Winter 12-18-2015

Email Similarity Matching and Automatic Reply Generation Using Statistical Topic Modeling and Machine Learning

Zachery L. Schiller
University of Maine, zacheryschiller@gmail.com

EMAIL SIMILARITY MATCHING AND AUTOMATIC REPLY GENERATION USING STATISTICAL TOPIC MODELING AND MACHINE LEARNING

By

Zachery L. Schiller

B. S. University of Maine, 2011

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

(in Computer Science)

The Graduate School

The University of Maine

December 2015

THESIS ACCEPTANCE STATEMENT

On behalf of the Graduate Committee for Zachery L. Schiller I affirm that this manuscript is the final and accepted thesis. Signatures of all committee members are on file with the Graduate School at the University of Maine, 42 Stodder Hall, Orono, Maine.

Signature: ______________________________ Date:____________

Roy M. Turner

Associate Professor of Computer Science

School of Computing and Information Science

EMAIL SIMILARITY MATCHING AND AUTOMATIC REPLY GENERATION USING STATISTICAL TOPIC MODELING AND MACHINE LEARNING

By Zachery L. Schiller

Thesis Advisor: Dr. Roy M. Turner

An Abstract of the Thesis Presented

in Partial Fulfillment of the Requirements for the

Degree of Master of Science

(in Computer Science)

December 2015

Responding to email is a time-consuming task that is a requirement for most professions. Many people find themselves answering the same questions over and over, repeatedly replying with answers they have written previously either in whole or in part. In this thesis, the Automatic Mail Reply (AMR) system is implemented to help with repeated email response creation. The system uses past email interactions and, through unsupervised statistical learning, attempts to recover relevant information to give to the user to assist in writing their reply.

Three statistical learning models, term frequency-inverse document frequency (tf-idf), Latent Semantic Analysis (LSA), and Latent Dirichlet Allocation (LDA), are evaluated to find which approach works the best in terms of email document retrieval and similarity matching. AMR is built using the Python programming language in order to take advantage of tools and libraries specifically built for natural language processing and topic modeling. Datasets include the author’s work email and personal email archives, the publicly available *20 Newsgroups* dataset, and the recently released email archives of U.S. Secretary Hillary Clinton from the Freedom of Information Act website. In addition to different datasets and statistical modeling approaches, two different system tools, GenSim and SciKit-Learn, are also compared.

The outcome of this work is an initial version of the AMR system, which is freely available from the author’s Github page[1]. The core components of AMR input an email corpus, create a model of that corpus based on unsupervised learning and predict useful replies to new email based on the model. These pieces could be used as a toolkit for many different purposes. Although the best topic modeling approach is not definitively determined, this thesis concludes that using SciKit’s LSA implementation yields the most consistent results ($p < 0.05$) across the tested databases. These results could be used for future work on developing a more sophisticated product to accomplish a range of machine learning tasks.

[1] https://github.com/zacheryschiller/amr

ACKNOWLEDGMENTS

The author would like to thank his committee, and especially his advisor Professor Roy Turner, for the help and support they have provided him. He would also like to thank his friends and family for their love and support throughout all his life. Last but not least, the author would like to thank his amazing girlfriend Dominique Scarlett for continuously encouraging him to go after his dreams.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

Chapter 1

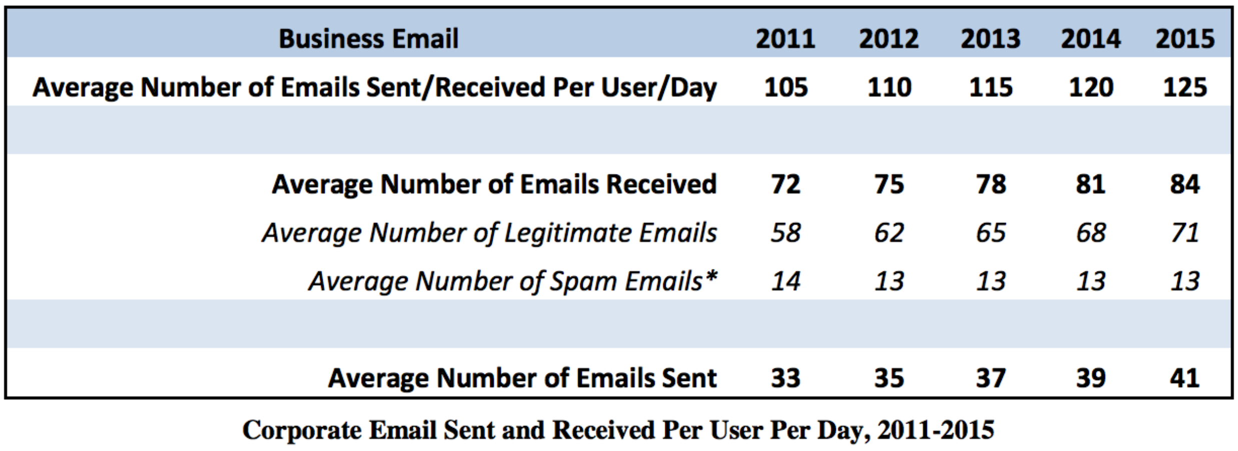
INTRODUCTION

1.1 Purpose

With the majority of workplace communication happening digitally, email has become a necessity. Unfortunately, reading and responding to email is also incredibly time consuming. Research done by the International Data Corporation found that workers spend ~28% of their workweek reading and answering email [1]. This is time that could be better spent working on tasks and projects.

There have been many attempts to make dealing with email quicker and easier, as seen in the related work section below. Most of these projects focus on filtering all of the messages in an inbox to only show relevant email that must be replied to. However, this is only one part of the problem. Another large part is actually creating the replies themselves. A report from Radicati Group [2] found that the average number of emails sent per person per day was 41 in 2015. This number of replies has been increasing since 2011, as seen in Figure 1.1. Since email is the dominant form of professional communication, it is likely that this number will continue to increase in the future.

