to similar emails, an extensive training model with a high volume of samples could be mandatory to achieve better performances. This investigation, however, suggests that user intervention could help to provide a better recommendation process by giving feedback on the most relevant results (Hewlett & Freed, 2008). Feedback was given after the system performed a search for the most similar sent emails. A higher score could be achieved when the users vote that particular result to be the most appropriate reply. Hence, the system also learned about user preference.

Our work is closely related to that of Lamontagne & Lapalme (2003) and Hewlett & Freed (2008), as we tried to utilise past email replies to answer an incoming email inquiry. We realise that future questions could be mapped by utilising similar old questions and reusing the answers. This approach would still allow the user to have control of the task. Further, it could also reduce the

complexity of composing from scratch or searching through the whole mailbox.

2.4 Case-Based Reasoning

In past decades, a problem solving approach emphasising the possession of past experience has been intensively studied. The case-based reasoning (CBR) methodology was inspired by research from cognitive science on human memory (Schank, 1983), which reflects previous experience on solving a similar problem. The reasoning behind the proposed solution is much influenced by cases (collection of past solutions). Before the system can solve a new problem, several processes need to be performed.

A broader perspective on CBR implementation has been shown in recent studies by Osiński & Weiss (2005), Díaz-Agudo et al. (2007), and Stahl & Roth-Berghofer (2008). These authors developed a generic platform for building a CBR system that could be applicable to any domain. As a result, these works demonstrated the flexibility of their platform as it has been implemented in various fields such as the medical (Abdrabou & Salem, 2010), and the financial (Martin et al., 2012).

Although some off-the-shelf platforms for problem solving using the CBR approach are already available, they are mainly built for the structured problem domain. The problem can be separated into the domain-specific query with its particular terminology. In our study, the problem domain is an email message. The content of a message is considered unstructured, since every query could be different from another. Besides this, the length of the message also varies.

There is no silver bullet that works for every problem. This understanding also applies in the CBR problem areas, where there are no specific CBR methods that suit every domain. However, the general method of CBR was first established

by Aamodt & Plaza (1994) and has been widely used since then. Thus, the challenge is to tailor the method to suit for problem solving in a particular domain area.

2.4.1 CBR Cycle

Case-based reasoning as a framework for problem solving has two main components: the modelling process translated as a CBR cycle, and the task-method structure associated with each process in the model (Aamodt & Plaza, 1994). The diagramme in Figure 2.1 represents the high-level process of adapting the CBR approach. It is widely recognised as the 4R's: Retrieve, Reuse, Revise and Retain.

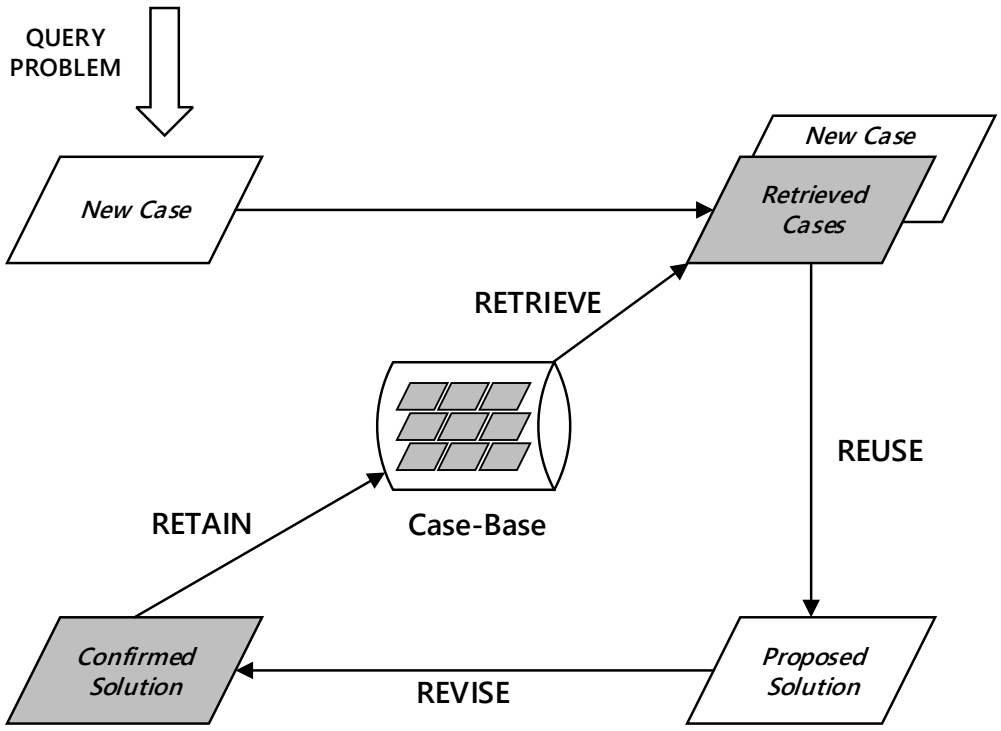


Figure 2.1: The CBR Cycle adapted from Aamodt & Plaza (1994) and Watson (1999).

A case typically consists of a problem description and its corresponding solution. When a query problem arrives, it is considered as a new case with a defined problem and no solution yet. Then an attempt is made to retrieve past cases from the memory (or case base) that have a similar problem description to the new case. Next, the solution to these past cases is obtained to be reused as a proposed solution. The system either automatically, or by using user intervention, attempts to revise the solution by considering the differences to the current problem. The proposed solution is evaluated by applying it to the initial problem, or assessed by the domain experts on its respective problem domain. Finally, the confirmed solution is retained in the case base to be considered for solving similar problems in the future.

A detailed explanation for each process in the CBR cycle is provided in the following:

Case Representation

In the CBR method, a proposed solution relies heavily on the presence of knowledge representation. The knowledge, specifically identified as the case, is a collection of past experiences that might be suitable to solve the problem. Further, Kolodner (1991) argues that the content of the case does not need to have a specific form, but can be focused on the kind of things (or information) that should be available. A proper formulation of the information can allow the case to be productively used for solving the problem. In addition, Kolodner (1991) suggests that a case can contain a problem description and its respective solution, with additional information on whether it worked (or might not work) in solving past problems.

A further CBR cycle is mostly associated with issues about how to organise the case storage for effective retrieval and reuse. Bergmann et al. (2005) identified some basic approaches to collecting cases, where some use textual representation or benefit from generalised cases. When a case is in a textual structure form, the text, such as words or phrases, is extracted into Information Entity (IE) (Lenz & Burkhard, 1996). IE allows the cases to be stored in the form of nets and nodes; thus the retrieval process can be automated by calculating the strength of the links between nodes. Meanwhile, a generalised case extends the capability of a single case solving single problem by providing a solution to closely related problems. This can be achieved because a generalised case can have various alternative plans to accomplish a common goal.

Case Retrieval

The retrieval process first begins with problem definition and finishes when the best match has been found. In particular, the retrieved cases are past experiences stored in the case representation. A set of cases that are most similar to a given problem are returned at the end of the process. Thus, extensive study has been conducted on finding an efficient similarity assessment technique (De Mantaras et al., 2005).

De Mantaras et al. (2005) argue that in some CBR implementations, a similarity assessment can be performed by examining the features of the case, either those that are provided in its description (surface), derived from an inference based on the domain knowledge, or represented by a complex structure in a form of graph. There is a trade-off between computational cost and the capability to retrieve more relevant cases, which places derived and structural examination as more resource consuming than the earlier. Alternatively, indexing

vocabularies (or special terms) to describe cases can help to reduce this extensive resource use.

With respect to the email problem domain, Lamontagne & Lapalme (2003) identify the structure of an incoming email message as the features that determine the problem. Then, named entities (a person's name) and a particular passage in the case base is considered in the matching and selecting of potential cases through the retrieval process. Furthermore, the cases returned in the retrieval process can be prepared for the next process: case reuse. The proposed solution could benefit from the pertained knowledge (or cases) stored earlier. As a result, instead of always composing a new blank reply, which is treated as a new case, Lamontagne & Lapalme (2003) argue that they could suggest the past case as a solution to the problem.

Case Reuse

Following the retrieval of a potentially identified solution, a CBR model attempts to reuse selected past cases. The task is mainly focused on determining the differences between returned past cases and the new case, as well as identifying which part of the old case can be used in the new case. Two approaches have been widely implemented: copying the details of a past case as the chosen solution, or adapting the past case to the context of the new case by considering transformation or derivation of information.

In terms of providing the response message to reply to an incoming email, adapting the past case was selected by both Lamontagne & Lapalme (2003) and Hewlett & Freed (2008). They selected only the most appropriate feature from the retrieved past case, which is the body of the email itself. While Hewlett & Freed (2008) present this task in the form of providing a recommendation to

the user, Lamontagne & Lapalme (2003) intelligently map some part of the body message to be further processed in the case revision step.

Case Revision

When the new case contains information from a previous case, revision can be made automatically or by an intervention from the user. Two main tasks are involved, namely evaluating the solutions and repairing the fault. The evaluation process is taken to avoid providing inappropriate solution. In addition, when the fault is found, it has to be repaired by considering the domain knowledge. Therefore, the goal is to generate a high-quality case to be retained in the next process.

Adapting past cases was selected by both Lamontagne & Lapalme (2003) and Hewlett & Freed (2008) in their studies in the context of providing the response message to reply to an incoming email. Although they use most of the content from a past email in the body part of the new email, the main difference is that Hewlett & Freed (2008) omitted some part of the message, while Lamontagne & Lapalme (2003) provided the entity name that was extracted previously and replaced a part of the message with a more appropriate reply content. Thus, both have the trade-off of simple implementation and a more personalised message.

Case Retain

This process is often referred as the learning process. As stated, the main goal of the CBR system is to store the solved problem in the existing knowledge base (or case base). Case retention starts by extracting the whole (or partial) information from the solution. This ensures the quality of the case stored. The process is followed by indexing, which includes a decision about what type of indexes

should be used for future retrieval. At the end, the case is integrated into the case base by maintaining either partial or whole information. The newly learned case may appear in the future if it matches a similar problem.

The nature of email conversation is a pair of query and reply sentences. Therefore, with the implementation of the CBR system that manages email response, the new case is considered as a query email associated with a reply. While studies by Lamontagne & Lapalme (2003) and Hewlett & Freed (2008) have provided a comprehensive description on how they implement case retrieval, reuse, and revision, it is not clear how they benefit from a successful new case that has been solved using a past case. They did not explicitly state the procedure of case retaining in their study.

2.4.2 Categorisation of the CBR System

There has been a significant development in various CBR systems. López (2013) categorises the differences based on four criteria (or dimensions): the source of knowledge, function, organisation and distribution. Although it is not an exhaustive list of every CBR system development, we found it useful to see the positioning of our CBR system in real world implementation. Thus, we attempted to examine our email client system based on the criteria previously mentioned, as summarised in Table 2.5.

Table 2.5: Positioning of our approach (appears in bold) based on the CBR system categorisation by López (2013)

Knowledge source	Function	Organisation	Distributiveness
Textual	Classification	**Sole**	**Single memory**
Structural	**Recommendation**	Multiple level	Multiple memories
Conversational	Tutoring	Hybrid CBR	**Single agent**
Temporal	Planning	Meta CBR	Multiple agents
Images	Monitoring		
	Knowledge management		

The first dimension identifies what the form of the case is. Some CBR systems have their source in the conversation between the user and the system, the interpretation of images or even in the information from temporal relationships such as user action history in a game. Furthermore, this information can be stored in predefined variables for a specific domain such as the medical field. Our system relies extensively on the presence of text in the email message, and this textual collection is easily obtained.

The second dimension groups the CBR system based on the function of its development. It is common to see a CBR system mainly utilised for a classification task, where it can predict labels or classes in a binary (positive or negative) or discrete (multiple) manner. A CBR system also assists in the processes of a development life cycle such as knowledge management, planning, and monitoring the deviation in the behaviour of a system. In some cases, it can also provide recommendations according to user preference and interaction in the system. Our system attempts to provide this recommendation based on information acquired from the user's past responses in replying to messages.

A CBR system can be combined with other knowledge-based systems or even another CBR system that has a different functionality to solve a problem. This combination may involve a multilevel organisation used by several CBR systems, hybridised with other problem-solving methodologies, or utilise meta information acquired by another CBR system to reason the best method to apply at every CBR stage. On the other hand, it is also common to use a single CBR system for problem solving. In our case, it is sufficient to only use a single CBR system, as the nature of our task is not too complex.

Finally, a CBR system can be categorised by the number of the case-bases (or memory) and what the processing distribution is. It can have one case-base or many, and the processing can be done by either single or multiple

agents in a system. Multiple distributions may be used for building case-bases to solve complex problems, or for sharing processes that are computationally expensive. Our system focuses on a relatively simple process, which involves finding and recommending email messages from our case-base. In addition, it is not necessary to build multiple case-bases since a single case-base already represents information storage in an email account.

2.4.3 Textual CBR for text-based problems

In recent years, there has been an emerging interest in exploring CBR that specifically deals with text problems. This is possible since CBR itself is a methodology (Watson, 1999); thus it is open to new problem domain discoveries, such as text. Experience and information from past problem solving processes is retained and explicitly reused to deal with new tasks or problems. This extension is commonly referred as *textual CBR*.

One of the typical areas that implements textual CBR is the customer support domain. Normally, this support service relies heavily on text-based documents such as error reports, technical documentation, or frequently asked questions (FAQ) to solve customer problems. The problem itself may not necessarily be solved within seconds, but the support staff needs a system to assist them in finding the documents that are relevant to a customer's query. In addition, a specific technique to retrieve information based on natural language text in a customer query is mandatory. Further explanation of this technique is presented in the next section.

Since CBR is a knowledge-based approach, it is important to note that a textual CBR system needs knowledge (or case) representations in any or some of the following categories: case collection, index vocabulary for each case,

similarity measurement, and knowledge adaptation (Lenz et al., 2003). First, text-based documents can be utilised as the source of the case base. Then, the index is constructed around the terms used in the document. Through considering the various terms in the index, the similarity measurement is performed between the query and the document collections. In addition, this measurement may go beyond the statistical term weighting by utilising semantics components such as thesaurus or ontology from a particular domain.

The domain that we are dealing with, email communication, has the characteristics that make it suitable for textual CBR implementation. The document collection used as the basis of the case base has a text-based form. Even though an email message has a clear structure consisting of sender, recipient, date, subject and body, the greatest source of information is located in the subject and body. Moreover, these fields may form a semi-structured or even unstructured text, making it a challenge to extract useful knowledge from it. Therefore, a textual CBR system mostly relies on a technique for identifying the natural language text that is available in an email message.

2.5 Related tasks and techniques in textual CBR

As explained in the previous section, Aamodt & Plaza (1994) defined the retrieval process as commencing the CBR cycle. It requires past problems similar to the query to be retrieved. The problems, as presented in the email messages, contain information in a form of natural language communication. Therefore, in this section, we attempt to discuss the process of retrieving information from email messages. Since these messages typically consist of textual content and file attachments, we scoped the study to process only the text-based information. We explored past studies in the area of text processing before finally implementing

these techniques in our proposed solution.

2.5.1 Information Retrieval

An information retrieval (IR) process emerges from the notion that the user needs access to information. This information is typically organised and stored in a particular place. Users should be able to access the information they have an interest in easily. However, understanding the information that the user needs is not always a straightforward problem. It has to be translated into a query to be processed by an IR system (Baeza-Yates & Ribeiro-Neto, 1999). It is also important to note that a query presented to the IR system should return information items (i.e in a form of documents) that are relevant to the query. Therefore, relevance in the retrieval process is the core of the IR field.

In order to provide efficient access to a large volume of documents, an index is typically built. While various index structures have been studied, one of the most popular is the inverted index (Baeza-Yates & Ribeiro-Neto, 1999). This index type has two components: the vocabulary and the occurrences. The vocabulary is the distinct words (or terms) that appear in the document. For each word, the position of its appearance in the text is stored in a list. This list is further referred as occurrences. This allows direct access to particular matching text positions. A fundamental study conducted by Voorhees (1986) showed that an inverted index search guaranteed the retrieval of all documents based on the greatest similarity, instead of zero. This was achieved by a search directed to the terms instead of scanning the document thoroughly. The results showed that the inverted index search is more efficient than a cluster search.

Furthermore, it is likely that not all the index terms are equally useful for representing the document contents. Some terms might be more vague than

others. With respect to the document storage (or corpus), terms that appear in every document in the corpus may be less useful as they do not represent a set of specific documents. In contrast, this generalisation could be avoided when, for instance, five terms are found, meaning the search could be narrowed down. These terms should be assigned a weight according to the relevancy of each in representing a document. Therefore, Baeza-Yates & Ribeiro-Neto (1999) define this phenomenon in three classic information retrieval models, namely the Boolean model, the Vector model, and the Probabilistic model.

The earliest model, the Boolean, is based on an understanding of theory and algebra. This model measures terms as 0 or 1, or as having a Boolean value of `FALSE` or `TRUE`. Thus, it considers the document to be either relevant or not relevant. While the clear semantics and its simplicity are the main advantages of this model, it lacks the notion of partial matching. This may lead to retrieving too many or too few documents (Baeza-Yates & Ribeiro-Neto, 1999).

The second model addressed drawbacks identified in the Boolean model. The vector (space) model (VSM), as introduced by Salton et al. (1975), attempts to assign non-binary weights to the terms in the documents. One of the simplest and most commonly implemented methods to calculate the weight of the terms is by considering the frequency of each term in a document (TF) as well as in the whole corpus (IDF) (Salton et al., 1975). By obtaining the weight of each term in a document, the degree of similarity between a query document and documents in the corpus can be computed. This also allows the retrieved documents to be ranked according to the degree of similarity. The ranked list also considers documents that only partially match the query documents. Thus, it is likely to have a higher chance of returning a document that matches the user's needs, compared to the Boolean model.

The last model utilises probabilistic inference in the process of document

retrieval (Robertson & Jones, 1976). The degree of similarity between documents is computed with respect to the probability of being relevant. Thus this model also generates a ranked list of relevant documents based on the degree of similarity, similar to the Vector model. While this model is capable of producing a ranked list, it is necessary to provide a set of assumptions to distinguish the documents into relevant and non-relevant sets (Baeza-Yates & Ribeiro-Neto, 1999).

Amongst all the models explained above, the Vector Space Model has been very popular in the study of IR (Baeza-Yates & Ribeiro-Neto, 1999). Therefore we decided to implement this model in our proposed solution, because of the following considerations. First, it seems challenging to provide an initial set of assumptions about which email is relevant to another email. Then, we assume that whenever an email query is made, the system is expected to retrieve the relevant documents, which include partially matching documents. This way the ranked list of possible solutions could be provided to the user. Further explanation about the corresponding technique is presented in the next section.

2.5.2 Natural Language Processing

In order to perform an effective retrieval of information, it is important to understand how it is presented. In an email communication, people send and receive textual information, which has a natural linguistic structure. Rules are applied to structure linguistic expression (Manning & Schütze, 1999). We attempted to identify common patterns that occur in language use. An approach to natural language processing (NLP) arises from an understanding that computers are good at recognising patterns that can be useful to automate basic linguistic tasks in human communication.

Liddy (2001) argues that there are several common approaches in natural language processing: symbolic, statistical and connectionist. The symbolic approach analyses linguistic phenomena based on explicit representation using human-developed rules. An implementation of this is in the rule-based system, where knowledge is represented as facts, a set of rules and semantic networks. Tasks such as text categorisation, lexical acquisition, and information extraction benefit from the presence of rules.

The second approach concerns adapting statistical techniques to establish a generalised model of linguistic phenomena by using large text corpora. While the symbolic approach uses a predefined set of rules, the statistical approach benefits from observable data as its source of knowledge. This approach is widely used in speech recognition and parsing since it utilises large corpora to model generalisation.

The connectionist approach is similar to the statistical approach in the way that both develop generalisations from linguistic phenomena. However, this approach represents the model in the form of interconnected network models where knowledge is stored in the processing unit and forms a particular relationship. Weight is applied to the connections between every processing unit. The interconnected network model has influenced solving semantic tasks such as word-sense disambiguation.

Furthermore, Liddy (2001) suggests applications that utilise text-based information to use NLP. Some examples include information extraction (IE), information retrieval (IR), and a question-answering system. The IE area focuses on extracting, recognising and tagging a particular element of information, such as a name, location, or organisation from a collection of texts. This extraction may significantly enhance the implementation processes of the system, such as data mining. Moreover, Liddy (2001) argues that only a few implementations of

IR use NLP, given that this particular area deals with text. The question-answer system benefits from an NLP approach, since this area attempts to provide automation to answer a user's query.

2.5.3 Document Preprocessing

As discussed in the previous section, it appears that not all of the index terms are equally meaningful in representing the documents. Thus, the preprocessing phase of the documents is typically considered to determine these index terms. In this section, we explain several document preprocessing techniques, as categorised by Baeza-Yates & Ribeiro-Neto (1999), which are further referred to as text operations or text analysis techniques. These operations include normalisation of the text by lexical analysis, removing common English words identified as stopwords, reducing syntactic variations of the terms using stemmer, and expanding the context of the term by finding its corresponding synonyms from a lexical database (i.e. thesaurus).

Lexical analysis

In a text document, information is formed by a group of words. To find meaningful information also means understanding the context. This objective stands behind performing lexical analysis: to provide normalisation of the text, including but not limited to the treatment of punctuation marks, digits and hyphens, as well as upper and lower cases. This process is continued by converting a stream of text in a document, commonly separated by a whitespace, into a stream of candidate words to be indexed. While lexical analysis could reduce noise in identifying the context of a document, careful consideration may be applied in several cases. Numbers, independently, can be vague when used as index

terms. However if surrounded by a context, for instance '1240A.D.', they can be meaningful. The same applies to punctuation marks, which can be used to abbreviate, as in 'a.m.' in 'ante meridian' or as 'am' in 'I am'.

Stopwords removal

Once the stream of potential words for index terms has been generated through lexical analysis, some may be shown to have less influence on the context of the document. This happens because they appear too frequently in a document, and thus can be useless in the retrieval task. These words are termed stop words (Baeza-Yates & Ribeiro-Neto, 1999), and include words commonly labelled as prepositions, articles and conjunctions. Early work done to identify stop words in the English language was made through analysing a large collection of corpus (Fox, 1989), as shown in Table 2.6 .

Table 2.6: An example of a stopwords list obtained from Fox (1989)

Top 20 stopwords in Brown Corpus	the, of, and, to, a, in, that, is, was, he, for, it, with, as, not, his, on, be, at, by ...

Although eliminating these stop words provides an important benefit, especially on the size of the indexing, Baeza-Yates & Ribeiro-Neto (1999) argue that it may also reduce recall. Phrases containing these stopwords may not be recognised in the retrieval process. However, the way to work around this issue is to implement full-text searching as an additional feature.

Stemming

Stemming is a process of deriving a word by its root. This method is commonly applied to reduce the variability of the words; thus more matching can be found.

For instance, the string such as "buy", "buys", and "buying" can be reduced to its root, "buy". This can reduce the size of index terms as well, since only a lower number of distinct words (word root) are stored.

One of the most recognised works in word stemming is the Porter algorithm (Porter, 1980). This algorithm stands behind the notion that most variants of a word are generated by an addition of suffixes instead of prefixes. Therefore Porter's algorithm uses a set of rules for modifying the suffixes of the words in a text document, for instance in examining plural words.

Term frequency and weighting

The text analysis techniques mentioned earlier allow for normalisation of the terms. With respect to the correlation between terms in the query document and documents in the corpus, documents that contain more matching terms tend to be similar and should receive a higher score (Manning et al., 2008). Thus, terms generated in the preprocessing phase can be considered when measuring similarity scores between documents.

Term weighting can be computed in various ways, and one of the most widely adapted in the IR field is the *TF-IDF* scheme. This scheme was introduced by Salton & Buckley (1988), and the acronym stands for *term frequency - inverse document frequency*. Statistically, it measures the importance of a word to a document within a corpus or collection. *TF-IDF* weighting considers the scaling mechanism: frequent terms are scaled down and rare terms are scaled up. For instance, a term that appears 10 times more than another may not necessarily be more important.

Given the term t and document d, the *TF-IDF* weighting of a term in the document $tfidf_{(t,d)}$ is computed using the following formula:

$$tfidf_{(t,d)} = f_{(t,d)} * log\frac{|D|}{f_{(t,D)}} \tag{2.1}$$

where $f_{(t,d)}$ represents the term occurrences count in a document, $|D|$ is the total of documents in the case base, and $f_{(t,D)}$ equals to the documents count containing term t.

Some studies report that the use of *TF-IDF* has assisted in performing further analysis (Weng & Liu, 2004; Dredze et al., 2008), and a textual CBR application has also been built by considering *TF-IDF* weighting in the retrieval process (Lamontagne & Lapalme, 2003).

Similarity measurement in Vector Space Model

Salton et al. (1975) introduced a model of IR for when term weights have been computed, which is capable of measuring similarities between documents in a vector space. The Vector Space Model (VSM) transforms a collection of words in a document into a high dimensional vector. The weight of the vector is represented by a set of document terms that have been weighted (in this case using *TF-IDF*). The distance between two vectors is measured using the cosine angle between them, which is usually referred to as Euclidean distance. Thus, it seems clear that the shorter the distance between two vectors, the more similar the documents are.

As shown in Figure 2.2, there are two documents in the case base, represented as vector $\vec{d}_1$ and vector $\vec{d}_2$. Vector $\vec{q}$ represents a user query document. Although these vectors have the same term count (x, y), their term weight (x_1, x_2) and (y_1, y_2) differs from each other. Thus, the distance between the two vectors $\vec{q}$

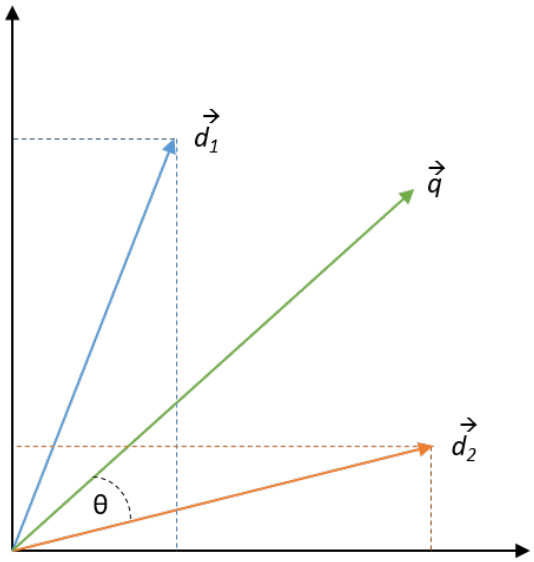


Figure 2.2: Vector Space Model as introduced by Salton et al. (1975)

and $\vec{d}_2$ is measured by the cosine angle (θ) score.

Given the above information, similarity $sim(\vec{q}, \vec{d}_2)$ in VSM is computed as a dot product of both vectors, as shown in the following formula:

$$\begin{aligned} sim(\vec{q}, \vec{d}_2) &= cos\ \theta \\ &= \frac{\vec{q} \bullet \vec{d}_2}{|\vec{q}||\vec{d}_2|} \\ &= \frac{(x_1 * x_2) + (y_1 * y_2)}{\sqrt{{x_1}^2 + {y_1}^2}\sqrt{{x_2}^2 + {y_2}^2}} \end{aligned} \tag{2.2}$$

Although the implementation of *TF-IDF* and VSM has been widely known for their simplicity and ease of use (Manning et al., 2008), they lack an identifying semantic context between the terms. Therefore, by implementing additional linguistic knowledge such as through the use of thesaurus, this issue can be diminished (Abdalgader & Skabar, 2010). In addition, VSM is capable of performing partial matching as well as exact matching between a query and documents in the corpus.

Synonym expansion using thesaurus

During the process of the semantic identification of words in a document, a common strategy is to find a set of words that closely relate to a given word. In linguistics, this set of related words is commonly identified as a synonym. A tool called thesaurus has been widely used to find synonyms. This capability is present since a thesaurus consists of a precompiled list of important words for a particular domain of knowledge and its set of related words (Baeza-Yates & Ribeiro-Neto, 1999).

The use of the thesaurus was motivated by advantages such as a reduction in noise, and retrieval based on concepts (semantic matching) rather than on words (exact matching). It is particularly useful when the domain has an extensive and specific use, such as in the medical field. In the general domain, a popular study on the thesaurus in the English language, known as WordNet, is available (Miller, 1995).

In a study conducted by Hliaoutakis et al. (2006), the authors focused on investigating semantic similarities in a text. They examined these by computing semantic similarities using WordNet (Miller, 1995) as the natural language ontology reference, and MeSH (Lipscomb, 2000) for medical and biomedical terms. The initial experiment was carried out through evaluating the model with the results generated using WordNet and MeSH references. The retrieval model, as identified by the Semantic Similarity Retrieval Model (SSRM), was proposed by taking into account some useful behaviours from the initial experiment. This model performs query re-weighting, synonym expansion and calculating document similarity. By proposing this model, the authors argue that the Vector Space Model by itself is not capable of capturing terms that are semantically similar (e.g. "ask" with "inquiry" or "demand").

Similarly, Abdalgader & Skabar (2010) also studied the influence of word sense disambiguation and synonym expansion in measuring short text similarity. They argued that the standard IR measures of word co-occurrence were not sufficient. Thus, lexical resources such as WordNet were employed to obtain the semantic information for each term in the sentence. An evaluation performed on three different datasets shows that synonym expansion leads to improvement in sentence similarity measurements.

When considering the results from Hliaoutakis et al. (2006), as previously explained, and the study conducted by Abdalgader & Skabar (2010), it seems promising that the utilisation of lexical resources such as WordNet could influence the similarity calculation for our proposed solution. Therefore, we considered adapting the synonym expansion approach in our study. Similarly, we implemented the thesaurus library using WordNet (Miller, 1995).

2.6 Summary

In this chapter, we presented the background to the problem domain that we address in our study. We explained previous approaches related to responding to email messages. While we found two studies (Lamontagne & Lapalme, 2003; Hewlett & Freed, 2008) that are similar to our approach, only one study implemented case-based reasoning as the underlying methodology (Lamontagne & Lapalme, 2003). Furthermore, all of the past text-based studies implemented text preprocessing techniques. However, we identified that in most studies, synonym expansion was not implemented. Therefore, in the following chapters, we attempt to implement and evaluate that technique, along with other text analysis techniques.

Chapter 3

Design and Implementation

While previous studies and potential techniques commonly used in email reply management have been explored in Chapter 2, this chapter starts with an overview of our proposed system - the smart email client. Afterwards we comprehensively present the architecture and functionality design of the system. The next section begins with the implementation of the CBR approach in our design, along with the tools utilised in its development. It describes how an email message is modelled as a case, either a new one or as a part of case base. We also present our retrieval strategies to find the most similar cases and reuse those solutions. Finally, the chapter ends with the current limitations in this development phase.

3.1 Overview of Smart Email Client

Here we attempt to solve the problem of task management in email overload issues, as mentioned in previous chapters, by assisting the user in responding to an email. We designed a prototype of an email client with reply recommendation features when the user is about to reply an incoming message. The system follows

the CBR methodology in retrieving similar cases: reusing the past solution to answer the new problem, revising the proposed solution if necessary, and retaining the successful solution seamlessly as a reply message or a solved case in the case base.

As can be seen in Figure 3.1, initially the user starts by browsing through their inbox messages. These messages are stored locally in the database, since the system is capable of retrieving it from the mail server. Two types of message are then identified in the inbox, whether it is a message that has been replied to or not. The user can see the details of each type. However, when they select a not-replied message to reply to, the system provides two options for replying. First, the normal reply option has a typical reply procedure - a blank text box with the original message attached at the end of the box. In the second option, the system provides a ranked list of replied to messages obtained from the user's own sent items. Finally, when the message composing is finished, the user sends the message and the system delivers it to the mail server as well as storing it in the database for future use.

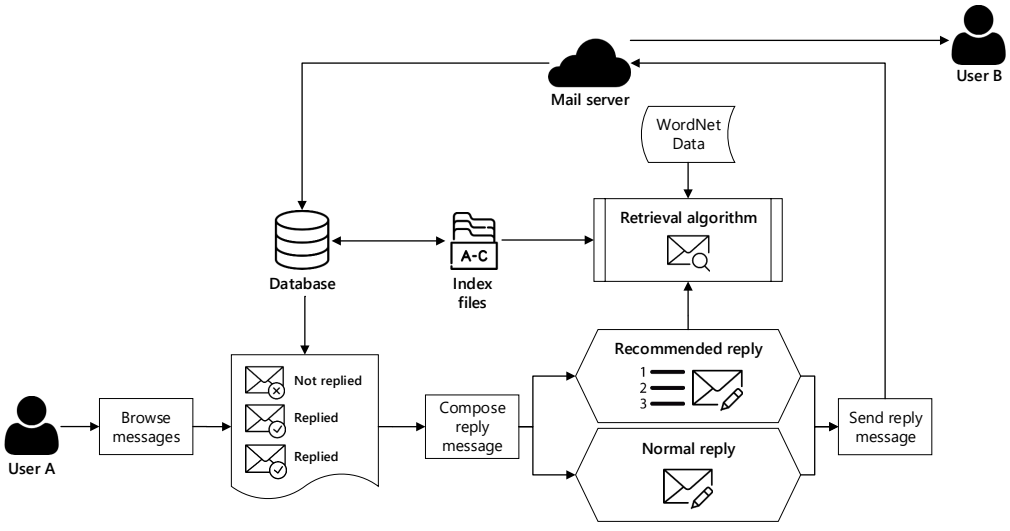


Figure 3.1: Flowchart of the system

Reply actions are provided whenever the user selects a message not replied to (or a query). The normal reply can be utilised if the user chooses to compose a reply message from scratch. However, when the user prefers to see recommended replies from his own past reply messages, a button to action this is also provided. It triggers the retrieval algorithm to search for previously answered emails that are similar to the email query. The system then ranks the top 10 results according to their similarity score. Next, it obtains the corresponding reply message and presents this on a reply recommendation list. This list allows the user to select the most relevant reply message, according to their preference. Finally, the selected reply message is already pre-populated in the reply box, which enables the user to efficiently reuse the solution or to make the necessary revisions before sending.

3.2 System Architecture

An email client application has recently been developed in various environments, including mobile systems and web-based applications, while those with the most comprehensive features are usually implemented in the desktop environment. This is possibly because these features also demand more processing power, and desktop applications benefit from the availability of the processing resources available in, for instance, a personal computer (PC) or laptop. As our prototype implements a new feature to build a recommendation system, we chose to start the development using the desktop environment.

As illustrated in Figure 3.2, we identified that three elements were involved in our application architecture – the mail server, the database, and the user. The main objective for an email client is to be able to communicate with the mail server. This form of communication is shown in activities such as fetching

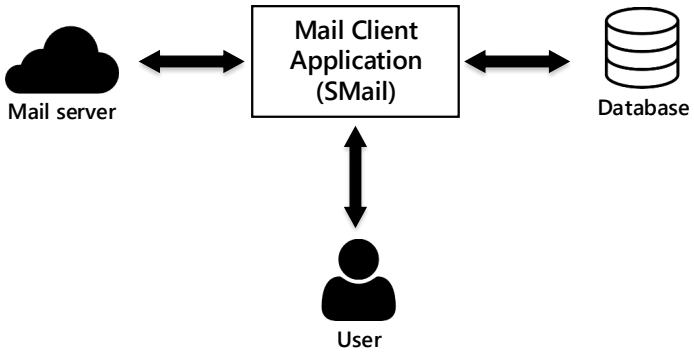


Figure 3.2: High-level architecture view of the email client

new inbox messages and sending a reply message back to the mail server from the application. There might be additional features such as draft management. However, we will address this limitation near the end of this chapter.

It was not feasible to build our case base in the mail server. We also consider that performing too many requests to the mail server to fetch messages is not efficient. Therefore we built a local copy, or case base, using a database management system (DBMS) application. Our main consideration in choosing a DBMS was that it had to be an open source application. Therefore, we chose a system widely used in DBMS by software developers; MySQL[1]. However, there could be the possibility of using a portable database such as SQLite[2] or a file-based DBMS, MongoDB[3]. More details of our database implementation are presented in the next section.

Finally, the usability of the application had a significant influence in our design thinking process. In her study, Mayhew (1999) argues that it is a matter of trade-off and compromise in the user interface design. At the same time, users want powerful functionality but a simple interface. In addition, the system has

[1]https://www.mysql.com/
[2]https://sqlite.org/
[3]https://www.mongodb.com/

to establish ease of learning as well as ease of use. As a result, we took into account the previous consideration. We attempted to design the user interface application by adapting it from an off-the-shelf email client. We identified unwritten guidelines about placing elements in an email client application. A comprehensive explanation about our design considerations is also presented in the next section.

3.3 Functionality Design

We have categorised the requirements above into functional layers, as shown in Figure 3.3. These layers relate to email communication, information retrieval (IR), and graphical user interface (GUI). In addition, a corresponding entity for each layer provides complementary information for the functionality implementation. Following is the further reasoning and discussion on each functionality layer, which further referred as the components.