In many cases, the responses that people write are very similar, if not identical, to responses they have written in the past. If there were a system that made this writing easier by providing suggestions based on previous interactions, it would significantly reduce the time wasted in rewriting the same responses.



Business Email	2011	2012	2013	2014	2015
Average Number of Emails Sent/Received Per User/Day	**105**	**110**	**115**	**120**	**125**
Average Number of Emails Received	**72**	**75**	**78**	**81**	**84**
Average Number of Legitimate Emails	*58*	*62*	*65*	*68*	*71*
*Average Number of Spam Emails**	*14*	*13*	*13*	*13*	*13*
Average Number of Emails Sent	**33**	**35**	**37**	**39**	**41**

Corporate Email Sent and Received Per User Per Day, 2011-2015

Figure 1.1. Average number of emails sent and received per user per day 2011-2015 [2]

1.2 The Automatic Mail Reply System

This thesis details the creation of a proof-of-concept Automatic Mail Reply (AMR) system that works in two parts, background setup and recurring automatic reply suggestions. First the user inputs all of his or her past email interactions into AMR by importing an archive of their email to the AMR system. The system then sets up a database with all of the messages that have been sent and received by the user. These messages are stored along with significant information about the emails based on the content found in the messages. The text of the email is saved as individual keywords and, using statistical modeling, topics are extracted from these keywords.

Statistical modeling, or *topic modeling*, is a form of machine learning, expanded upon in the related work section below, which finds underlying structure in unlabeled data and then attempts to make predictions based on that structure. Different forms of statistical modeling are used in AMR, and they are explained in much more detail in Chapter 2.

After the system is set up and running, it waits until a new message is received. When the user receives a new email message, the message is sent to AMR, which then searches the archive for a previous email that matches the new query email. This old message is then provided for them to use in creating a response to the new query email. If the matching message contains both the original message and the reply, the user can either send the original response or modify it before sending. Each time the user gets a new email they are presented with a past message that has been matched to the new query. This reduces the amount of time needed to respond to messages by eliminating the need to write similar responses again.

The goal of this research was to build the proof-of-concept Auto Mail Reply system and to ensure it works by testing it with multiple email datasets and new email queries. Once built, different topic modeling approaches were compared to find which works best in retrieving relevant messages from an archive of email based on a new query email. In the future, a full product could be developed based on AMR, using the best topic modeling approach. This product could be used by anyone with an email account to make his or her email responses easier to write.

1.3 Related Work

Although there has been little work on generating responses for emails, there has been considerable work on reducing the number of emails that a user has to read and deal with. Most of this work centers on looking at previous email interactions and categorizing new email as something that the user will probably respond to. One popular example of this was Google's Priority Inbox [3]. Priority Inbox took the user's past email and determined which messages the user was likely to actually care about and want to read. It marked these emails with a priority tag to make them stand out. This was done through machine learning [4]. Google has since created a newer way of filtering email and an entirely new email client named Inbox [5] focused on only showing relevant email and information to the user.

Research done by Ayodele [6] used past email interactions to attempt to predict which emails would need a reply from the recipient and which could be ignored. Ayodele used vector space modeling and term weighting to make these predictions. These ideas are explained further in Chapter 2 and are key concepts used in the approach of this thesis.

A paper by Dredze et al. [7] explained how machine learning could be used to help with the tasks of summary keyword generation. Dredze used two topic modeling approaches, discussed in more detail below, to attempt to summarize an email into keywords. This research suggested that using keywords to summarize an email was superior to using the entire message body. Later in this thesis when comparing emails to find their similarity, they are broken into keywords instead of using the raw text of the email based on the outcomes of this research.

In another paper [8], Dredze, along with Google's Gmail team, compares different approaches to suggesting viewing filters for sets of emails. A viewing filter is a tool used in many modern email applications that sorts received email into folders or labels based on a predefined setting. After comparing multiple approaches, machine learning is found to be the best way to categorize emails to give better results when searching through an inbox. Dredze's work is similar to this thesis in that it also incorporates the user's previous archive of email as the main dataset for the system. However, this thesis uses this learning approach to predict responses instead of filtering email.

Most recommendation systems, such as Google's Priority Inbox [3] mentioned above, rely on feedback from the user to create the recommendation. Having the user complete some action and then learning from this action, as in this approach, is called supervised learning. As an example, in Google's Priority Inbox the user can mark an email as important or not important. The system then uses this feedback to improve the model. Although supervised learning could be helpful once AMR is built, the initial goal is to make a system that will utilize the data already present in the user's archive of past interactions in order for matches to be found, rather than waiting until enough feedback has been supplied to build a model.

Supervised learning is one of three broad categories of machine learning. The other two categories are reinforcement learning and unsupervised learning.

In supervised learning the computer is given training data with specific inputs and desired outputs for it to learn how to classify new data. Once the computer is taught how to classify information, it is given the dataset to make predictions on. Reinforcement learning uses rewards to guide the system into predicting the correct output. When the correct output is selected, the system is rewarded which encourages the correct output again. With unsupervised learning the computer is given no training data and instead looks for structure in the dataset by recognizing patterns.

There has been work on creating a similarity metric for a user based only on their previous data without asking for additional feedback. Minkov et al. [9], using unsupervised learning, developed a recommendation system for suggesting new academic talks to users based on their previous interests. The system took previous data and attempted to make recommendations about what to do based on patterns found in the data. Another name for this searching is data mining [10]. The proof-of-concept AMR system works in a similar way, mining the email archive for information. The reply that is suggested to the user is based only on the information found in the user's past email interactions. The goal of AMR is to mine the person's archive of email to make this suggestion. Since the desired output is unknown and there is no reward system, unsupervised learning is the appropriate type of machine learning to be use.

Drezde [7] references two topic modeling approaches, Latent Semantic Analysis (LSA) [11] and Latent Dirichlet Allocation (LDA) [12]. These will be discussed in more detail in Chapter 2. Here, it is important to note that these two approaches underlie much of the machine learning research regarding email. Due to the amount of work that suggests these are the best approaches to use [6, 7, 8, 9, 11, 12], LSA and LDA are the primary topic modeling ideas explored in this thesis.

1.4 Outline

The remainder of this thesis will include an initial explanation of topic modeling and other definitions that are important for understanding the research. The next chapter discusses the overall approach taken for the proof-of-concept and comparison of the topic models. Chapter 4 explains the implementation of this approach. In Chapter 5, the results of the research are discussed. Chapter 6 presents conclusions and discusses possible future work, including turning AMR into a full product based on this research.

Chapter 2

TOPIC MODELING CONCEPTS

Topic modeling, or statistical modeling, is a method of unsupervised machine learning, which analyzes unlabeled data to uncover hidden, or latent, patterns. In this research the unlabeled data is a large body of text made up of emails. The patterns found in the model are in the form of *latent concepts*, or themes, that tie multiple words together. These latent concepts will later be used to find the similarity between documents.

Three topic modeling approaches are used in this thesis tf-idf, LSA, and LDA. These three approaches, along with other concepts necessary to understand them, are detailed below. It is important to note that the topics in this section are described using the terms “document” and “collection”; later each email will be treated as an individual document and the dataset, or email archive, as a collection.

2.1 Bag-of-Words

The simplest way to look at a body of text is to break it down into the individual words that exist in the text. This is called the *bag-of-words* model and does not include any extra complexity. The rest of the models discussed in this chapter include some other processing to get more context out of the text, however most start with this step.

2.2 Term Frequency (tf)

Term frequency [13] is a way of assessing how closely a document matches another document by counting the number of times a word (i.e., term) is used in each document and comparing the counts. The *count* is the number of times that term occurs in the document. For example, suppose there are three documents A, B, and C where A and B contain the word “cat” 8 and 10 times respectively, while C only contains the word once. A and B would give a higher similarity rating than A and C or B and C due to the count matching more closely.

The relevance of the document to another does not increase linearly with the frequency of a word in the document. Continuing the example from above, if a fourth document D contained the word “cat” 40 times, this document is not necessarily four times more likely to be similar to A than B is with its 10 “cat” instances. To correct for this, a logarithmic scale is used rather than the specific number of times a word occurs. The sum of the logs of the weights for the frequency of each word is the value representing how closely the document matches the query. The term frequency weight $T_{t,d}$ is calculated using Equation 2.1 below as $1 +$ the log of the count $c_{t,d}$ of the term t in the document d. To prevent the value being negative, if $1 +$ the log of the count is negative, T would be zero. The sum of these values for each term in the two comparison documents is the score for their similarity.

$$T_{t,d} = \max(0, 1 + \log_{10}(\, c_{t,d}))$$

Equation 2.1.

2.3 Inverse Document Frequency (idf)

When comparing two documents in a collection of many documents, one factor to consider is how common a word is in the collection. If there is a collection of 1,000 documents and the word ”computer” occurs in 400 of them while the word ”vegetable” occurs in 800 of them, “computer” should have a higher weight in determining the similarity of two documents. The “computer” term is more significant because it is less common. The weight of this significance is called the *inverse document frequency* [13].

Note that this is the frequency of the term in the *entire collection* of documents, which is not the same as the term frequency from the previous section. To calculate the inverse document frequency weight, each document is checked to see if the term exists in the document or not. There are only two outcomes: the document contains the term or it does not, regardless of the number of times the term occurs in the document. When a new document is added to the collection, the inverse document frequency is updated with the terms in the new document.

Since the desired outcome is to find the words that occur the least, the inverse of the frequency is used. The logarithm is used, just as in the term frequency weight equation above, because the effect is not linear. In Equation 2.2 below the inverse document frequency weight $\iota_{t,C}$ of the term t in the collection C is the number of documents in the collection $|C|$ divided by the number of documents d that contain the term.

$$\iota_{t,C} = \log_{10}(\frac{|C|}{|\{d|d \in C \text{ and } t \in d\}|})$$

Equation 2.2.

2.4 Term Frequency-Inverse Document Frequency (tf-idf)

Term frequency-inverse document frequency (tf-idf) is the combination of the previous two concepts to create a better metric for determining the similarity of a query and a document in a collection. The tf-idf weight $\tau_{t,d,C}$ of each term t in document d of collection C is found in Equation 2.3 by multiplying the term frequency weight $T_{t,d}$ of the term and document by the inverse document frequency weight $\iota_{t,C}$ of the term. These weights are summed for all of the terms in the query q and the document d as in Equation 2.4, producing a score $\Psi_{q,d,C}$ for how closely a document matches a query. This is one of the best ways to determine similarity matching of documents [13] and is the first topic modeling approach compared in this research.

$$\tau_{t,d,C} = T_{t,d} \times \iota_{t,C}$$

Equation 2.3.

$$\Psi_{q,d,C} = \sum\nolimits_{t \in q \cap d} \tau_{t,d,C}$$

Equation 2.4.

2.5 Vector Space Model

Another approach to comparing document similarity is using a *vector space model*. In this model, first created by Salton et al.[13], a multidimensional space is created where the terms in the documents are the many axes and the documents themselves are vectors in this space. Figure 2.1 below shows an example of this vector space model. The query vector and two document vectors are shown with terms as the axes. In this case the terms are “cat”, “dog”, and “bird” are terms that exist in a collection while the query, d1, and d2 vectors are representations of documents in the collection that contain these three terms.