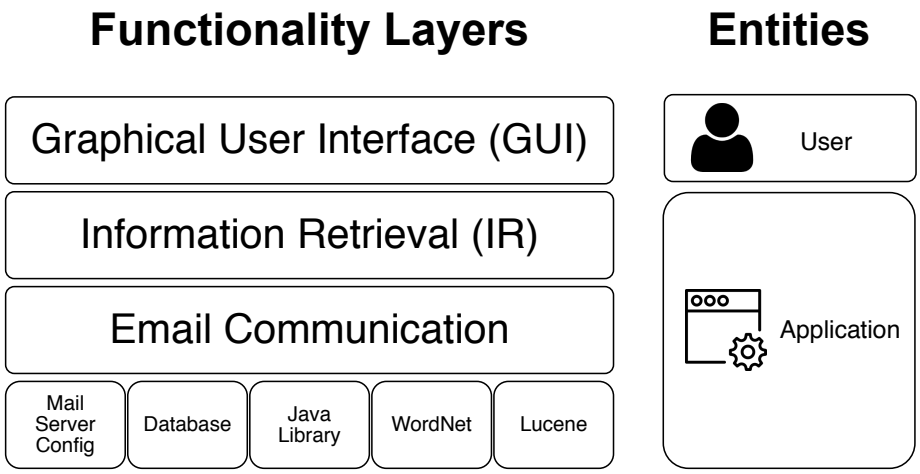


Figure 3.3: Proposed functionality layers in the application

3.3.1 Email Communication

In order to fetching and sending email, we developed an email communication component, which resides under the application entity. This component has the following main tasks: (1) establishing a connection to the mail server using given email account settings; (2) fetching all email messages in the first run and only fetching new messages on the following runs, and (3) using mail server settings to send the email. Our system utilises Internet Message Access Protocol (IMAP), thus it is possible to fetch from the inbox and send messages from its corresponding folder. We also managed to develop a sending capability using the Simple Mail Transfer Protocol (SMTP) settings of the mail server. All these tasks were achieved by implementing JavaMail API[4].

The email message that has been fetched needs to be further processed, since it mostly comes with embedded Hypertext Markup Language (HTML) tags. This process is known as *parsing* the content in the body part of the message. Parsing is performed to ensure the content of the message does not contain significant noise. While we carefully dealt with the quality of the content, we left the parsing for multiple email recipients to future development. At this stage therefore, our system focuses on one-to-one email communication between a sender and a recipient.

After building the communication capability and parsing the content, the messages are stored in the MySQL database. The table definition in the database is presented in Figure 3.4. The table name is self-explanatory - messages from the inbox and the sent folder are stored in their respective tables. We decided to deal with email attachments in the future, so we only stored email header information and the body content of the email.

[4]https://javamail.java.net/

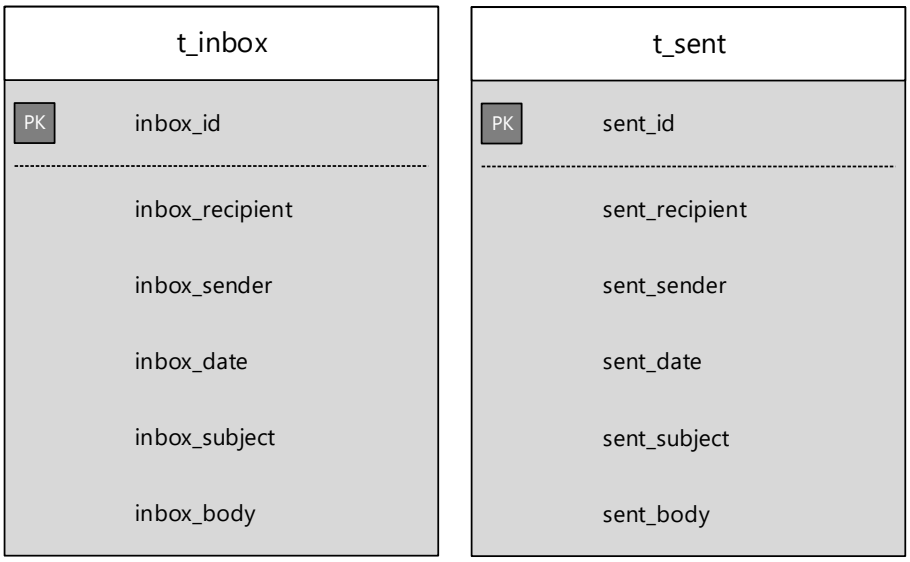


Figure 3.4: Table definition in the email client application

Because of the simplicity of our table definition, we established the relationship between an email message and its corresponding reply using the Structured Query Language (SQL) syntax, as shown below.

Listing 3.1: SQL syntax to establish relationship between inbox and sent email

```
SELECT * FROM t_inbox a LEFT JOIN t_sent b
ON a.inbox_sender = b.sent_recipient
AND a.inbox_subject = SUBSTRING(b.sent_subject, 5)
ORDER BY inbox_id DESC
```

Our implementation was successful in that it recognised messages in the inbox table and matched them to a corresponding reply in the sent table. We implemented `SUBSTRING()` function in order to start the subject reading after the character "Re:" that is usually applied in an email reply. It enables exact matching between the subject in the inbox table and the sent table. However, we have identified the problems in this approach in terms of the variability of

the subject messages. For instance, a user might accidentally modify the subject content, which with our approach, means that we are unable to identify the corresponding reply message, if is any. However, it is not a trivial thing to fix this issue and we leave this for our future development.

3.3.2 Information Retrieval (IR)

The next component that we developed is related to IR capability. Like the previous component, it resides under the application entity. This is the core component in this research as it enables the ad-hoc retrieval of documents (in this case the email message), according to the user query. Moreover, it performs the following tasks: (1) the indexing of message entries in the database; (2) the querying of the index to find the most similar document, and (3) ranking the most relevant documents that are similar to the user query.

From the several IR libraries that are available off-the-shelf, for example the Lemur project[5], we chose Apache Lucene[6] for the following reason. First, Lucene is an open source library and available for Java-based application development, which aligns with our approach. Being open source, Lucene is frequently updated and maintained by the community. Second, it is capable of performing text analysis, such as tokenisation, word stemming, and stop words removal (Białecki et al., 2012). In addition, with further alteration, it is able to parse the WordNet database to apply synonym expansion during text analysis. Lastly, by default, it builds its term indexes using *TF-IDF* weighting (Białecki et al., 2012), which simplifies our work on using term weighting for calculating the similarity between query document and documents in the corpus.

[5]https://www.lemurproject.org/
[6]http://lucene.apache.org/core/

3.3.3 Graphical User Interface (GUI)

Since one of our goals is to make the system more available to the user, we also emphasised the development of the GUI, which resides under the user entity in Figure 3.3. We adapted the interface layout guidelines from several widely known email client applications, such as Mozilla Thunderbird[7] and Opera Mail[8]. The most noticeable layout observed in those applications regards the placement of the message list on the left-hand side and the message details on the right-hand side. Interestingly, this observation aligns with Nielsen & Pernice (2010) in their study of identifying hot spots in web content. These hot spots, also called eye-tracking spots, shape an *F* pattern from the user's point of view. It is more likely to happen when the content draws their attention. It is also more likely to happen when the content draws their attention (Nielsen & Pernice, 2010). By placing the message list on the left-hand side, assisted by colour differentiation (white is for the replied messages, red is vice versa), as shown in 3.5, we expect to provide more assistance to the user.

Building a desktop application means facing the challenge of the limited and outdated library for user interface elements. However, we overcame this problem by implementing the latest Java interface library, the JavaFX[9]. While some outdated interface elements have been replaced, this library also expands the capability of applying the Cascading Style Sheet (CSS) styling scheme and multimedia embedding (Heckler et al., 2014). This improvement extends the flexibility of applying CSS features in our application, such as colour schemes, icon placement, and font adjustment, as shown in Figure 3.6.

[7] https://www.mozilla.org/en-US/thunderbird/
[8] http://www.opera.com/computer/mail
[9] http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

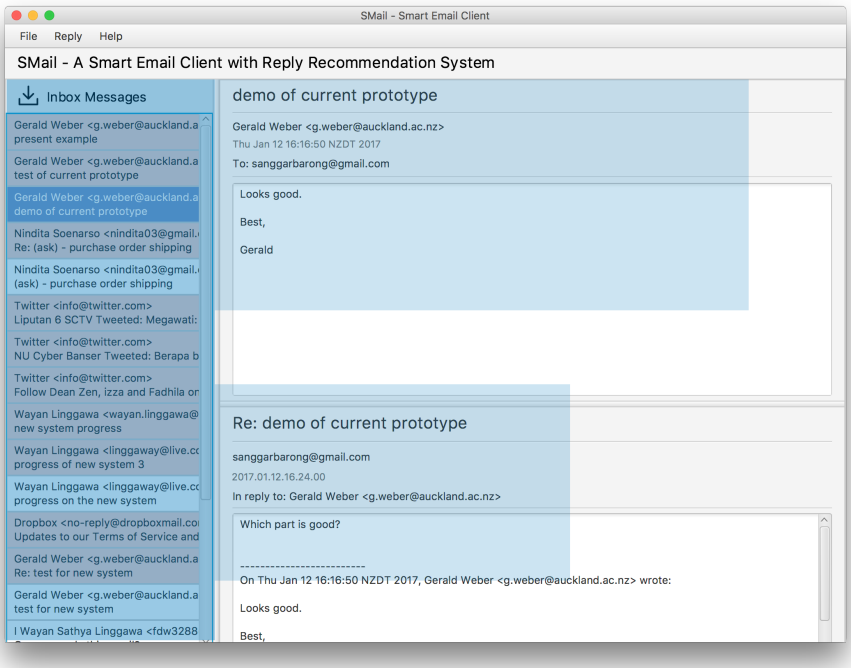


Figure 3.5: The user interface layout forms an F-pattern

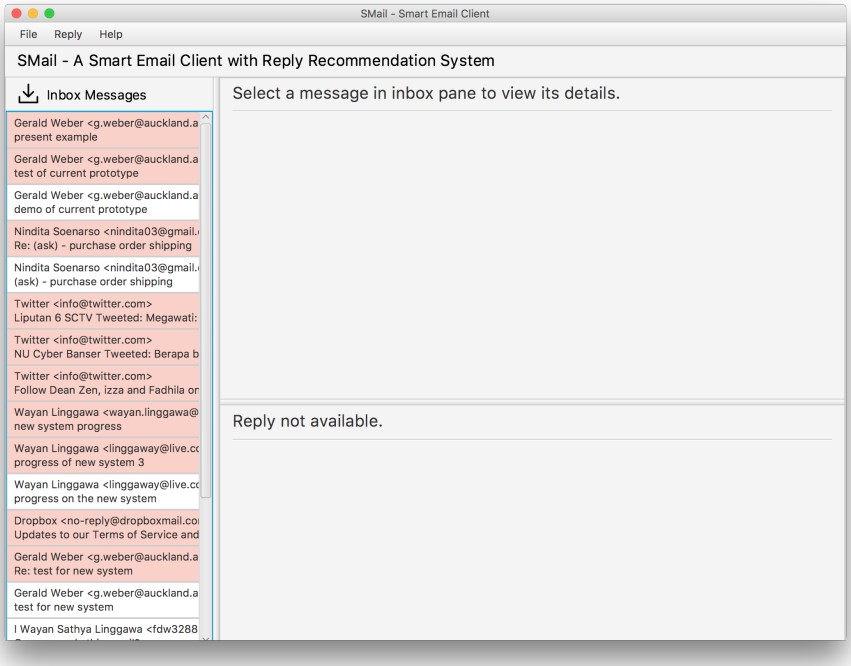


Figure 3.6: Look and feel of the application using JavaFX library

3.4 Adapting the CBR in Smart Email Client

This section provides a detailed explanation of how we adapted the CBR approach in the system beginning with illustrating the overall process in a flowchart. Each phase in the CBR cycle is then mapped. This includes how we represent the email reply problem as a case, define the case base, retrieve similar past cases, reuse the solution, then revise it and retain the solved problem in the case base. We further explain the tools that we used during development.

3.4.1 Process Flowchart

The process in our system adapts CBR methodology, as represented in Figure 3.7. It begins when the user starts the system and email messages are fetched from the mail server to the database (case base). This is the user intervention. Further, these messages can be considered as new cases or past cases.

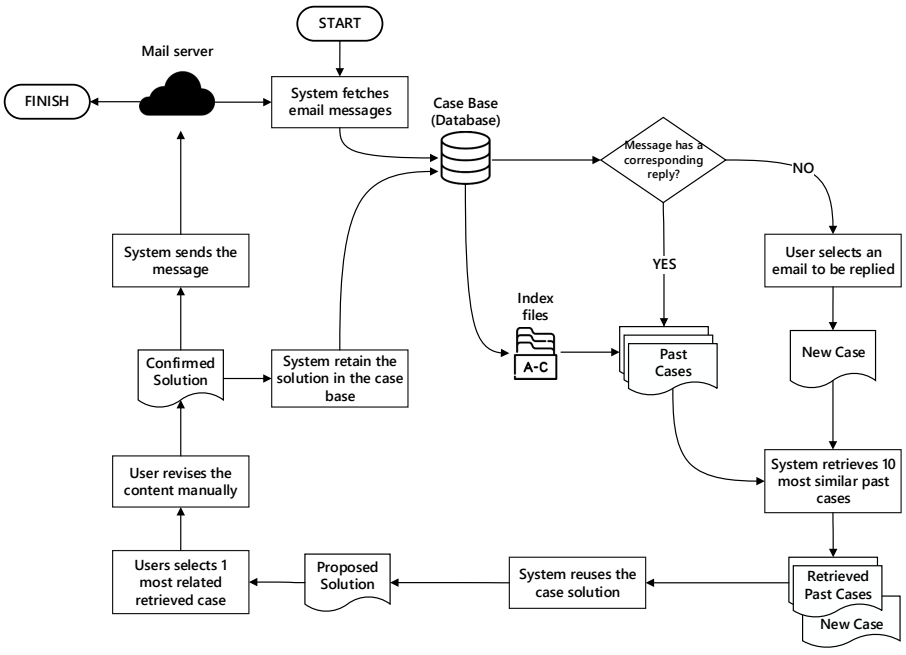


Figure 3.7: Process workflow adapting CBR methodology

When the user selects an email to be replied to, the process of case retrieval starts. The system attempts to retrieve the 10 past cases that are most similar to the selected email. Furthermore, it also reuses the content of the email and presents them as the proposed solution. The users then select one the most preferred proposed solutions and revises the content, manually if necessary. Whenever the user finishes composing the content and clicks send, the system sends the message to the mail server and saves the copy to the database.

3.4.2 Case Representation

In the table definition, we showed a rich content of messages for both inboxes and sent email. However, to adapt the CBR approach, a case representation must be created. A case consists of a problem description with or without a solution. It should have a structured pair of feature and value. With respect to the email features obtained, we propose the case representation as shown in Table 3.1.

Table 3.1: Case representation from the email corpus in database

Feature	**Value**
Problem Title	"Subject" field of an inbox message
Problem Description	"Body" field of an inbox message
Solution Title	"Subject" field of a sent message
Solution Description	"Body" field of a sent message

From the proposed case representation it can be seen that we have not used other header fields such as the "To", "From", or "Date" fields, in contrast to the study undertaken by Lamontagne & Lapalme (2003), where they used the recipient and sender information. The main difference is that our approach focuses on analysing the content of the email message and applying text processing and semantic analysis technique. Lamontagne & Lapalme (2003) benefitted from using the sender or recipient fields since they also performed an information

extraction technique to obtain the named entity.

Moreover, we categorise the acquisition process of a case under two conditions. First, if the email has no corresponding reply, that means the case does not have a solution. We consider these as *unsolved cases*. Then, any messages with a reply associated with that message are considered to be *solved cases*. Cases remained unsolved until the user attempts to select one as a new case by starting the reply action. Each reply message that is successfully sent we retain as a solution to be used for future similar new cases.

3.4.3 Case Base Population

Previously proposed case representation presents an efficient structure for the system to store the cases. This is done by excluding supplementary information from the email corpus in the database, material such as the sender and recipients information. One way to store the cases in the case base is by indexing the case feature. In this case, the feature is textual information. Thus we adapted the *TF-IDF* calculation and vector space model presented in Chapter 2.

The process of indexing a case base is shown in Figure 3.8. In the beginning, the fetched email may contain HTML tags, which might increase the noise in the email message. Thus, we parse the message into plain text before storing it in the database. The system then starts the analysis process by fetching records from the database. Various text analysis techniques are performed to generate ready-to-index terms. In addition, a lexical resource for synonym expansion is employed, namely WordNet. As a result, the system can expand a term with its corresponding synonym terms to be indexed. Finally, these terms are indexed using a native indexing method, the inverted index, from Lucene.

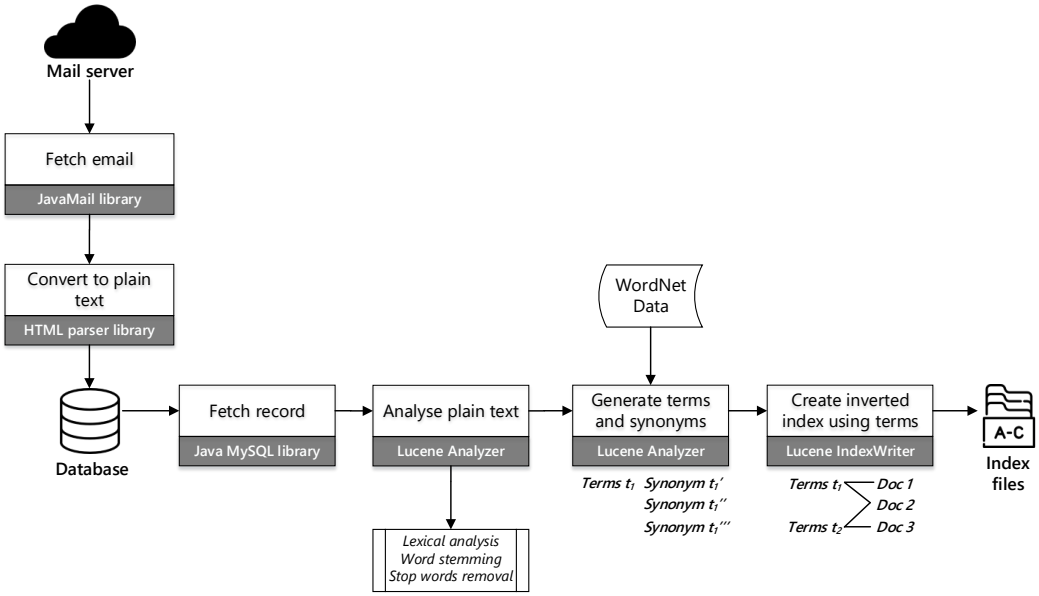


Figure 3.8: Flowchart of indexing case base

The indexing process involves converting the text into a bag of words (or bag of terms). In Lucene, this process is utilised using *Analyzer* class, where it represents a text analysis functionality performed in the conversion process. Lucene further allows the creation of *CustomAnalyzer*, where we can define our own text analysis technique instead of using the default one. For the purposes of evaluation, several types of *Analyzer* were built.

Algorithm 1 shows that the process of building *Analyzer* was started by initialising `tokenStream` to collect processed words at the end (line 1). `Tokenizer` allows tokenising words in a document by considering whitespace between each word (line 2). It also omits punctuation and other non-numeric signs, which is usually called the *normalisation* of a text document. Furthermore, Lucene also provides a library for further normalisation by transforming all words into lowercase using `lowercaseFilter` (line 9 and 15). `snowballFilter` then implements the Porter stemming technique (Porter, 1980), to stem the suffix of the words

(line 10 and 16). In addition, Lucene has a collection of English stopwords (line 5) by default which can be read by `stopwordsFilter` (line 11 and 18).