The document vectors only exist between terms that exist in the documents themselves, making the space extremely sparse. In Figure 2.1, the query vector would lie flat along the “cat”, “dog” plane if it did not contain the term “bird”. These vectors are created for all of the documents as well as the query. After the vectors are created, the lengths must be normalized to unit vectors by dividing the vectors by their respective lengths. Now these vectors can be compared.

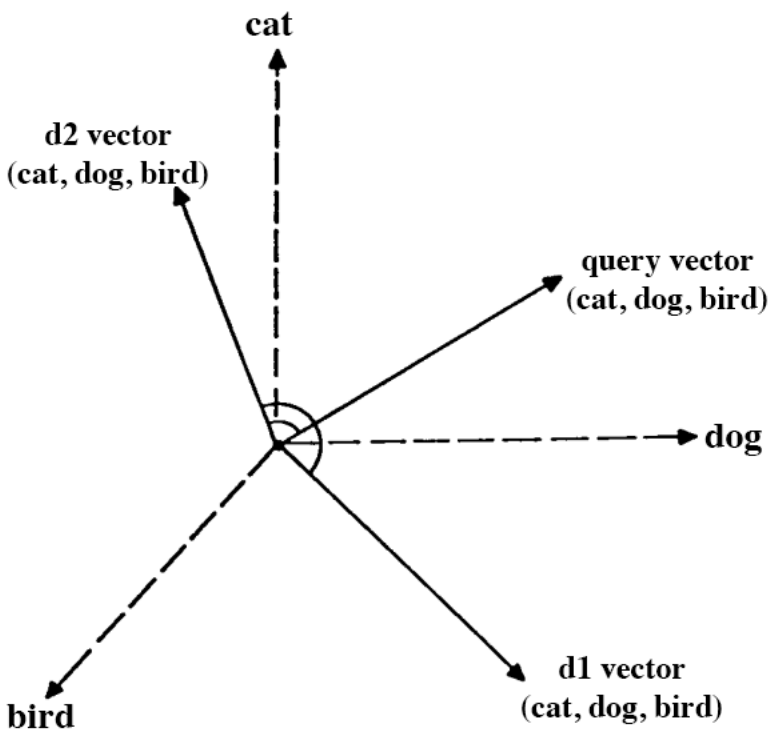


Figure 2.1. Representation of the vector space model with three document vectors between three term axes

The document vector that is the closest match to the query vector will be the most similar document to the query in the collection. Rather than looking at the endpoints of the vectors, the angles of the vectors are compared. This is a more accurate way to compare the vectors as the length is removed from the equation. The smaller the angle is between two document vectors, the more similar they will be. The cosine of the angles gives a way to compare them, resulting in a numerical value of similarity between two documents or a query and a document.

Equation 2.5 and 2.6 show how to find the cosine similarity S of the query vector $\vec{q}$ and document vectors $\vec{d}$ in this vector space V. The angle between $\vec{q}$ $\vec{d}$

is represented by α, q_i is the tf-idf weight of the ith term in the query and $d_i$is the tf-idf weight of the ith term in the document. These vectors are length normalized by dividing the vectors by their lengths to create unit vectors at the angle α. Equation 2.5 includes the length normalization step whereas the following Equation 2.6 shows the equation once the terms are length-normalized.

$$S(\vec{q}, \vec{d} \in V) = cos(\alpha) = \vec{q} \cdot \vec{d} = \frac{\vec{q}}{|q|} \cdot \frac{\vec{d}}{|d|} = \frac{\sum_{i=1}^{V} q_i d_i}{\sqrt{\sum_{i=1}^{V} q_i^2} \sqrt{\sum_{i=1}^{V} d_i^2}}$$

Equation 2.5.

$$S(\vec{q}, \vec{d} \in V) = \sum_{i=1}^{V} q_i d_i$$

Equation 2.6.

2.6 Principal Component Analysis

Principal component analysis [14] is a process used to evaluate data by transforming it from one set of axes to a new set in order to highlight a specific feature. As principal component analysis applies to AMR, the transformation is based on the query vector and is a step leading to the second topic modeling approach in this thesis, Latent Semantic Analysis. As discussed in the vector space model above, the initial axes are the individual terms that exist in the collection of documents. The documents are initially represented as length-normalized vectors in this space as depicted in Figure 2.1 above. With principal component analysis the document vectors are transformed onto new axes made up of the query vector and the orthogonal match to the query vector. These axes are called the principal components and are used to find the distance from the document vectors to the query vector.

Figure 2.2 below shows an example in 2D space where the axes are represented by terms “cat” and “dog” in the collection; the query vector is also the new axis after the transformation. The blue points are the endpoints of the document vectors and the distance from the new axis $|d1 - d1_q|$ would be the variance from the query.