Moreover, with some customisation efforts, expanding each word with its synonym can be done when the WordNet database (line 6) is parsed using the `wordnetSynonymParser` (line 7). Finally, the `synonymFilter` matches the tokenised words with entries in the WordNet database before the parsed synonym can be retrieved and injected into that word (line 19). This allows every word that has a synonym in the WordNet database to be paired with its corresponding synonyms. Finally, the stream of terms (or tokens) can be further processed for indexing (line 21).

Algorithm 1 Building $Analyzer$ for text analysis and synonym matching

Input: email message Doc
Output: stream of terms T
1: **initialise** $tokenStream(T)$
2: **initialise** $tokenizer(Doc)$
3: **initialise** $tokenFilter$
4: **initialise** $analyzerType$
5: **initialise** $stopwordList$
6: **initialise** $wordnetDir = \%file\_directory\%$
7: **initialise** $wordnetSynonymParser(wordnetDir)$
8: **if** $analyzerType = Standard$ **then** ▷ Basic text analysis process
9: $tokenFilter \leftarrow lowercaseFilter$
10: $tokenFilter \leftarrow snowballFilter$
11: $tokenFilter \leftarrow stopwordsFilter(stopwordList)$
12: $tokenStream(tokenizer, tokenFilter)$
13: **end if**
14: **if** $analyzerType = StandardWithSynonym$ **then** ▷ Use WordNet DB
15: $tokenFilter \leftarrow lowercaseFilter$
16: $tokenFilter \leftarrow snowballFilter$
17: $tokenFilter \leftarrow synonymFilter(wordnetSynonymParser)$
18: $tokenFilter \leftarrow stopwordsFilter(stopwordList)$
19: $tokenStream(tokenizer, tokenFilter)$
20: **end if**
21: **return** $TokenStream(T)$ ▷ Stream of processed words

After *Analyzer* has been built, as can be seen in Algorithm 1, it can be implemented in the indexing process. Algorithm 2 shows that the *Analyzer* is used (line 1) in the Lucene's native index writing functionality (lines 2 and 8-12) to write the index files to the specified directory (line 3). During this process, the case feature from our case base (lines 5 and 6) that is going to be indexed should be defined as to whether it is the subject (problem title), the body (problem description), or has both features (line 4).

Algorithm 2 Indexing email message (case)

Input: stream of terms T
Output: index files Ix
1: **initialise** $Analyzer(T)$
2: **initialise** $indexWriter(Analyzer)$
3: **initialise** $indexDir = \%file\_directory\%$
4: **initialise** $caseFeatureName$
5: **connect** to DB
6: **query** DB to "SELECT caseFeatureName FROM table_inbox" as $resultSet$
7: **while** $resultSet \neq NULL$ **do**
8: $luceneField \leftarrow resultSet$
9: $luceneDoc \leftarrow luceneField$
10: $luceneDoc.property(getTermVector)$
11: $Ix \leftarrow indexWriter(Analyzer) \leftarrow luceneDoc$
12: $indexDir \leftarrow Ix$
13: **end while**

Furthermore, Lucene allows for the state-of-the-art *TF-IDF* calculation of the words (or terms), until the term vector is obtained. It generates its indexes in the form of special files stored on the hard drive. Terms in a document contribute to the magnitude of the vector, making it possible to represent a document vector in a vector space (line 10). Therefore, finding similar cases is easier since the case can be mapped as a vector in the multi-dimensional space. Further explanation is provided in the next section of case retrieval.

3.4.4 Case Retrieval

The retrieval of similar cases starts whenever a user triggers the reply action, particularly when they choose to reply using the recommended reply option. We employed a retrieval algorithm that measures the angle of two document vectors in the multi-dimensional space. As can be seen in Algorithm 3, it takes on the input of a new case, which is a selected email that is about to be replied to. Lucene's native functionality to read the index file is then defined (lines 1-2). A document object, which contains the vector of terms in that particular document, is collected and normalised (lines 4-8) according to the cosine similarity formula (Salton et al., 1975). The similarity score of the two documents (the query document and the document in the case base) is obtained by calculating the dot product of both vectors (lines 10-14). At the end of the process, the algorithm generates a ranked list of usable past replies (lines 15 and 16).

Algorithm 3 Retrieving similar past email (past cases)

Input: newCase C_n & pastCases C_p in case base Ix
Output: n most similar pastCases C_{pn}

1: **initialise** $indexReader()$
2: **read** Ix using $indexReader()$
3: **initialise** $caseVector[]$ as **count** total C_{pn} in Ix $caseIx$
4: **for each** C_n, C_p in Ix **do**
5: **get** term vector $caseTermVec$ in Ix
6: $caseVector[] \leftarrow$ **count** $caseTermVec$
7: $\mid caseVector[] \mid$ ▷ Normalisation of vector
8: **end for each**
9: $i = 0$
10: **for each** $caseVector[]$ **do**
11: $cosineSimilarity \leftarrow caseVector[C_n] \bullet caseVector[C_p]$ ▷ Dot product
12: $mapRetrievedCases \leftarrow map(i, cosineSimilarity)$
13: $i++$
14: **end for each**
15: **sort** $mapRetrievedCases$
16: **use** top n $mapRetrievedCases(C_{pn})$

When the similarity measurement has been calculated in every case, it is ranked according to the similarity score. The higher the score, the more similar the new case is to the retrieved past cases. The next step, reusing the solution from the past case, is triggered by the user when they select a result from the recommendations list.

3.4.5 Case Reuse

The process of reusing past retrieved cases includes two aspects. First is the differences between the new and past cases. The second is about which part of the retrieved cases can be used in the new case. In our email reply problem, consideration of these differences was partially made during the retrieval process by calculating the similarity of our case feature. Then the system identifies the corresponding email reply and reuses it. Implementation of case reuse in our system is illustrated in Figure 3.9.

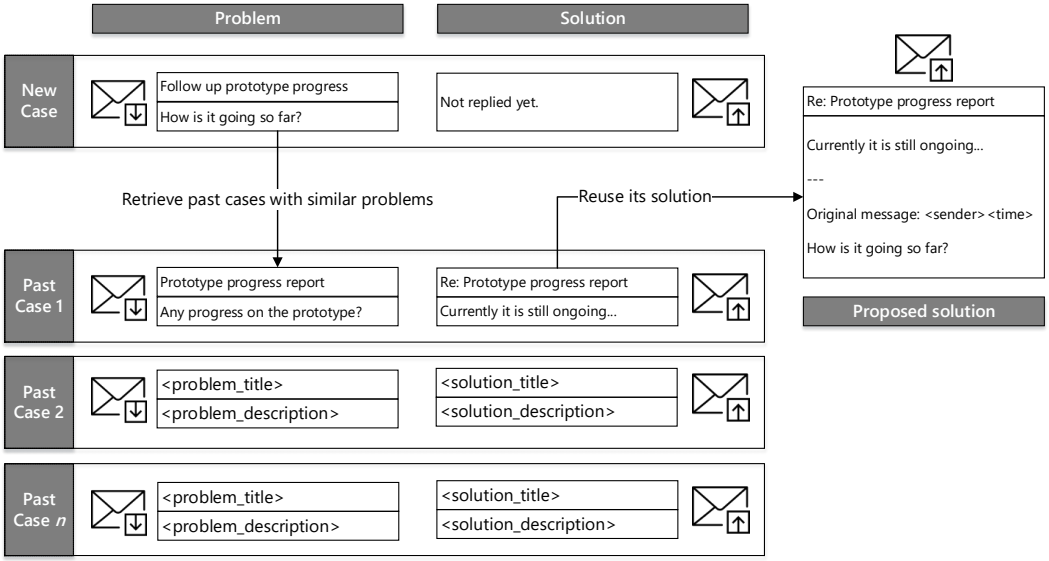


Figure 3.9: Case reuse

3.4.6 Case Revision

During the revision phase, solution evaluation and repair are involved. The solution can be evaluated directly by an evaluator or indirectly measured by certain measurement indicators. The repair process may involve reviewing information in the solution. The addition or removal of content can then be performed. Our system implements case revision, which involves user intervention as shown in Figure 3.10.

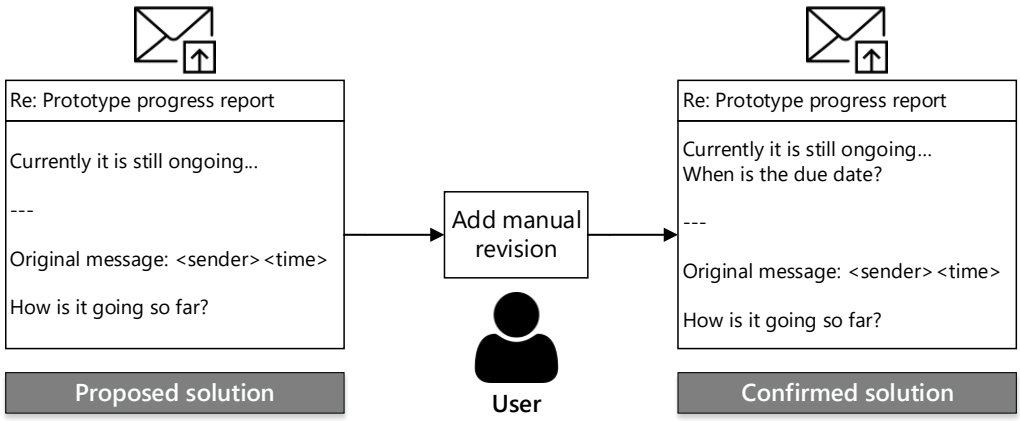


Figure 3.10: Case revision

The reason behind involving user intervention is because the user should have the direct role of being an evaluator. While partly subjective, it is the closest approach to match with his personal preference. If the user is pleased with the result, they are able to use it directly. However, if they are not, our system provides a feature for editing the solution. In this case, the user is able to revise the predefined reply message by doing further editing before sending the message.

3.4.7 Case Retain

The case retaining process is concerned about what information to store and how the solved case can be indexed for the future retrieval process. In our system, when the user sends the message it performs two actions, as can be seen in Figure 3.11. First, the message is delivered to the mail server to be further processed for the recipient. Second, the sent message is automatically paired with its incoming message and stored in the database. Thus, we can consider this to be a solved case. Furthermore, when the future retrieval phase is triggered, the solved case can be indexed and retrieved if it is similar to the new problem.

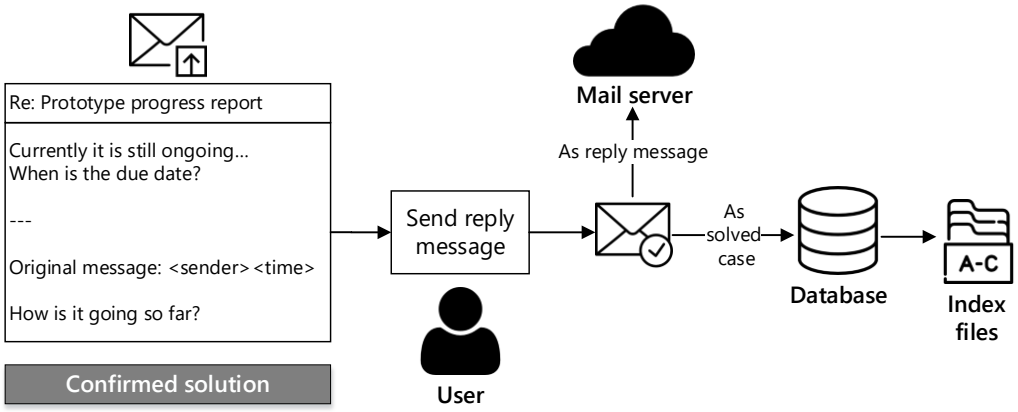


Figure 3.11: Case retain

3.5 Implementation Results

This section provides several screen captures of the smart email client application. We briefly explain what the user can do on the corresponding screen. As previously explained, we developed a Java-based desktop application as a "proof of concept" of the CBR approach.

Start screen

The start screen consists of several interface components. First, there are four headings to highlight the following information: application name, inbox message area, incoming email area, and reply email area. Second, the inbox list is constructed from a scrollable list to assist navigation for a large number of messages.

When the user starts the application, the list of their inbox messages is shown on the left-hand side of the screen, as shown in Figure 3.12. The red colour represents the unreplied messages, while the white is for replied messages. The details of the messages are shown in a split screen, where the top is for the incoming message and the bottom is for its corresponding reply. The user is asked to select the message before the message details appear.

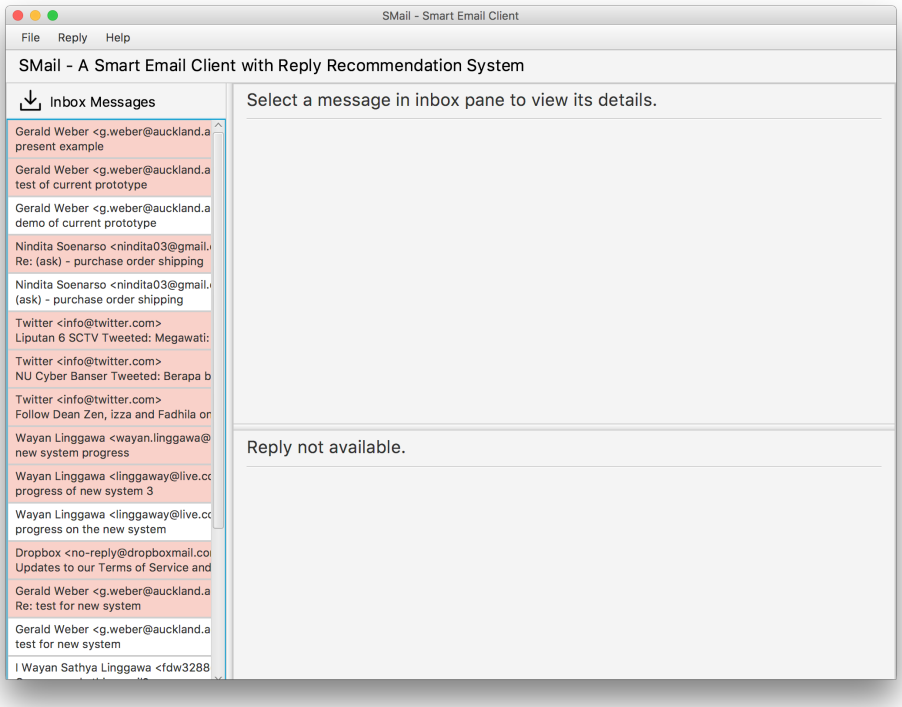


Figure 3.12: The start screen user is asked to select a message

Unreplied message screen

Adapted from the previous interface, this screen shows the detail of an unreplied to message by using several components. Four text labels map the following information: subject, sender, date, and recipient. Then the non-editable text area shows the body of the message. Finally, two buttons provide options for replying.

The unreplied to message is selected by the user, as shown in Figure 3.13, and the details are shown on the right-hand side of the screen. The reply section shows the "not replied yet" notification, since it does not have a reply message associated with it. To start replying to the message, the user can choose between using a normal reply or a recommended reply.

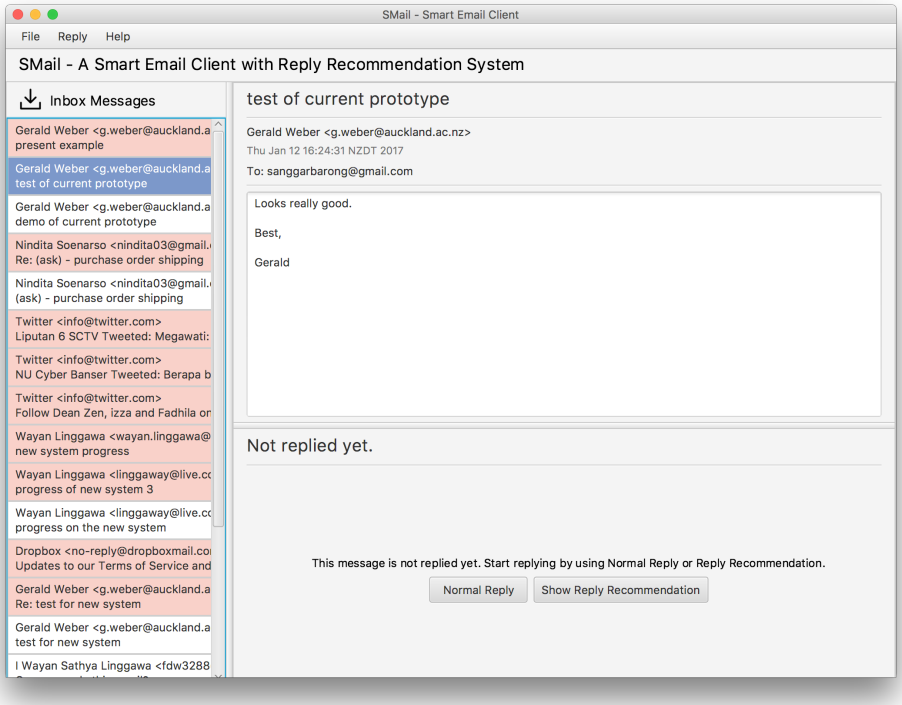


Figure 3.13: The unreplied to message screen and two options for replying.

Replying to an email using the normal reply screen

In this screen, a pop-up window was built as the underlying container for other components. It has a heading, which acts as an instruction. Text labels map information about the email that is to be replied to. The editable text area is provided for typing the body message. Finally, there are two buttons for sending the email and for cancelling the reply action.

Figure 3.14 shows the screen using normal reply. It appears when the user clicks on the normal reply button. In the reply box, the user can type the reply message manually. The corresponding original message is placed at the bottom of the message, separated by a divider to distinguish between the reply message and the original message.

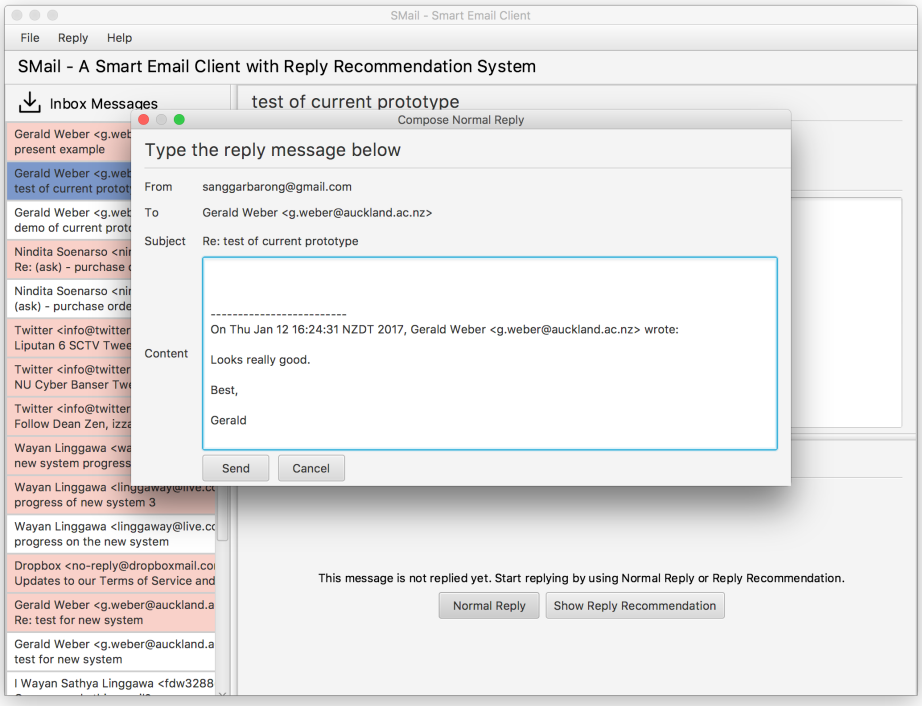


Figure 3.14: Reply window using normal reply

Replying an email using recommended reply screen

In addition to normal reply, the system also provides an assisted reply via the recommended reply option. When the user clicks on that option, the system starts the retrieval of similar past cases and reuses its past replies so they are presented in a ranked list, as shown in Figure 3.15. The relevance column shows the similarity of the corresponding past reply; the higher the score, the more relevant it is.

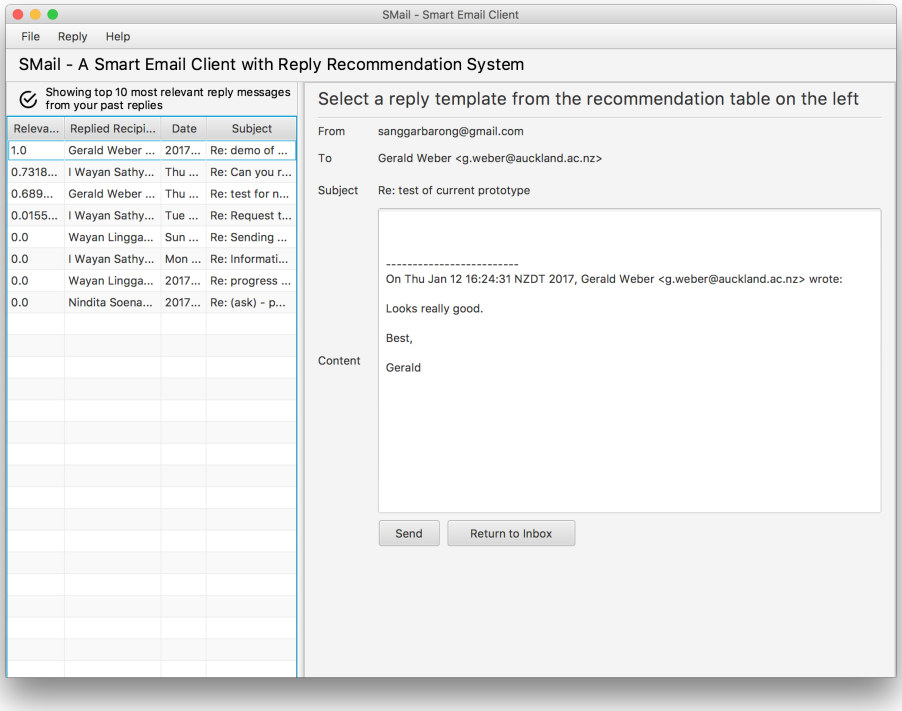


Figure 3.15: Implementation of case retrieval in the application

This screen preserves the layout of the previous screens: the area for navigation on the left-hand side and for reply information on the right-hand side. Several headings are used for instruction purposes. The table is built to display recommendation results. The replying text area is constructed with a predefined message.

Whenever the user selects a result from the recommendation table, the content is prepopulated automatically in the reply box. This allows the user to see the content of their past replies, as illustrated in Figure 3.16. If the user is satisfied with the content, they can send the message directly.

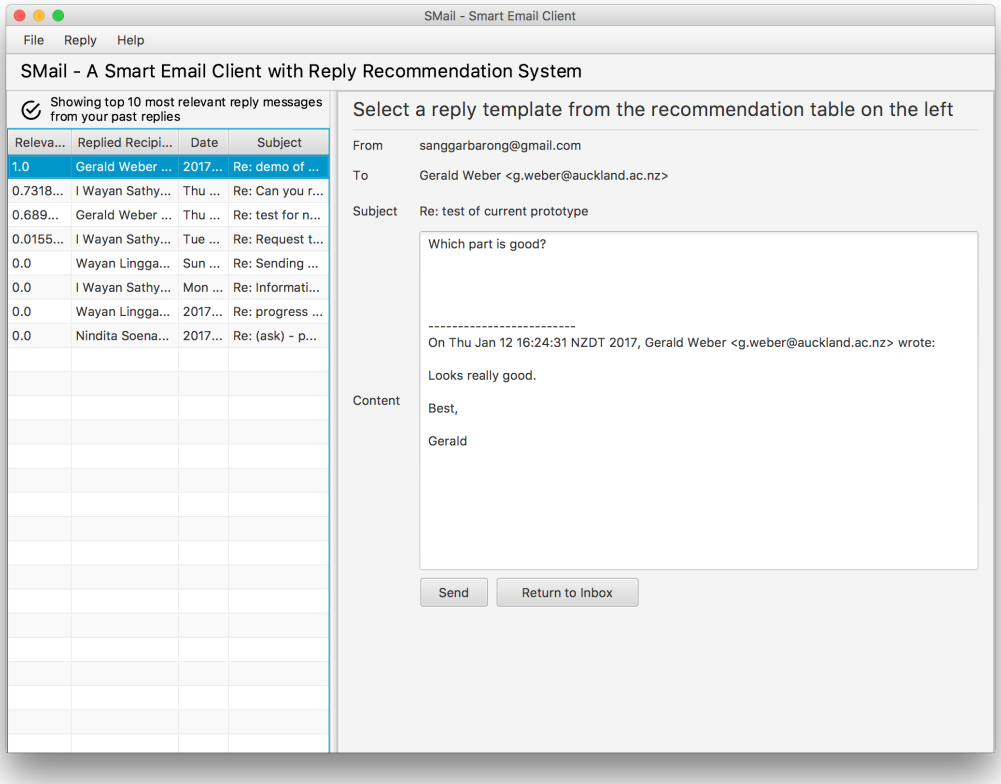


Figure 3.16: Implementation of case reuse in the application

The table constructed for the recommendation list has the capability to populate its content in the given text area. As mentioned previously, it is triggered by user selection. Thus, every change in selection automatically changes the populated content, while the details from the origin email that is about to be replied to is still preserved.

However, it is also possible to add more or to revise the content according to the user's preference. The editing process is shown in Figure 3.17. The original content is placed similarly to that in a normal reply at the bottom of the message. The separator for original and reply content is also provided.

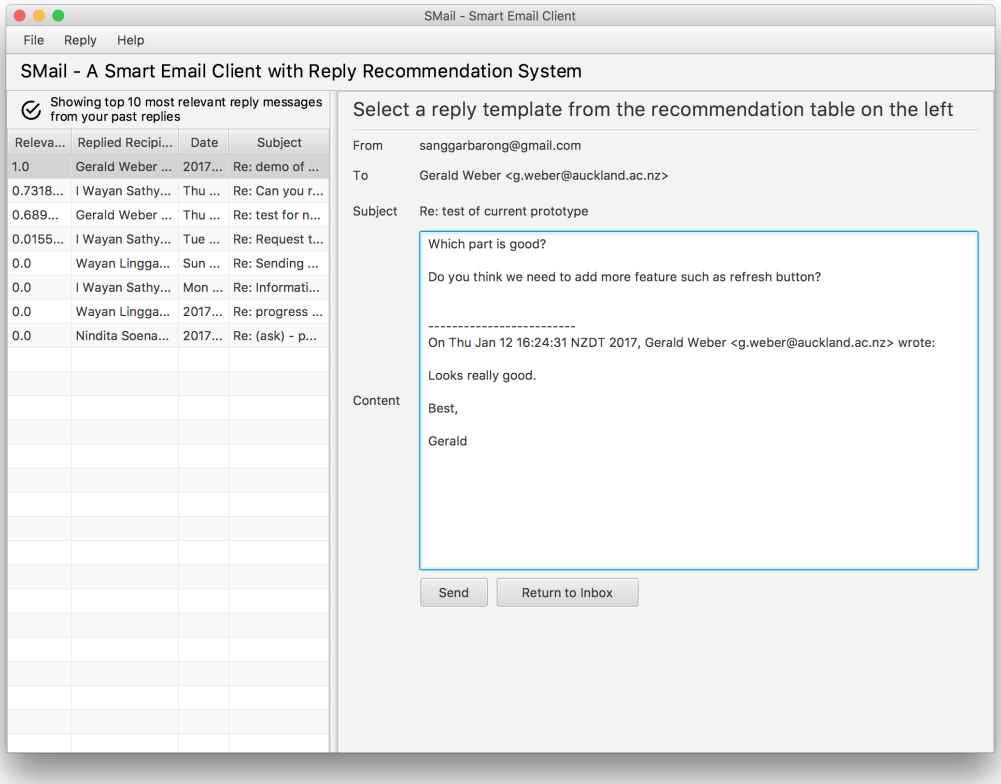


Figure 3.17: Implementation of case revision in the application

The text area provided enables explicit editing of the body part in the reply message. Considering the limited capability of the text area, the content typed is treated as plain text. Finally, two buttons are provided to send the email or to return to the start screen respectively.

Replied message screen

At the end of the process, after the user has clicked the send button, the system delivers the message to the mail server. In addition, the system also retains the reply message as a solution, making the case solved. Since it resides inside the database, this pair of incoming and sent replies is able to be retrieved for a future similar problem.

Further, the colour of the message changes from red to white once it has been replied to. The detail of the replied message can be retrieved from the inbox list, as shown in Figure 3.18. The user can be brought back to the start screen after the system finishes processing the reply message to the mail server and the database.

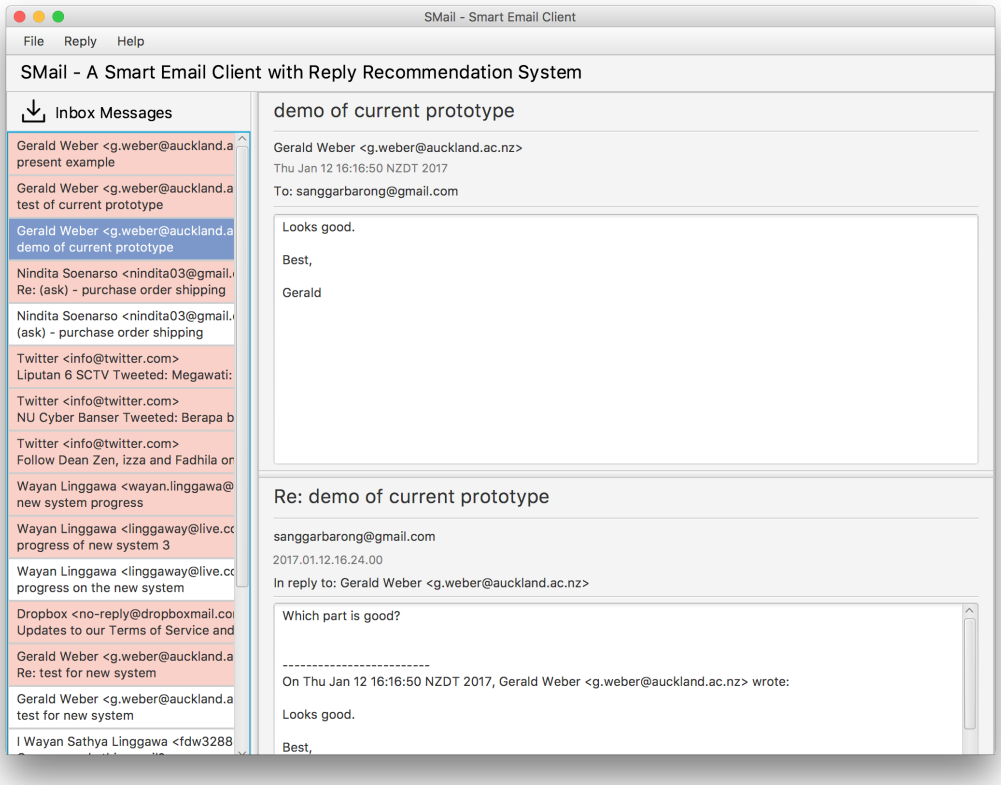


Figure 3.18: Implementation of case retain in the application

3.6 Restrictions and Limitations

3.6.1 Logic

The prototype has only been developed as a "proof of concept". Therefore, we developed only the most vital functionality of an email client. These relate to sending emails, fetching emails, and checking for an incoming email. Supplementary features that are commonly available to the email client, such as message deletion, draft management, and extensive message formatting capability will be developed in future work.

The system has been configured using the author's personal email address, because an email account is a personal matter and is confidential. It would be a challenge to use another user's account details, as permission would be needed.

The system has been tested under the Windows and MacOSX operating systems. Since it was developed using Java, the system should work well as long as a Java runtime environment has been installed by the client. However, there should be a DBMS running under the service in order to store the data in the database.

With respect to inbox management, there are occasions where the messages form a conversation, which means it could have the attributes of a thread. It is ideal to group these messages in a conversation thread, as can be seen in the modern email client. Therefore, this issue can also be addressed in the future.

3.6.2 User Interface

The user interface is of a simple design to serve the purpose of providing basic functionality and to show "proof of concept". For instance, in the message editing window, no enhanced rich text functionality is implemented. Instead, the user is presented with a basic text box form.

3.7 Summary

In this chapter, we presented a smart email client prototype design adapted from the CBR approach. The architecture and components developed for the system were described. In our Java-based prototype, we utilised Apache Lucene as the IR component, JavaMail library as the email communication component, and JavaFX library as the GUI component. In addition, we explained the CBR cycle and its implementation in our prototype. Finally, we addressed several restrictions and limitations of our prototype from two points of views: the logic and the interface.

Chapter 4

Evaluation

In the previous chapters, we identified that similar problems might have similar solutions. Therefore, the retrieval strategy plays an important role in finding the most similar past problems (cases) before the solution can be reused. This chapter provides our experimental design for evaluating the retrieval algorithm. It begins with a description of the evaluation process, including the dataset used and the metrics applied. Then, we evaluate the algorithm to explore the influence of text processing and feature selection on the retrieval quality, index size, and processing time. After that, we summarise the results using visualisation. Finally, this chapter finishes with a discussion section, where we explore and analyse interesting findings based on the results obtained.

4.1 Evaluation Configuration

In our approach, whenever the user selects an email and starts replying using recommended reply, the retrieval system attempts to find the most similar past email, rank it according to its degree of similarity with the selected email, and

presents it back to the user. In other words, the user triggers an ad-hoc information retrieval process. The typical method for measuring ad-hoc information retrieval consists of: (1) A collection of documents, (2) A test suite, also known as the queries, and (3) The relevance judgments as a reference (similar to supervised learning), whether the test is relevant or irrelevant (Schütze, 2008). This section identifies our method in performing the evaluation of the retrieval system. We start by describing the objectives of the evaluation and the relevance metrics commonly used to measure the objectives. Then we explain the environment setup where we performed the experiments. Finally, we specify the dataset we used in this evaluation.

4.1.1 Evaluation methods

Retrieval phase is the first step in the CBR cycle and an effective retrieval process ensures the system generates the most similar past cases with respect to the given problem (Aamodt & Plaza, 1994). We identified three objectives in evaluating the retrieval system. First, it is important that our system returns only the most relevant results to satisfy the user needs, according to the user query. This is known as the retrieval effectiveness (Chapelle et al., 2009). Second, the more cases stored in the case base, the bigger the index size grows. Since cases are indexed using the terms, it seems important to maintain the size of the index by only storing the most representative terms. Therefore, we also measure the index size of the system. Third, it is also essential to provide a seamless response as the user triggers an ad-hoc retrieval using a query. Thus, we record the time required to process a query. While this measurement may vary according to the hardware specifications, the results generated can still provide a general notion of the system speeds. This is further considered as retrieval efficiency (Chapelle

et al., 2009).

Figure 4.1 depicts a flowchart of how the sequence of the experimentation phase is implemented. We performed an initial analysis of the dataset and obtained the term distribution. Then, we selected the case feature (or attribute) from the dataset along with a text analysis method to conduct the experiments. We consider the combination of both elements as the experiment trials.

Afterwards, we performed retrieval evaluation using a list of queries obtained from the dataset. These queries also have corresponding right answers, therefore we can identify whether the provided answers match with the results given by our retrieval algorithm. At the end of each trial, we recorded the results by counting the number of terms indexed, the processing time, and the top 10 most similar email messages ranked by their similarity scores. In addition, we also calculated the Mean Reciprocal Rank (MRR) score for that query. More details about MRR are presented in the next section.

4.1.2 Evaluation metrics

As discussed in the previous section, we identified three main objectives in this evaluation; the retrieval effectiveness, the index size, and the retrieval efficiency. In this section, we explain the corresponding metrics used for each objective respectively.

To measure the effectiveness of the retrieval process, we used Reciprocal Rank (RR). This is a relative score that calculates the average or mean of the inverse of the ranks at which the first relevant document was retrieved for a set of queries (Voorhees & Tice, 1999). For instance, if a search for a specific query returns a relevant document at the 1^{st} position, its relative rank or RR is 1. If the relevant document is at position 2, then the score is 0.5 and so on. If there

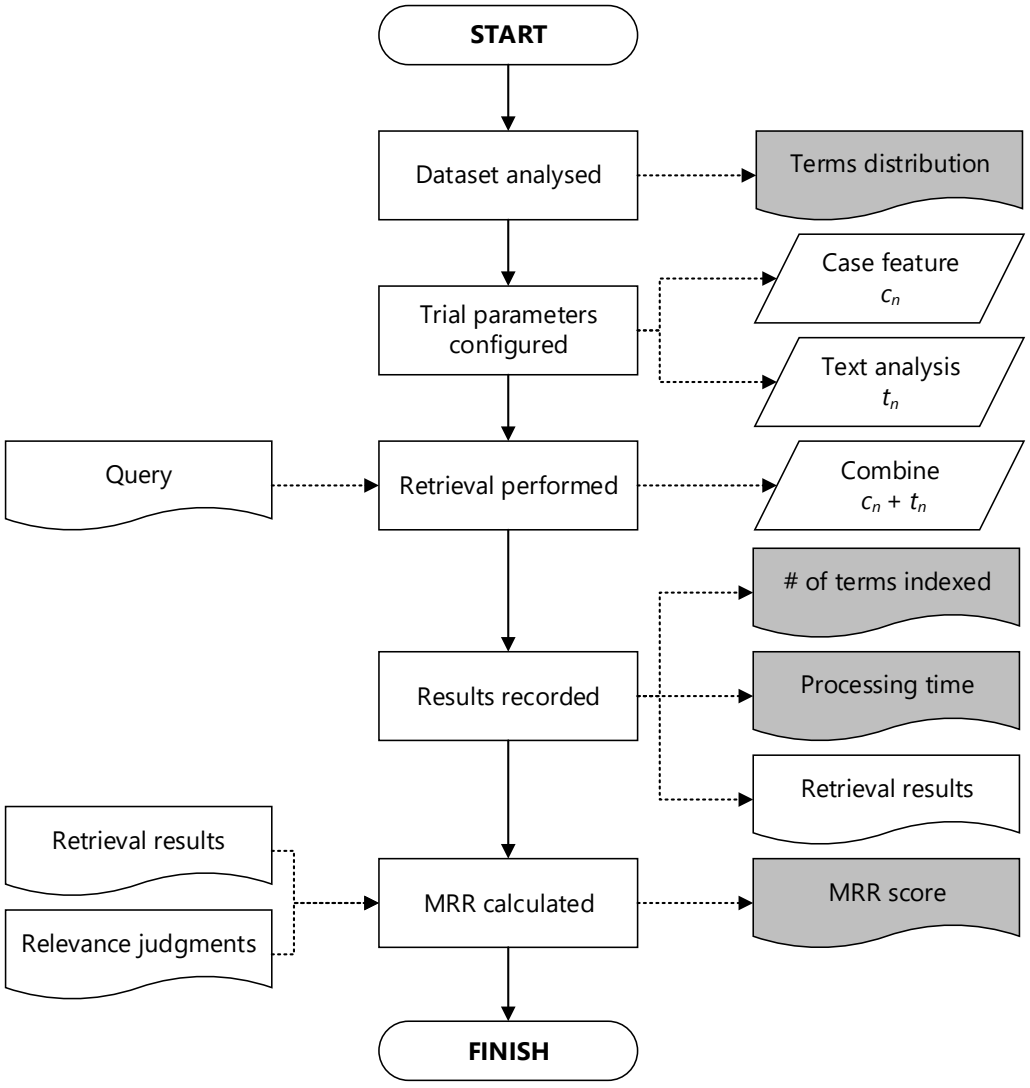


Figure 4.1: Diagram showing summary of our experiments process

are no relevant documents then the score is 0.