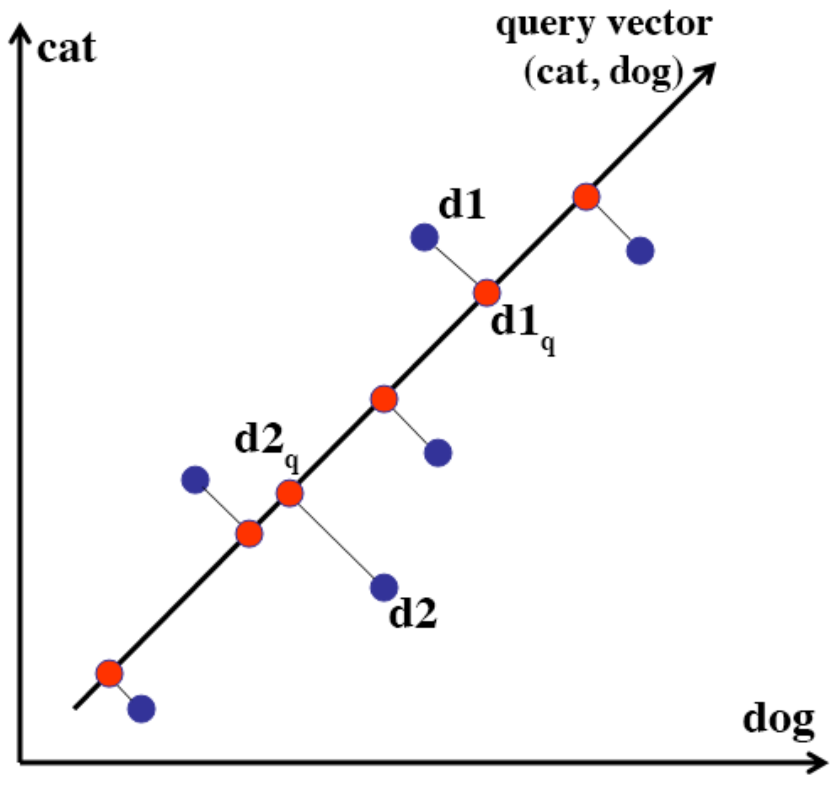


Figure 2.2. Principal Component Analysis 2D Example [15]

This is done in order to “flatten” the data as all of the document vectors can now be represented in terms of their variance from the query vector. This creates a way to filter out the least significant data points, as the vectors that are more closely related to the query will be closer to the new axes and the outliers will have a much larger variance. Principal component analysis works regardless of the complexity of the data or the number of dimensions, which is important, as the vector space model for the collection of documents will have many dimensions.

2.7 Singular Value Decomposition

Singular value decomposition [16] is a method of factorization used on a matrix of data in many statistical models. For the purposes of this thesis it is only important to understand how it is used as a step in the process of the second topic modeling approach used in AMR, Latent Semantic Analysis. As the vector space model and principal component analysis allow for comparisons based on the individual terms found through tf-idf weighting, singular value decomposition uses the same approach to find patterns that exist within the entire collection of documents. Instead of taking the principal components based on the tf-idf values in just the query document, it uses relative occurrences of terms to highlight topics across the entire collection. These topics are relationships that exist between words as they are frequently used together. A matrix is created with the tf-idf weights of the terms in the topics from the entire collection. Singular value decomposition uses this matrix to highlight the similarities of the terms in these topics.

2.8 Latent Semantic Indexing / Analysis (LSI / LSA)

In the paper introducing *Latent Semantic Analysis*, Deerwester explains the idea of LSA quite simply, *“We take a large matrix of term-document association data and construct a ‘semantic’ space wherein terms and documents that are closely associated are placed near one another.”* [11] The process uses the weight information that comes from tf-idf analysis, builds a vector space model of the documents and then uses principal component analysis and singular value decomposition to group the individual terms in the collection of documents into concepts. These concepts are then used to make a comparison of a query to the documents that produces results that would not be possible by simply looking at the individual terms.

As an example suppose the term “mouse” is frequently used with the terms “keyboard”, “computer”, “monitor”, “speakers” and “desktop” across many documents in a collection. The singular value decomposition in latent semantic analysis will take the frequency of the terms in proximity to each other and group these words together into a concept that can then be used when doing a similarity comparison. So if a new query document has the sentence ”I need a new keyboard and mouse for my desktop”, LSA will produce a higher similarity matching score to documents that contain any of the terms that exist inside of the concepts. An interesting result of comparing topics instead of individual terms is that a term that was not in the query could still show up in the similarity match due to the term’s proximity to the topic in the query. LSA is the second topic modeling approach compared in this thesis.

2.9 Bayesian Network

A *Bayesian network* [17] is a model of a system represented as a graph with nodes for pieces of the system. Each node in the graph has states and a probability distribution for those states. The nodes each have probabilistic outcomes based on the states of their predecessor nodes’ values. If a node in one part of the network is changed, the successor nodes will change their probability distributions leading to decision making at each successor node. In order for this to work correctly the graph cannot contain any cycles.

2.10 Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA) is a generative probabilistic topic modeling approach. The topics in the model come from probabilities generated by data. Created by Blei et al [12], LDA is similar to latent semantic analysis in that it also finds and then compares concepts in the documents of a collection. In fact, LDA is referred to as the Bayesian version of LSA [12].

The process for LDA starts off by creating a Bayesian network with a “Dirichlet” probability distribution based on an initial parameter. The Dirichlet probability distribution determines the number of concepts that the documents will be sorted into. The nodes in the Bayesian network will cluster the terms into a finite number of concepts.

Each document is represented by an array of concept groups along with the percentages of how well the document fits each of these concept groups. The similarity of documents to each other or to a query is found based on these groupings. Once the words are all placed into the concept groups based on the initial probability distribution, the documents are iterated over, refining the underlying topics through changes in the probability distribution. As the underlying topics are changed, this in turn changes the probability of terms matching topics and the categorization of the documents change.

Suppose that a collection of documents is compared with LDA using three topics. Each document in the collection would then be represented by a percentage of how much that document matches each of the topics. For example document A might be represented by being made up of 40% topic 1, 35% topic 2 and 25% topic 3. These percentages can then be used to compare a query to the documents and find the document that matches the percentages of the query most closely. If document B is made up of 35% topic 1, 35% topic 2 and 30% topic 3, the topical makeup of the two documents are similar due to their percentages. Each of these three topics are, in turn, made up of smaller topics modeled through the Bayesian probability distribution.

Of the three topic modeling approaches compared in this thesis, LDA is expected to perform the best similarity matching between documents and queries because it improves on the other two approaches by continuously refining the model.