When averaged across the set of queries, this measure is called the Mean Reciprocal Rank (MRR). It is associated with the use case where the user wishes to see only one relevant document for a search, subsequently assuming that the user will keep scrolling down on the search results until the first relevant document is found. Thus the document is found at rank n, and the quality of

the retrieval is measured by the reciprocal of the rank (i.e. 1/n).

We observed that each user query has one correct answer and the assumption is that the user will stop searching once the correct document, based on his/her preference, is found. Alternatively, the time required by the user to find the relevant document corresponding to the search is inversely proportional to the rank (Voorhees & Tice, 1999). In this case, the better the rank, the lower the time taken by the user to get to the relevant document. For instance, an MRR score of 0.8 indicates that the information retrieval system is 80% relevant.

In addition, Mean Average Precision (MAP) is also a popular scoring method used in measuring the effectiveness of an information retrieval system (Voorhees, 2000). It is used mostly when the user is expecting more than one possible relevant result for their search query (Voorhees, 2000). However, since in our system we assume that the user chooses only the most relevant answer from the provided recommendation list, we excluded this metric in our evaluation.

After identifying the retrieval effectiveness measurement, our next objective was to evaluate the case base index size. We measured the size of the case base according to the size of indexed terms and counted a number of distinct terms in the index files. Finally, an evaluation of the retrieval efficiency was performed by obtaining the processing time that elapsed in the system while performing the retrieval process on a query.

4.1.3 Dataset description

In order to evaluate our system in conditions closer to a real world scenario, we considered the following components to find the most appropriate dataset. First, since we are dealing with email problems, it is best to use a dataset of email messages. However, due to privacy issues around the nature of email

communication, it was challenging to find a publicly available email dataset. Until recently, there was only one realistic email dataset available and that has been widely used for research purposes. The dataset was collected from a company that went bankrupt called Enron. Originally, this dataset was published during an investigation into the bankruptcy, then, after the data was compiled and cleaned up, it was made available to be used in research (Klimt & Yang, 2004)

Our second concern in selecting the dataset was that it should have relevance judgments, meaning the dataset has been manually annotated by experts. Therefore, when we perform the retrieval evaluation, we already have the list of an email and its similar or adjacent pairs. We can then compare our retrieval results with that list.

Interestingly, a study has been done that used a subset of the Enron dataset [1] and annotated it (Minkov et al., 2006). Initially, the authors used this dataset for person name disambiguation and an intelligent email threading task. While they examined both email headers (e.g. recipients, date, subject) and email content, we only used the subject and the content in our evaluation. Mailbox from two Enron employees was prepared: `germany-c` and `farmer-d`. Table 4.1 shows an example of the email and other similar other emails in the repository, which annotated by a researcher.

Table 4.1: A query and its annotated relevant response from farmer-c mailbox

Message	**ID**	**Subject**	**Body**
Query	164	Re: HILCORP old ocean volume	Gary thinks that we will not have any problems with this. D
Relevant response	166	Re: HILCORP old ocean volume	Are we going to have some quality problems with this gas? EDG

[1] http://www.cs.cmu.edu/ einat/datasets.html

The details of both datasets are provided in Table 4.2. It can be seen that both contain a similar number of email messages. While the `germany-c` dataset has slightly more messages than in `farmer-d`, it has a lower average word count, reflecting the content of that email. On the other hand, `germany-d` contains a blank subject in the message, whereas in `farmer-d`, the blank parts are mostly found in the body message.

Table 4.2: Datasets used in this evaluation

Dataset	**germany-c**	**farmer-d**
Overall		
Count of email messages	2651	2642
Count of annotated similar email pairs	81	127
Average word count per message	48.36	53.38
Field: Subject		
Blank messages	91 (3.4%)	41 (1.6%)
Total word count	10018	11916
Field: Body		
Blank messages	276 (10.4%)	289 (10.9%)
Total word count	113778	121514

Initially, the dataset is provided in the text-based format. Therefore, before proceeding with the evaluation, the dataset need to be changed into a format that can be imported to the database. We stored the records in the database instead of the text files so it would match the approach in our real-life implementation. In the early steps, we trimmed unused information and adjusted the structure to a comma-separated value (CSV) format. We also replaced the missing values with the word "null" to avoid errors during the indexing process. We were aware that its appearance would have no significant effect on the term weighting calculation. Although the word "null" is indexed, our scoring system would give a low score to terms that appear too frequently, both in that document and in the whole corpus.

4.1.4 Experiment configuration

The experiments were performed in a machine with the following specifications: 4.6GHz Intel Core i5 with 16GB RAM memory, 500GB solid-state drive (SSD), and running Windows 10. The evaluation module was implemented in Java SE version 8.

In addition to the machine specification, we prepared a spreadsheet document to manage the results, as shown in Figure 4.3, along with an example of the recorded retrieval process. In this document, we list the query and its corresponding relevance judgment, as provided in the dataset. Next, we specified columns to store our retrieval results, their similarity scores, and the processing time over three repetitions. We then matched the retrieved results with answers from the relevance judgment and found its position in the rank according to the similarity score. Finally, we calculated the RR score with respect to the rank.

Table 4.3: Template used to record experiment results

Query	**Answer**	**Retrieved Results** *(1-10)*	**Rank**	**Reciprocal Rank**	**Similarity Score** *(1-10)*	**Runtime** *(1-3)*
messageId	*messageId*	*messageId*	*0-10*	*0-1*	*0-1*	*in nanosecond*
Example						
23	***33***	21 ***33*** 287 ... 3730	2	0.5	1 0.914552 0.831411 ... 0.673435	1655383632 1576946565 1661965385
109	...					

4.2 Results

In this section, we presented the results of evaluating the retrieval algorithm. We performed an initial exploration of the dataset by analysing its term distribution. Then, as mentioned in the previous section, we described the results based on three indicators that we measured in this evaluation: the quality of retrieval results, index size, and processing time elapsed. Finally, we identified several interesting issues arising from the evaluation results and discussed the justification.

4.2.1 Dataset distribution

Before we started the experiment, we performed an initial analysis to see the distribution of the terms in our dataset. To do this, the email messages were tokenised to obtain the bag of words and stored them as index terms. These terms were counted and ranked with respect to the frequency of occurrence in each document in the corpus. We only kept terms that have a frequency of at least two in the corpus. We considered trimming this to provide better distribution visualisation.

The term frequency calculation was obtained using Luke [2]. This is an open source GUI tool that examines the Lucene index files. The interface of the tool is shown in Figure 4.2. We can select the feature indexed, either subject or body, and copy the results from the table on the right-hand side of the pane.

The term frequency obtained is meaningful for two reasons. First, we can see that some of the most frequent terms that occur in the corpus are stopwords. Therefore, we considered examining the stopwords removal influence in our experiment.

[2]https://github.com/DmitryKey/luke

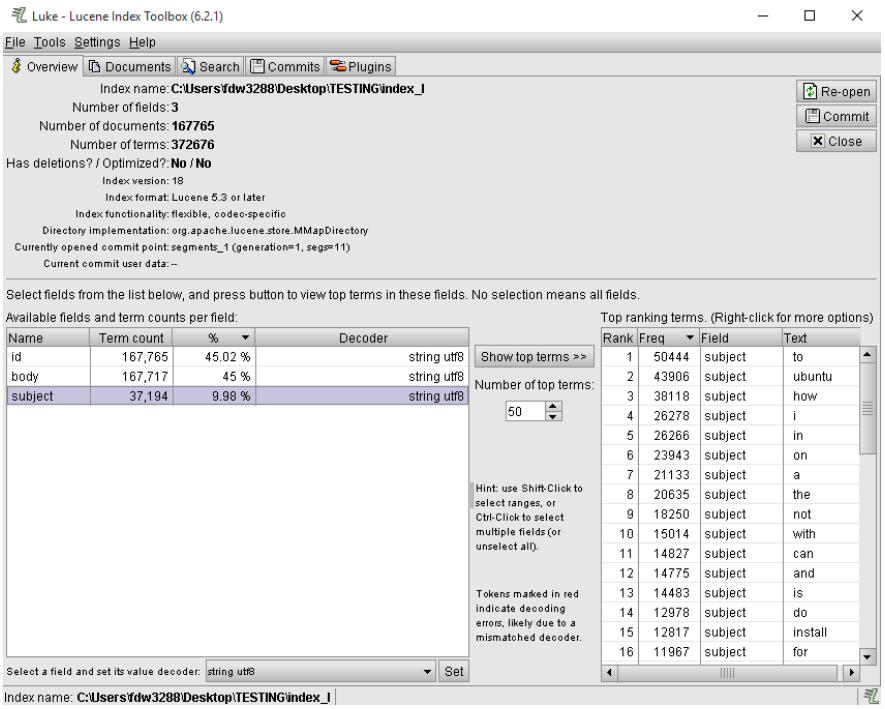
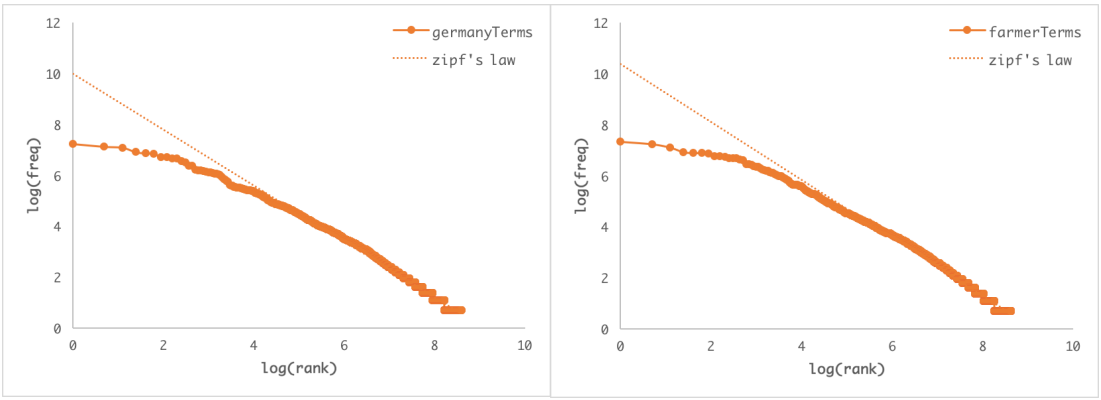


Figure 4.2: Luke: An open source tool to read Lucene index files

Second, we compared our term distribution with Zipfian Law, which is also commonly referred to as Zipf's distribution. In the implementation of this distribution in the linguistics domain, the frequency of a word is inversely proportional to its rank in the frequency count. A study by Ha et al. (2002) demonstrated Zipf's distribution in a corpus using a graph. We also attempted to observe the data according to this distribution by plotting it on a log-log graph, were each log represents log(rank order) in the y-axis and log(frequency) in the x-axis.

It can be seen from Figure 4.3 that the term distribution is a near fit to Zipf's. The deviation seems to appear in the most frequent words since the plot merges with the trend line at half way. Both datasets show similar behaviour, while slight differences are only present between the log (rank) of 2 and 4.



(a) germany-c (b) farmer-d

Figure 4.3: Dataset distribution according to Zipf's Law distribution

4.2.2 Experiment results

Exploring the term distribution provides the insight that some frequent words are stopwords. Therefore we considered dealing with this issue using text analysis techniques. The following are the results categorised by the three indicators mentioned in the earlier chapter. We describe the results using charts and tables. In this way it is possible to perform a comparison between the experiment's methods and to examine the trends in the results.

Retrieval effectiveness

The retrieval effectiveness of our algorithm was measured by comparing the retrieved results with the relevance judgments given for each corresponding query, as annotated by Minkov et al. (2006). The MRR score reflects the retrieval effectiveness according to the score. The higher the score, the more relevant the results retrieved by our algorithm.

Furthermore, Table 4.4 presents the detailed MRR score from 18 different trials. Overall, the highest MRR score was achieved by applying the `SA` method to the case feature subject. Then, combining the features subject and body as

all during retrieval still yields a higher MRR score compared to that in the body only. It is interesting to see that the `SA_NOSTM` method gives a better score when implemented in all case features.

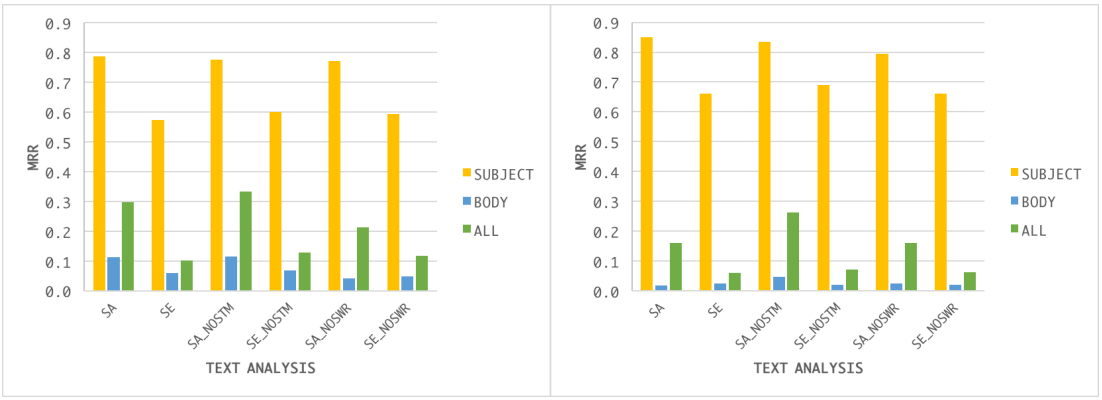
Table 4.4: MRR score from experiments in both dataset

MRR score	SA[1]	SE[2]	SA_ NOSTM[3]	SE_ NOSTM[4]	SA_ NOSWR[5]	SE_ NOSWR[6]
Lexical analysis	✓	✓	✓	✓	✓	✓
Stopwords removal	✓	✓	✓	✓		
Stemmer	✓	✓			✓	✓
Synonym expansion		✓		✓		✓
Germany-C						
All	0.2970	0.1026	**0.3326**	0.1287	0.2126	0.1177
Body	**0.1126**	0.0592	0.1146	0.0680	0.0416	0.0491
Subject	**0.7881**	0.5741	0.7757	0.6008	0.7716	0.5936
Farmer-D						
All	0.1610	0.0608	**0.2627**	0.0719	0.1591	0.0618
Body	0.0177	0.0234	**0.0457**	0.0195	0.0245	0.0209
Subject	**0.8543**	0.6647	0.8340	0.6890	0.7940	0.6604

[1]Standard analysis, [2]Synonym expansion
[3]Standard analysis without stemmer, [4]Synonym expansion without stemmer
[5]Standard analysis without stopwords removal, [6]Synonym expansion without stopwords removal

In order to provide better visualisation for comparison, Figure 4.4 depicts the summary of the results in a chart. It is apparent from this chart that the MRR score shows a decreasing trend when the `SE` method is applied in the trials, in spite of the stemming and stopwords removal techniques. Moreover, the results also affirm that the case feature subject returns more relevant emails that match the relevance judgments. This is followed by all and the body.

While the MRR score was used as the primary metrics for measuring retrieval effectiveness, we also examined the cosine similarity score generated from matching a query with documents in the corpus. This examination was performed for both datasets, `farmer-d` and `germany-c`, to determine the influence of synonym expansion. Thus, `SA` represents results from standard analysis techniques, while `SE` enhances standard analysis `SA` with synonym expansion.



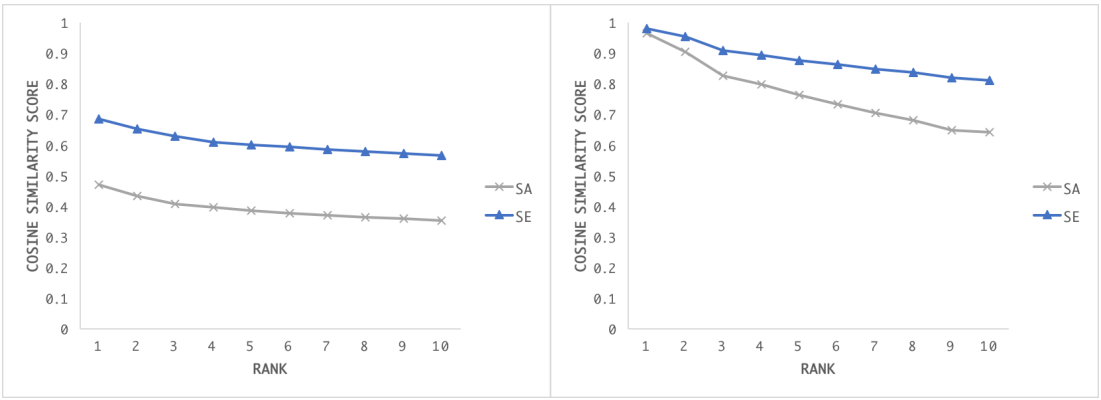
(a) germany-c (b) farmer-d

Figure 4.4: Retrieval effectiveness as represented by MRR score

We particularly observed the trends from the top 10 retrieved results with the highest cosine similarity score on two case features: body and subject.