Chapter 3

DESIGNING AUTOMATIC MAIL REPLY (AMR)

3.1 Overview

This chapter covers the step-by-step approach used for the proof-of-concept AMR system. This includes how to get the user's emails, how to parse the emails, how a corpus and model is created, how the similarity matching is computed, and how the new response is sent to the user.

3.2 Data

In order to automatically create a response based on previous messages, an archive of the user's previous email is needed. This archive of email is used to build a text corpus. This corpus acts as the raw dataset to use when making the model. The user will download this email archive and give it to the system. The system, in turn, will parse this archive and split it into individual pieces that will later be searched. This process may take a considerable amount of time because it is very likely that the user will have thousands of emails to parse. In fact, a large archive of emails is preferred, as having more data will allow the system to build a better model. The creation of the corpus text only needs to be done once before being saved and recalled later when creating the model. When a query is compared against the model, its terms are also added to the corpus text.

3.3 Reading Email and Parsing Important Information

When the email archive is obtained, it is parsed into keywords that are then included in the model. Each of the individual emails is split up into these keywords. A vector space model is created for the corpus based on the frequency of the keywords across the emails.

Since there are many common words in the English language, splitting words also includes removing common words in the dataset. These common words, called “stop words”, such as “a”, “as”, “if”, and “the”, are removed from the keyword list of each email. Lists of common English stop words are available [18], but an additional list was created to add in email-specific terms such as “original message”, “reply”, and “forward”.

The header information also helps find relevant matches. This data, including the subject, sender, and recipient, is saved and added to the keywords. This will also be valuable information to give to the user once a matching email is found because it could impact how the response is written. For example, if a user has a previous email sent to their friend that matches the new query email from their boss, the user could use this information to choose the appropriate tone for the new response email.

It is important that the original document is also saved separately without modification or removing words. Although only the keywords are necessary to perform the search and matching, it will be helpful to give the user the full body of the email so they can choose which parts to use in the new reply.

3.4 Creating a Model

After the keywords are found and set up for each document, a dictionary is created containing all of the keywords along with how frequently they are used. The model is then created with the combination of the corpus text and this dictionary. There are different topic models that are built with the corpus and dictionary. The topic models that are used in AMR are tf-idf, LSA and LDA, as described in Chapter 2. These are compared to see which performs the best with the given data and provides the most relevant results. The model comparison, along with comparing two Python tools, GenSim and SciKit, are discussed in Chapter 5. For the purposes of the proof-of-concept for AMR, it does not matter which approach is used, only that a model is created so that query emails can be compared against it.

3.5 Obtaining The Query Email

The next step is to have an incoming email sent to the system so that it can perform its search against the corpus. AMR checks the user's inbox and, when a new email is received, takes the new message to compare it against the corpus.

3.6 Comparing Against The Model

After the model has been created and the query email is in the AMR system, the check for similarity is used to determine what previous email has the highest probability of being useful to the user. First the information from the query is added to the model. The entire model does not need to be rebuilt because the model only needs to add or update the terms that exist in the new query. After the model is updated with the query information, the comparison can finally be done.

The result of this comparison, explained in depth in Chapter 5, is the top five matching emails ranked according to the percentage of how well they match the query. Along with these ranked results, a threshold percentage is used to ensure that only the most closely matched messages are sent to the user.

3.7 Reconstructing a Message

After using the keywords of the email to match the query with a previous response, this response is recalled to help build a new response. The actual body of the message is used instead of just the keywords, as the user would prefer coherent sentences rather than a random grouping of keywords.

Since the subject of building coherent sentences based solely on keywords could take multiple thesis projects, the proof-of-concept AMR system sends the user the body of the closest matching email so that the user can copy or delete the parts that they do and do not want. This may include only the original matched message or a reply as well as the original message depending on the structure of the matched email.

The purpose of this step is to give the user the best information possible so that they can create their response to the query email. This may be the entire response, allowing them to simply copy and paste. However, the AMR system can also be useful as a reminder about the user's previous responses or topics in a similar interaction. The user can use information such as to whom the matched email was sent, when it was sent, or the body of the message to construct their new response.

3.8 Sending a Response to the User

The response sent to the user includes the subject of the original query message so that it is obvious what message it refers to. In addition to the original body of the matching email, the body of the response includes the header information from the match along with the match percentage so the user knows how well it matches the query. Once this new message is sent to the user, they can use the information to construct their reply. By providing the user with this automatically generated content to go along with the email that they receive, they will be able to create their reply more quickly than they could have without this information.

In order to be useful the system should provide the sample response faster than the user could read and produce a response separately.

Chapter 4

IMPLEMENTING AMR

4.1 Datasets

4.1.1 First Two Datasets (Author’s Work and Private Email Archives)

In order to test the approach, datasets in the form of email archives were obtained and formatted for use in the creation of the corpus text, as described in section 3.4, before being turned into a model. The first two datasets come from the author’s own archive of emails. One is the author’s private email archive (5,249 messages) and the other is the author’s archive from working in information technology as a technology director and network administrator for a school district (17,725 messages). Due to privacy issues, these two datasets are not publicly available, and the results of this research will not show any specific emails from them.

Two of the most popular email clients, Google’s Gmail and Apple Mail, allow any user to create an archive of their email and export it. The format of this export, denoted as .mbox [19], is an industry standard. Another popular email client, Microsoft Outlook, allow exporting to an email archive that can easily be converted to the standard .mbox format. In the case of these first two datasets, the export was simply done from Gmail using Google’s Takeout service.

The two archives in the .mbox format needed to be broken down into the individual emails so that they could be searched to find important data. This is the very first step in the completed AMR system. A Python program was created for this process. This program asks the user for the location of the .mbox file and splits the archive into the individual emails and stores them as plain text files. This program supports extracting the messages from one single .mbox file or a group of many .mbox files if the user would like to use multiple archives. This allows anyone to use his or her own email archive in the AMR system.

4.1.2 Third Dataset (Secretary Hillary Clinton's Released Emails)

The third dataset of emails was chosen based on the current interest in the content of U.S. Secretary Hillary Clinton's emails, which were released to the public [20] under the Freedom of Information Act (FOIA) [21]. Since this content is freely available for anyone to download and look at, it provided an easy-to-obtain dataset that is unique. The content of the emails provide an interesting dataset to use to test similarity matching for different topic models.

Although the content of Secretary Clinton's emails (7,386 messages) would prove to be a useful dataset, the emails were not stored in the same simple .mbox file as in the initial datasets. Thus more work was required to get the data into a form that could be used for this research. This was completed in two steps: first by creating a tool to pull all of the emails from the US Department of State FOIA website and then formatting the emails to be used for the project.

Since there is not a simple way to download a large number of emails from the FOIA website, a Python program was written to retrieve the files individually from the server that they were stored on. A short script by McGill [22] was found that described the process of getting a JSON file with a list of all of the documents that are returned from searching the FOIA database. With this information, the FOIA database was searched to create a list of Secretary Clinton’s publicly released emails. Once this list was obtained, it was possible to use the Python tool “urllib2” to download the email documents.

This program to download the released emails will also work for any other search of the FOIA site that returns publicly available documents. In addition to the work done for the purpose of this paper’s research, this program will be publicly available on the author’s GitHub page [23] for anyone to use to easily download other publicly-available documents from the FOIA website.

Unfortunately, all of the emails downloaded from the FOIA site were in the form of PDFs. This format is easy for a human to read but is not optimized for insertion into the dictionary and corpus model used in the AMR system. The process of converting the data into a form that fits this research started with the conversion of the PDF documents to plain text files using Adobe Acrobat Pro’s [24] batch convert function. Once the individual emails were converted to plain text documents, they were adjusted to match the same structure found in a standard .mbox archive as in the previous two datasets.

In order for documents such as these to be released by the US State Department, the content had to be declassified first. In some cases, this declassification included removing some parts of the header information of the emails, such as the sender and/or recipient. All of the documents included a note saying that they were declassified by the U.S. Department of State. Another Python program was created to remove this note and move the header information to the top so that the messages would match those found in a .mbox archive. For messages with missing sender/recipient information, the sections in the header were left blank rather than including a replacement as this might have skewed results. This program also removed excess whitespace and added a line at the top forcing the use of UTF-8 encoding for the characters in the email. With this process completed, the dataset could be used in the AMR system just as if it came from a regular .mbox export.

4.1.3 Fourth Dataset (20 Newsgroups Dataset)

In order to have a benchmark to compare to the other datasets, it is necessary to have a dataset that others have used in the past for other email related processing projects. The 20 Newsgroups Dataset [25] provides this benchmark. It is available in many formats and includes 18,882 email messages on a variety of different subjects. These messages include the sender and subject header information and are already formatted correctly for inclusion in the AMR system. Since this data was already available in plain text and formatted as email messages, the data simply needed to be copied and imported into the system. This was done in the same way as the first two datasets, excluding extracting the files from an archive, since they were added as individual documents instead of a .mbox archive.

4.2 Email Aggregation (Python Email Utility)

Once the datasets are obtained and cleaned up, the next step for the AMR system is to go through the individual emails and extract the important pieces.

The AMR system includes a Message class to hold all the information for a specific email message. Python's built-in Email library is used to fill out this information. Each email is made into a separate instance of the Message class that contains variables for the file location, to address, from address, subject, body, and tokens. Most of these variables are straightforward, however the tokens of the email require more explanation.

4.3 Getting Keywords (NLTK)

To create the keywords for each email, the individual words in the body are turned into tokens using the Natural Language Toolkit Python library. The Natural Language Toolkit (NLTK) [18] is an open source platform for use with Python that offers many built-in tools for working with natural language processing. With NLTK, the body of the email in the form of sentences is easily split into tokens for each word. When the body of text is converted by NLTK, the punctuation is also removed from the tokens.

As mentioned in Section 3.3, English stop words are stripped out in the process of creating the Message tokens. NLTK contains a set of English stop words, but email specific words were removed as well. There was also a set of dataset-specific keywords that were removed such as the author's name from the first two datasets. The subject, sender, and recipient information is included with the body as tokens of a specific email to add additional context. In the case of forwarded and quoted replies, the full original text was saved to be given to the user at the end of the similarity matching process.

If a message contains multiple parts, such as an original message and a reply, AMR attempts to separate these sections. The lines in the email are parsed looking for common ways that replies are quoted by email applications. This includes beginning a section with “Original message:”, or “On Tuesday [Sender] wrote…”, or a section in which each line begins with the ”>” character. For the purposes of the proof-of-concept AMR system, these multiple parts are tokenized and included just as the main body is. However, future work could allow for matching specific pieces of an email to help narrow down the relevant information to give back to the user.

The last part of the email that is removed in the Message class is the signature people often leave at the end of their emails. Signatures can be included in many different forms depending on the email application used so the removal of the signature is tailored to the most popular email clients, Gmail, Apple Mail, and Outlook.

4.4 Creating Keyword Text and Index

After all of the email messages have been parsed and tokenized, the next step in the AMR system is to combine the individual email tokens into one keyword text document. This keyword text is the first form of the corpus that will be used later in the statistical modeling sections below.

The AMR system asks the user what they would like to call this corpus and then goes through all of the messages and saves the tokens of each email as a line in the corpus text. For the proof-of-concept system, an index file is created, along with the corpus text, with the location of the email file represented by the tokens at each line of the corpus text. This is done so that each instance of the Message class does not need to be saved in memory. Later, when a match is found, the Message class is recreated from the file location found in the index. If a full product were to be built after this thesis work, this information can be stored in a database table for improved speed and data storage.