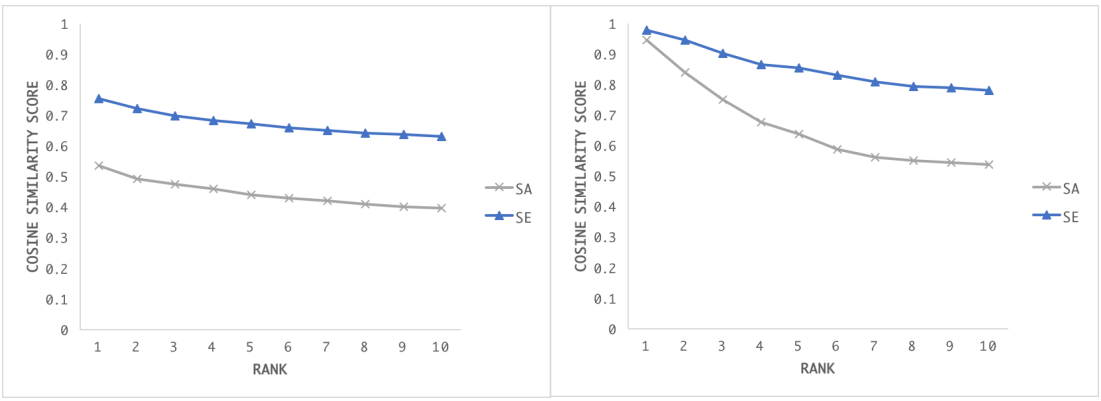
The observations from Figures 4.5 and 4.6 show that both datasets provided a similar decreasing trend in their respective body and subject features as the ranking goes lower from 1 to 10. This decreasing trend seems linear in the case feature body. However, the score in the case feature subject, as depicted in Figures 4.5b and 4.6b, shows gradually higher disparities between `SA` and `SE` compared to the results in Figures 4.5a and 4.6a. Thus, we could conclude that synonym expansion `SE` outperforms the standard analysis `SA` techniques in terms of the cosine similarity score. As a result, it could generate a higher chance of relevant matching, as shown in the higher score, compared to `SA`.

According to the results presented previously, it can be seen that the decreasing trends occur over the higher rank, despite the case features. This is possibly due to the sorting of the retrieval results according to the highest similarity score. However, it also seems interesting to examine the percentage of increase before and after applying synonym expansion. This increase is calculated from cosine similarity score of `SA` and `SE`. Given the same query, when the score of `SA` and `SE` is 0.5 and 0.75 respectively, the rate of increase 50%.



(a) body (b) subject

Figure 4.5: Average similarity score over top 10 results in germany-c dataset



(a) body (b) subject

Figure 4.6: Average similarity score over top 10 results in farmed-d dataset

In contrast to the decreasing trend over the higher ranks in the cosine similarity score, Figure 4.7 depicts that nearly 40-60% of the increasing rate appears in the body feature, with a relatively low disparity between both datasets. However, this disparity gradually increases in the subject feature. This validates our previous findings that synonym expansion could improve the chance of retrieving more relevant matching, even in the lower ranks, by increasing the cosine similarity score up to 30% in the subject and up to 60% in the body.

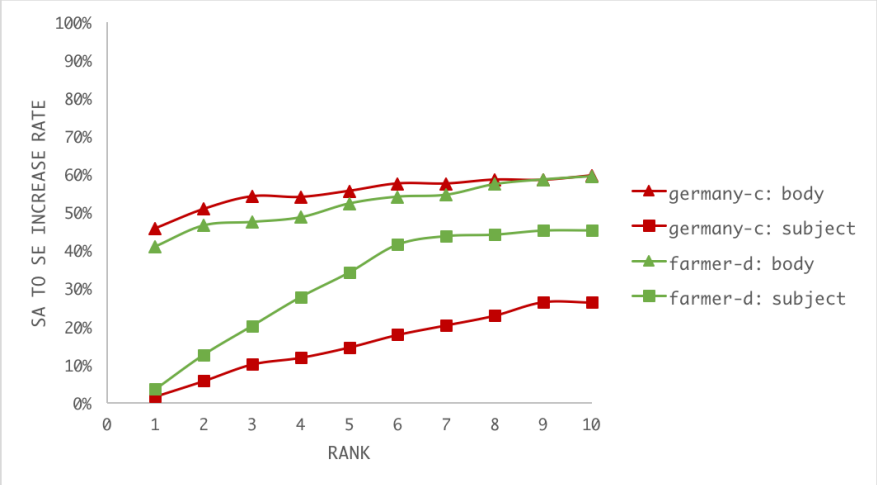


Figure 4.7: Similarity score increase rate before (SA) and after (SE) applying synonym expansion with respect to case features body and subject

Although SE could increase the cosine similarity score, its MRR score shows the opposite. This might be related to the relevance judgments provided by Minkov et al. (2006). Since it is highly dependent on human annotation, it might affect the MRR score. This is further explained in the discussion section.

Index size

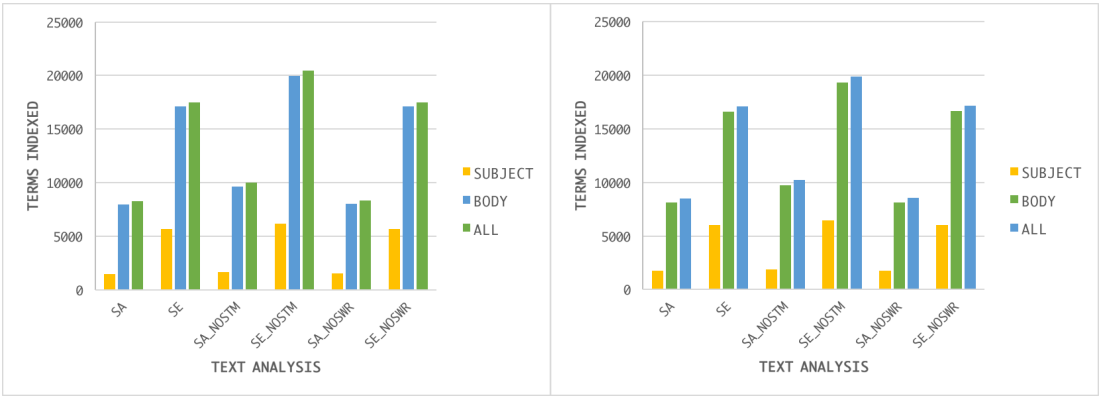
In addition to measuring the retrieval effectiveness, we also examined the influence of text analysis and case feature selection on the size of the index. According to Table 4.5, the SE_NOSTM method generated the highest number of terms over all case features. In addition, the case feature all had the most terms, whereas the subject seems to have the least.

It is also shown in Figure 4.8 that a significant amount of terms are generated by synonym expansion SE method. In addition, by not applying the stemming method, as shown in SA_NOSTM and SE_NOSTM, more terms are generated compared to other methods.

Table 4.5: Total terms indexed from experiments in both dataset

Terms indexed	SA[1]	SE[2]	SA_ NOSTM[3]	SE_ NOSTM[4]	SA_ NOSWR[5]	SE_ NOSWR[6]
Lexical analysis	✓	✓	✓	✓	✓	✓
Stopwords removal	✓	✓	✓	✓		
Stemmer	✓	✓			✓	✓
Synonym expansion		✓		✓		✓
Germany-C						
All	8292	17474	9994	**20456**	8325	17507
Body	7999	17101	9622	**19978**	8032	17134
Subject	1482	5678	1651	**6187**	1512	5709
Farmer-D						
All	12060	17110	10200	**19864**	8561	17143
Body	11214	16618	9746	**19296**	8154	16651
Subject	2317	6027	1898	**6455**	1766	6052

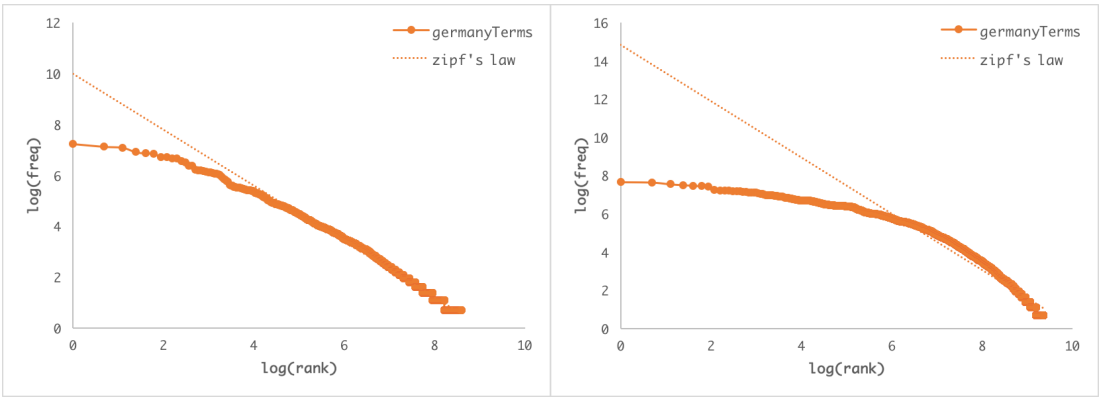
[1]Standard analysis, [2]Synonym expansion
[3]Standard analysis without stemmer, [4]Synonym expansion without stemmer
[5]Standard analysis without stopwords removal, [6]Synonym expansion without stopwords removal



(a) germany-c (b) farmer-d

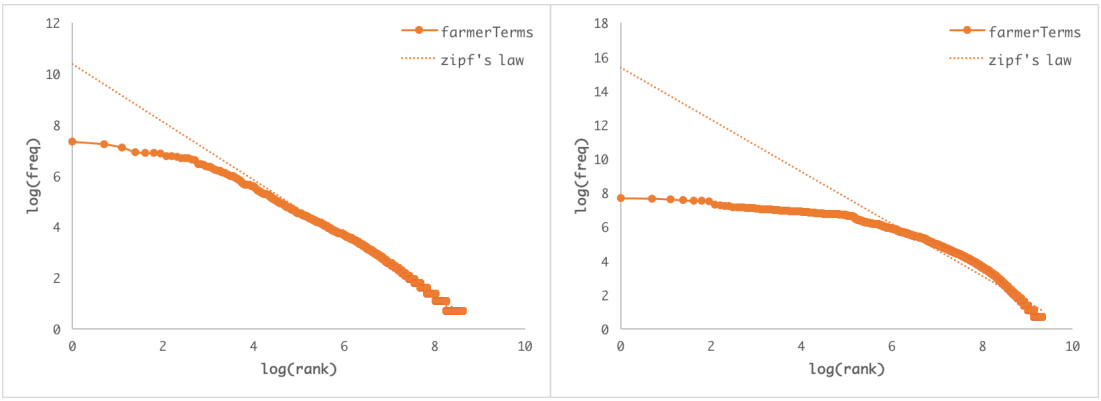
Figure 4.8: Index size as represented in a count of terms indexed

While the collected terms are used in measuring index size, we can see a different perspective on term distribution by plotting the term frequency before and after we apply synonym expansion. Figures 4.9 and 4.10 illustrate the term distribution before and after the synonym expansion was applied. Both present a flatter distribution in the higher ranks and fit Zipf's distribution in the lower frequency.



(a) before applying synonym expansion (b) after applying synonym expansion

Figure 4.9: Influence of synonym expansion for term distribution in germany-c



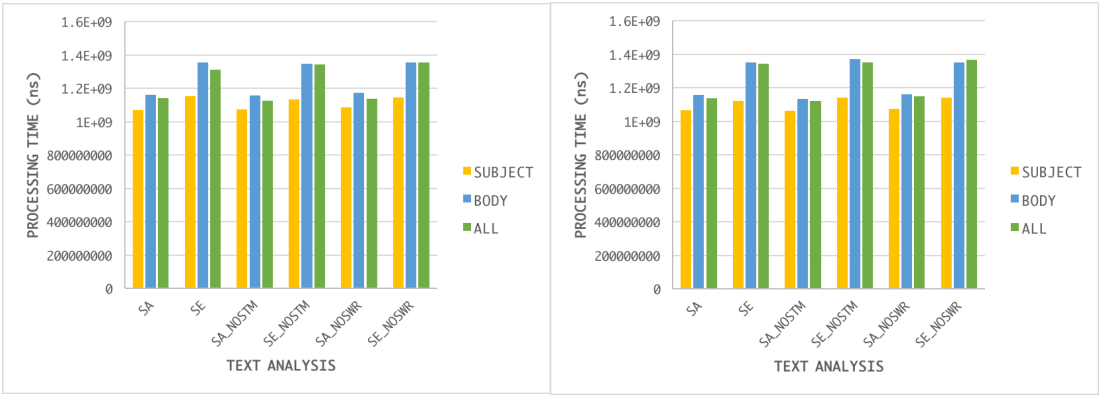
(a) before applying synonym expansion (b) after applying synonym expansion

Figure 4.10: Influence of synonym expansion for term distribution in farmer-d

Retrieval efficiency

Finally, we also measured the retrieval efficiency as represented by the processing time elapsed for executing a retrieval process in each query. We argued in the previous section that although the processing time might be highly dependent on the hardware used, we can still compare the influence of the different text analyses performed in the trials. We obtained the results by averaging the processing time from three trial repetitions. This processing time was generated using the Java *nanotime* library, which provided a more precise calculation as it was captured within nanoseconds.

In Figure 4.11, it is clear that the retrieval process that used the case feature subject was the fastest above all other features. In addition performing synonym expansion, as shown in any SE methods, generates a slightly slower processing time. Overall, applying stemming and stopwords removal seems not to significantly affect the processing time.



(a) germany-c (b) farmer-d

Figure 4.11: Retrieval efficiency as represented in processing time elapsed

4.3 Discussion

In the previous section, we presented the results obtained from the evaluation process. Therefore in this section, we provide a further justification of the results. We explore the influence of text analysis, case feature selection, and the dataset.

4.3.1 Influence of text analysis and case feature selection

During the experiment, we performed trials using several text analysis techniques. First was the lexical analysis. This technique treats whitespace, symbols, and punctuation marks. Table 4.6 shows the implementation of this technique on the query by removing an apostrophe (’), a dot (.), and a slash (/). As all the

words in the content were transformed to lowercase, there is a likelihood of increasing the chance of matching that particular word. This normalisation is typically useful when an abbreviation is misspelt, for instance, "EtNG" instead of "ETNG".

Table 4.6: Lexical analysis: words are treated equally by normalisation

No Lexical Analysis	**Case feature: Body**
Query #1	I know it, you know it, who's telling all the single women??
Query #2	I have based this transport against Tennessee LA Index, as follows April May TGP Index 2.83 3.03 Fuel/commodity .138 .145 TGP Demand .02 .02 Gas Cost into ETNG 2.988 3.195
With Lexical Analysis	**Case feature: Body**
Query #1	i know it you know it whos telling all the single women
Query #2	i have based this transport against tennessee la index, as follows april may tgp index 283 303 fuelcommodity 138 145 tgp demand 02 02 gas cost into etng 2988 3195
Potential match #1	I know, tell all the women you know.

While lexical analysis technique allows terms to be "treated" as equal, it can be seen that the chance of matching could be improved by applying the second technique, stemming. For instance, the word "telling" could match "tell" if the suffix -ing is chopped. Another example of stemming implementation in our evaluation is shown in Table 4.7, where term "handling" can match "handle" if both terms are stemmed. The new term "handl" seems to be an unknown English word for human, but the computer could process it. The Porter stemmer simplifies this analysis for computer processing (Porter, 1980). It should be considered however, that the stemming process would typically be done after the synonym matching process, otherwise, the chance of matching a word with

its synonym pairs might be less likely. The stemmed word "handl" might have no match, but "handling" might.

Table 4.7: Advantages of stemming: more discovery of matching terms

No Stemmer	**Case feature: Body**
Query	Chris, Isabel Resendez is handling *Ashland Chemical*, her ext. is 3-0440. Jeff
Retrieval result	Who handles the billing for *Ashland Chemical*?
Matching terms	*Ashland, Chemical*
With Stemmer	**Case feature: Body**
Query	Chris, Isabel Resendez is *handl Ashland Chemical*, her ext. is 3-0440. Jeff
Retrieval result	Who *handl* the bill for *Ashland Chemical*?
Matching terms	*handl, Ashland, Chemical*

Furthermore, not all the terms should be indexed. There are set of words that are likely to appear frequently in textual content and these are commonly identified as stopwords. While humans have no issue with this, a computer considers these words as less rare in term weighting if they occur too often. The appearance of stopwords in our dataset can be seen in Table 4.8. Our results are in line with Baeza-Yates & Ribeiro-Neto (1999), which shows that by removing stopwords, we could compress the index size.

Interestingly, we found that applying lexical analysis and stopwords removal generally improved the number of relevant retrieved emails, as shown in the increase of the MRR score. It seems possible that the removal of these words could reduce the noise in the content and might also increase the chance of distinct terms to be fairly calculated while comparing between two documents.

Another text analysis technique used in evaluation is synonym expansion.

Table 4.8: Advantages of stopwords removal: reduced number of terms indexed

No stopwords removal	**Case feature: Body**
Query	Daren would you look *at the* price *for* 29 *and* 30th *of* March 2000. *There are no* prices *for these* days *but* volume *was* scheduled *for these* days. Thanks Charlene
Retrieval result	Charlene, We should *not* have scheduled any volume *for the* 29-30th. *No* price *was* negotiated *that* I can find. However, since gas did flow, I rolled the last price *on the* deal *to* cover those days. What price *is* Hesco showing? D
Stopwords found	at, the, for, and, of, no, these, but, was, that, on, not, to, is, are, there
Index size	54
With stopwords removal	**Case feature: Body**
Query	Daren would you look price 29 30th March 2000. prices days volume scheduled days. Thanks Charlene
Retrieval result	Charlene, We should have scheduled any volume 29-30th. price negotiated I can find. However, since gas did flow, I rolled last price deal cover those days. What price Hesco showing? D
Index size	37

This technique is performed by matching terms with their corresponding synonyms. In our experiment, we utilised WordNet as additional linguistic knowledge, and this provided the database of semantic relations. In an early observation, we found this technique could extend the capability of exact matching by searching through a word's synonym pairs too, as shown in Table 4.9

Table 4.9: Synonym expansion: extending the capability of exact matching

Message	**Case feature: Body**
Query	It's *fine* with me to allocate everything to Torch if they are paying for it. D
Synonym found	fine ⇒ alright, o.k., *ok*, okay, cool, very well
Retrieval result	That's *fine*. D
Retrieval result	HPL is *ok*. D
Retrieval result	So, is this *ok* now? D

The next observation shows a trend where experiments that use synonym expansion tend to show an increase in the retrieval execution time for each query. This might be due to the larger number of terms to be calculated between the documents. The increasing number of terms, as described in Table 4.10, occurred because the synonym words were also considered in the similarity calculation.

Table 4.10: Expanding terms with their corresponding synonyms in the content

Message	**Case feature: Body**
Query	Should this deal be extended?
Terms calculated	3 ⇒ (should, deal, extended)
Synonym #1 found	deal ⇒ trade, bargain, address, consider, take, cope, manage, care, handle
Synonym #2 found	extended ⇒ widen, broaden, extend, continue, stretch, expand, continue
Terms calculated	19 ⇒ 7 (synonym "deal") + 9 synonym "extended" + 3 (should, deal, extended)

We also observed that different selected feature could affect the retrieval of relevant messages. By using the case feature subject, the retrieval effectiveness yielded a higher score compared to when the body was used. There could be several possible explanations for this result. First, the subject line is likely to have fewer words, which might lead to a higher chance of matching keywords.

Second, we found that the subject content contains a relatively uniform pattern. For instance, in the given relevance judgments, the query and its relevant judgment are shown in the Table 4.11. It is a trivial problem for the algorithm if it only considers the subject, because of the high number of exact matching words.

As can be seen from the experiment results, it is challenging for the case feature body to generate a higher MRR score since our retrieved email mostly did not match the relevance judgments. However, our observations showed

Table 4.11: Retrieval results using "subject"; italic indicates matching keywords

Message	**Subject**
Query	Re: HPL Meter #980074 Bammel HPL D/P to Transco
Relevance judgment	HPL Meter #980074 Bammel HPL D/P to Transco
Retrieval result rank #1	*HPL Meter #980074 Bammel HPL D/P to Transco*
Retrieval result rank #2	*Re: HPL Meter* #980070 RUSK *D/P* - LONE STAR HPL
Retrieval result rank #3	*HPL Meter* #981525 Texoma *D/P*- GSU HPL

that applying the synonym expansion technique could increase the chance of matching relevant results, even in the lower rank. This finding is similar to Abdalgader & Skabar (2010), who argue that synonym expansion leads to an improvement in the similarity measurement. Hence, we presume that this might be related to the collection method implemented by Minkov et al. (2006) when building the relevance judgments. In the next section, we discuss our justifications for this issue.

4.3.2 Dataset critique

Earlier in this chapter we talked about the challenge of performing an evaluation using a real-life email dataset due to the privacy concerns. However, we overcame this by discovering that the Enron email dataset is available online. This dataset came in a collection of raw messages, until some researchers annotated them for study purposes (Minkov et al., 2006).

When we performed an evaluation using this dataset, we found that the results from the subject field tended to be high, as seen in the MRR score when compared to that of the body feature. We presume that the process of performing a relevance judgment might be to consider the small number of features instead of all features in the email message. Then we found that, according to the

authors, the relevant messages were annotated using the subject line and time stamp only (Minkov et al., 2006). This also explains why using the feature all still generated higher result than the body. This could be due to the combination of the content subject and body, but it still provides a higher chance of matching the relevance judgments.

We also noticed that in a number of messages, the subject line contains fewer meaningful words in terms of context. We consider this as less meaningful since they have an empty value or are only one word with three or fewer characters. This is described in Table 4.12, along with the frequency of the corresponding messages in the corpus. While it might reflect the reality of email communication, where people are likely to reply using the same subject, it is challenging in a retrieval task, which considers the subject only. Thus, it is important to consider another field such as the body itself.

Table 4.12: Summary of messages with less meaningful context

Subject line	**Number of messages**	**Occurrences in corpus**
Germany-C		
<null>	97	3.7%
Re:	81	2.6%
Hey	81	1.1%
Yo	27	1%
Farmer-D		
<null>	41	1.5%
Re:	19	0.7%

We also attempted to analyse our retrieval results using the case feature body, which did not match with the relevance judgments. Table 4.13 shows that the relevance judgment content is less relevant compared to our results. This can be seen from the number of matching terms in the content. In addition, the context of the message in our retrieval results seems to be slightly more relevant from our point of view.

Table 4.13: Given relevance judgment compared to our retrieval results; italics indicates matching keywords

Message	**Body**
Query	Janet, *Please* submit this *name change* to the TPC as *soon* as *possible*. *Thanks*, hgm
Relevance judgment	Toni, Attached is the job posting for my group. *Please* process this as *soon* as *possible*. *Thanks*. Daren
Retrieval result	is it *possible* that his first *name* show as "J Darren" ? Daren is his middle *name* and the *name* that everyone addresses him by. We would also be glad to *submit* the *change* in SAP if the system will accept J Darren as the first *name*. *Please* advise. *Thanks* for your help, hgm
Retrieval result	Once this is fixed in SAP I will be able to *change* it in PEP. The Data Integrity Team is being really strict on the *name* field. It has to be *changed* in SAP before I am allowed to *change* it in PEP. Please let me know when this has been done and I will make the *change* asap. Enron Capital & Trade Resources Corp.

Finally, we concluded that this dataset is more appropriate for finding a relevant email by considering the email header, such as sender, recipient, date and time. The subject field is also suitable for use as a case retrieval feature. However, we found that the body field is less useful since the annotation process might not consider this field.

4.4 Summary

Earlier in this chapter, we stated that retrieval is the first step in the CBR cycle. We started by describing the evaluation configuration using a combination of text analysis and case feature. We used the email dataset with relevance judgments from a real-life scenario. Then, we summarised our results according to the objective of the evaluation: retrieval effectiveness, efficiency, and index size. We also assessed the compatibility of the dataset with respect to our results.

Our experiments showed that text analysis techniques such as lexical analysis, stemming, and stopwords removal enhance the effectiveness of our retrieval algorithm. Synonym expansion tends to be less effective„as reflected in the lower MRR score. This might be related to the relevance judgments provided in the dataset. In contrast, this technique improves the similarity score of the retrieval results, which align with the study conducted by Abdalgader & Skabar (2010). We found that the relevance judgments provided in the dataset might be less appropriate when evaluating the case feature body as during annotation, other features, such as subject and date, were considered (Minkov et al., 2006).

Chapter 5

Conclusions

5.1 Conclusions

Today email is still considered one of the most preferred channels of communication and is widely used for information exchange. As a result, a large number of emails are received every day by many people, leading to overwhelmed and cluttered email management. This phenomenon is commonly referred to as the email overload problem. One of the tasks discussed in this problem relates to the behaviour of a user in replying to the incoming email messages.

To address this problem, this thesis presents a smart email client; a desktop-based email client application enhanced with a reply recommendation system. This application is designed to assist users in replying to an incoming email, potentially considered as a query, by providing a reply message template. It starts from an assumption that a similar query might have a similar solution, and that these queries have probably been answered in the past. Therefore, users could reply to the query efficiently without writing from scratch. The system already provides a similar past reply as a starting template, so this should be achievable.

The nature of the problem that we are attempting to solve is closely related to a problem-solving method called Case-Based Reasoning (CBR). The CBR method solves a new problem by using the stored past cases, which consist of a problem description and the corresponding solution. The process of solving a new problem in CBR is organised in a 4R's cycle: Retrieve similar past cases from the case base, Reuse its solution, Revise the solution if necessary in order to fit to the problem, and Retain the solved problem along with its solution for future similar problems. Further, the implementation of CBR can use a variety of approaches. In this thesis, we noted that the problem domain in email is closely related to textual information. Thus, we employed techniques related to text-based processing, such as information retrieval and natural language processing.

One of the challenges encountered in this study was related to how the email domain can be mapped into CBR methodology. Through discussion, we found that the retrieval process plays an important role throughout the CBR cycle. Therefore, we decided to evaluate our retrieval algorithm in order to identify which combination of parameters provided the better results so we could implement them in our prototype. These parameters include various text analysis technique and case feature selection.

Our retrieval algorithm relies heavily on the terms found in the email message, since we utilised the Vector Space Model (VSM) approach. By calculating the term weights according to the frequency of occurrence, two documents can be transformed into vector space. Since these vectors form an angle, similarities between the two can be obtained by measuring the cosine angle. In the implementation, the ranked top 10 results with the highest similarity score are presented to the users.

Our evaluation methods also involved relevance judgments obtained from the email dataset. As each query was already annotated with the correct response, we could measure that against the number of matching queries from the retrieved results. According to the results, we found that standard text analysis techniques, such as lexical analysis, stemming and stopwords removal, enhanced our retrieval process.

In addition, our evaluation using the relevant judgments showed that most matched cases were retrieved when the algorithm used the case feature subject. However, compared to the standard analysis techniques, retrieval effectiveness was reduced when the synonym expansion technique was applied. In contrast, the similarity score was improved up to 30% in the case feature subject and up to 60% in the case feature body. This finding aligns with a study conducted by Abdalgader & Skabar (2010), who argued that synonym expansion leads to improvement in the similarity score. Thus, further investigation found that the relevance judgments provided by Minkov et al. (2006) may only be suitable for evaluating the case feature subject, not the body, since in the process of annotating, the only features considered were elements such as subject, recipients, and the date.

5.2 Future Work

In this thesis, we have described our proposed solution in the form of a smart email client application. The system design and implementation adapted the CBR approach. While the system was only developed for proof of concept, the reply content could be further enhanced by performing an automatic name extraction, using the information extraction technique. This information could be used at the beginning of the reply message as the salutation, for instance. On

the other hand, the name-based entity can be omitted from the reply message in order to provide a more generic reply message.

We also explained that the retrieval process has a significant influence on the CBR cycle. Thus, we performed an evaluation of our retrieval algorithm. Although our evaluation showed good results by implementing a standard text analysis (including lexical analysis, stemming, and stopwords removal), the retrieval algorithm still uses a linear search to compare email messages. The efficiency could be improved by identifying or grouping potential similar messages first instead of directly calculating the similarity score in each message. Thus the algorithm could narrow down the scope of the retrieval to a particular group. This kind of approach has been studied by Lenz & Burkhard (1996), combined information extraction and other techniques to form case retrieval nets (CRN).

Finally, we believe that more insights could be gained if testing involved a group of users. It might be interesting to see their behaviour when presented with the list of recommended replies. it would be useful to study the users' preferences when selecting the recommendations in order that they can be improved.

References

Aamodt, A. & Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, *7*(1), 39-59.

Abdalgader, K. & Skabar, A. (2010). Short-text similarity measurement using word sense disambiguation and synonym expansion. In *Australasian joint conference on artificial intelligence* (p. 435-444). Springer.

Abdrabou, E. & Salem, A. M. (2010). A breast cancer classifier based on a combination of case-based reasoning and ontology approach. In *Computer science and information technology (imcsit), proceedings of the 2010 international multiconference on* (p. 3-10). IEEE.

Ayodele, T. & Zhou, S. (2008). Applying machine learning algorithms for email management. In *Pervasive computing and applications, 2008. icpca 2008. third international conference on* (Vol. 1, p. 339-344). IEEE.

Ayodele, T. & Zhou, S. (2009). Applying machine learning techniques for e-mail management: Solution with intelligent e-mail reply prediction. *Journal of Engineering and Technology Research*, *1*(7), 143-151.

Baeza-Yates, R. & Ribeiro-Neto, B. (1999). *Modern information retrieval* (Vol. 463). ACM press New York.

Bergmann, R., Kolodner, J. & Plaza, E. (2005). Representation in case-based reasoning. *The Knowledge Engineering Review*, *20*(03), 209-213.

Białecki, A., Muir, R., Ingersoll, G. & Imagination, L. (2012). Apache lucene 4. In *Sigir 2012 workshop on open source information retrieval* (p. 17).

Chapelle, O., Metlzer, D., Zhang, Y. & Grinspan, P. (2009). Expected reciprocal rank for graded relevance. In *Proceedings of the 18th acm conference on information and knowledge management* (p. 621-630). ACM.

Coussement, K. & Van den Poel, D. (2008). Improving customer complaint management by automatic email classification using linguistic style features as predictors. *Decision Support Systems*, *44*(4), 870-882.

De Mantaras, R. L., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., ... Forbus, K. (2005). Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, *20*(03), 215-240.

Dredze, M., Brooks, T., Carroll, J., Magarick, J., Blitzer, J. & Pereira, F. (2008). Intelligent email: Reply and attachment prediction. In *Proceedings of the 13th international conference on intelligent user interfaces* (p. 321-324). ACM.

Díaz-Agudo, B., González-Calero, P. A., Recio-García, J. A. & Sánchez-Ruiz-Granados, A. A. (2007). Building cbr systems with jcolibri. *Science of Computer Programming*, *69*(1), 68-75.

Fisher, D., Brush, A. J., Gleave, E. & Smith, M. A. (2006). Revisiting whittaker & sidner's email overload ten years later. In *Proceedings of the 2006 20th anniversary conference on computer supported cooperative work* (p. 309-312). ACM.

Fox, C. (1989). A stop list for general text. In *Acm sigir forum* (Vol. 24, p. 19-21). ACM.

Grevet, C., Choi, D., Kumar, D. & Gilbert, E. (2014). Overload is overloaded: Email in the age of gmail. In *Proceedings of the sigchi conference on human factors in computing systems* (p. 793-802). ACM.

Ha, L. Q., Sicilia-Garcia, E. I., Ming, J. & Smith, F. J. (2002). Extension of zipf's law to words and phrases. In *Proceedings of the 19th international conference on computational linguistics-volume 1* (p. 1-6). Association for Computational Linguistics.

Heckler, M., Grunwald, G., Pereda, J., Phillips, S. & Dea, C. (2014). *Javafx 8: Introduction by example*. Apress.

Hewlett, W. R. & Freed, M. (2008). An email assistant that learns to suggest reusable replies. In *Aaai workshop, technical report ws-08-04* (p. 28-35).

Hliaoutakis, A., Varelas, G., Voutsakis, E. P., M., E. G. & Milios, E. (2006). Information retrieval by semantic similarity. *International journal on semantic web and information systems (IJSWIS)*, *2*(3), 55–73.

Klimt, B. & Yang, Y. (2004). The enron corpus: A new dataset for email classification research. In *European conference on machine learning* (p. 217-226). Springer.

Kolodner, J. (1991). Improving human decision making through case-based decision aiding. *AI magazine*, *12*(2), 52.

Kosseim, L., Beauregard, S. & Lapalme, G. (2001). Using information extraction and natural language generation to answer e-mail. *Data & Knowledge Engineering*, *38*(1), 85-100.

Lamontagne, L. & Lapalme, G. (2003). Applying case-based reasoning to email response. In *Iceis* (p. 115-123). Citeseer.

Lapalme, G. & Kosseim, L. (2003). Mercure: Towards an automatic e-mail follow-up system. *IEEE Computational Intelligence Bulletin*, *2*(1), 14-18.

Lenz, M., Bartsch-Spörl, B., Burkhard, H. & Wess, S. (2003). *Case-based reasoning technology: from foundations to applications* (Vol. 1400). Springer.

Lenz, M. & Burkhard, H. (1996). Case retrieval nets: Basic ideas and extensions. In *Annual conference on artificial intelligence* (p. 227-239). Springer.

Liddy, E. D. (2001). Natural language processing. *Encyclopedia of Library and Information Science, 2nd Ed*.

Lipscomb, C. E. (2000). Medical subject headings (mesh). *Bulletin of the Medical Library Association*, *88*(3), 265.

López, B. (2013). Case-based reasoning: a concise introduction. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *7*(1), 1-103.

Malik, R., Subramaniam, L. V. & Kaushik, S. (2007). Automatically selecting answer templates to respond to customer emails. In *Ijcai* (Vol. 7, p. 1659-1664).

Manning, C. D., Raghavan, P. & Schütze, H. (2008). *Introduction to information retrieval* (Vol. 1) [Book]. Cambridge university press Cambridge.

Manning, C. D. & Schütze, H. (1999). *Foundations of statistical natural language processing* (Vol. 999). MIT Press.

Martin, A., Uthra, R., Kavitha, A., Divya, B. & Venkatesan, V. P. (2012). A business intelligence model for predicting bankruptcy using cbr with essential features. *International Journal of Information Technology and Engineering*, *3*(1-2), 91-97.

Mayhew, D. J. (1999). The usability engineering lifecycle. In *Chi'99 extended abstracts on human factors in computing systems* (p. 147-148). ACM.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, *38*(11), 39-41.

Minkov, E., Cohen, W. W. & Ng, A. Y. (2006). Contextual search and name disambiguation in email using graphs. In *Proceedings of the 29th annual international acm sigir conference on research and development in information retrieval* (p. 27-34). ACM.

Nielsen, J. & Pernice, K. (2010). *Eyetracking web usability*. New Riders.

Osiński, S. & Weiss, D. (2005). Carrot2: Design of a flexible and efficient web information retrieval framework. In *International atlantic web intelligence conference* (p. 439-444). Springer.

Partridge, C. (2008). The technical development of internet email. *IEEE Annals of the History of Computing*, *30*(2).

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130-137.

Radicati. (2015). *Email statistics report, 2015-2019* (Tech. Rep.). The Radicati Group.

Robertson, S. E. & Jones, K. S. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science*, *27*(3), 129-146.

Rohall, S. L., Gruen, D., Moody, P., Wattenberg, M., Stern, M., Kerr, B., ... Wilcox, E. (2004). Remail: a reinvented email prototype. In *Chi'04 extended abstracts on human factors in computing systems* (p. 791-792). ACM.

Salton, G. & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval [Journal Article]. *Information processing & management*, *24*(5), 513-523.

Salton, G., Wong, A. & Yang, C. (1975). A vector space model for automatic indexing. *Communications of the ACM*, *18*(11), 613-620.

Schank, R. C. (1983). *Dynamic memory: A theory of reminding and learning in computers and people*. Cambridge University Press.

Schütze, H. (2008). Introduction to information retrieval. In *Proceedings of the international communication of association for computing machinery conference.*

Sneiders, E. (2016a). Review of the main approaches to automated email answering. In *New advances in information systems and technologies* (p. 135-144). Springer.

Sneiders, E. (2016b). Text retrieval by term co-occurrences in a query-based vector space. *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics*, 2356–2365.

Sneiders, E., Sjöbergh, J. & Alfalahi, A. (2016). Email answering by matching question and context-specific text patterns: Performance and error analysis. In *New advances in information systems and technologies* (p. 123-133). Springer.

Stahl, A. & Roth-Berghofer, T. R. (2008). Rapid prototyping of cbr applications with the open source tool mycbr. In *European conference on case-based reasoning* (p. 615-629). Springer.

Voorhees, E. M. (1986). The efficiency of inverted index and cluster searches. In *Proceedings of the 9th annual international acm sigir conference on research and development in information retrieval* (p. 164-174). ACM.

Voorhees, E. M. (2000). Variations in relevance judgments and the measurement of retrieval effectiveness. *Information processing & management*, *36*(5), 697-716.

Voorhees, E. M. & Tice, D. M. (1999). The trec-8 question answering track evaluation. In *Trec* (Vol. 1999, p. 82).

Watson, I. (1999). Case-based reasoning is a methodology not a technology [Journal Article]. *Knowledge-based systems*, *12*(5), 303-308.

Weber, R. O., Ashley, K. D. & Brüninghaus, S. (2005). Textual case-based reasoning. *The Knowledge Engineering Review*, *20*(03), 255-260.

Weng, S. & Liu, C. (2004). Using text classification and multiple concepts to answer e-mails. *Expert Syst. Appl.*, *26*(4), 529-543.

Whittaker, S. & Sidner, C. (1996). Email overload: Exploring personal information management of email. In *Proceedings of the sigchi conference on human factors in computing systems* (p. 276-283). ACM.