4.5 Turning the Keyword Text Corpus into a Model

As discussed in previous chapters, there are three different topic modeling approaches used in this thesis, tf-idf, LSA, and LDA. In order for AMR to turn the corpus of tokens into a statistical model using these approaches, two different Python tools are used: “GenSim: Topic Modeling for Humans”, created by Řehůřek [26], and “Scikit-learn: Machine Learning in Python”, created by Pedregosa et al. [27]. For the remainder of this thesis these tools will be referred to as GenSim and SciKit respectively. Both of these tools are free to use and have been designed to work with statistical models.

GenSim was created specifically to work with text corpora while SciKit has a large number of uses, one of which is topic modeling with text corpora. The next two subsections will explain how these two tools were used in AMR to create the topic models that are then used in the search for similar email messages. In Chapter 5 the two models will be compared, along with the three topic models, to see which combination gives the best results for the datasets in this thesis.

4.5.1 GenSim Models (tf-idf / LSA / LDA)

In the AMR system there is a class for each of the two tools, GenSim and SciKit. These classes are used to create the topic models based on the corpus. The approach using GenSim starts by iterating through the keyword text corpus and adding each message's individual tokens to a dictionary. GenSim's implementation of the dictionary is identical to a regular Python dictionary, which allows for the use of Python's built-in functions. The next step is turning this dictionary into the bag-of-words counts of the tokens in the entire dictionary. With these counts, AMR is ready to create the model. To do this, one of the topic modeling approaches must be selected.

With tf-idf, a new corpus is created by counting the bag-of-words totals, term frequency, across all of the documents and multiplying by the inverse document frequency as described in Chapter 2. This creates a sparse vector model of the corpus where each of the documents has its own tf-idf weight.

The LSA model is created in a similar way as the tf-idf model in GenSim. After the tf-idf weights are found, principal component analysis and singular value decomposition are used to look for a number of latent topics in the dataset. The number of topics to extract from the data is specified in the AMR program.

If the number of topics is set too low, 10 for example, and the number of terms is large, 100,000 for example, the number of terms that fit in each concept is going to be very large, leading to the concepts being too broad and not very useful. On the other hand, having too many concepts, 1,000 for example, will make too many very specific concepts that will not be shared by many documents. The number of topic concepts for the LSA models used in this research was chosen to be 300, as research by Landauer et al. [28] found this number of topics to be the maximum number needed for finding the best results with LSA.

The LDA topic model requires more steps and a few more variables specified in the AMR program. As explained in Chapter 2, the LDA model finds a number of latent topics in the data and then iterates through the data multiple times, refining the topics in order to better represent the messages in relation to each other. As with LSA above, LDA also requires setting the number of latent concepts to extract. AMR also sets this number to 300 for consistency across models and tools.

The GenSim LDA model is also pre-set with two other variables, “chunk size” and “passes”. Both of these settings determine when the concepts will be updated while iterating through the data. The chunk size variable sets how many documents the model will categorize before stopping to update the concepts. For example, if the number of documents is 10,000 and the chunk size is 2,000, the model will stop every 2,000 documents, five times, to update the concepts. The pass number is the number of times the entire dataset will be iterated through, with an update to the concepts each time the end of the dataset is reached.

The chunk size for AMR’s LDA models is set at 2,000, as this number was the default for both GenSim and SciKit and, according to the Python tools [25, 26], this number is appropriate for the size of the datasets. Like the chunk size, the number of passes has an effect on how well refined the concepts are. The goal is to iterate through the data enough times that the concepts match the model and are no longer changing when they are updated. The Python tools suggested that two passes would be enough to solidify the concepts, but since the model only needs to be created once, five passes are done to ensure the best concepts are created for the model. In order to compare the different models as well as the different tools, the same numbers are selected for these values in the LSA and LDA models in both GenSim and SciKit.

Both the dictionary and the models are saved after being created with AMR's GenSim class so that all of this processing only needs to be completed once per dataset. When a query is done, the models are updated with the query's information, but the model does not have to be completely rebuilt. In this way, multiple queries can be compared against the model without having to reconstruct it each time a similarity check needs to be done.

4.5.2 SciKit Models (tf-idf / LSA / LDA)

Just as with the GenSim tool, AMR uses a separate class to build the topic models using SciKit. As in the first step in creating the models in GenSim, the SciKit class starts by iterating through the keyword text corpus extracting each message's individual tokens. Instead of a dictionary of these tokens, they are stored in a two-dimensional array $n \times m$, where the row n is the number of each document and the column m contains the corresponding document's tokens.

After the array is created, this data is passed into what SciKit calls a “vectorizer”. To create the first model, a tf-idf vectorizer is used which turns each of the elements of the array into a vector corresponding to the tf-idf weight of the terms in the documents. This is the sparse vector matrix that has been discussed multiple times previously in this thesis. The vectorizer transforms the data and fits it to the tf-idf model to be used later for comparison.

To create the LSA model in SciKit, another transformation of the data needs to be done. Starting with the already created and vectorized tf-idf model, singular value decomposition is applied with a set number of latent topics to be extracted. As mentioned in Chapter 2, the vectors must be normalized after being transformed in order to match the features that are extracted. GenSim takes care of this step while building the models, but SciKit requires normalizing the vectors as a separate step. Again, for consistency across models and tools, the number of latent topics to be found is set to be 300.

Where the LSA model in SciKit could use the already computed tf-idf weights as a starting point, the LDA model in SciKit uses a separate “count vectorizer”. This count vectorizer starts with the count of the term frequencies as with tf-idf, but also uses the initial vectorization to set the chunk size for how often to update the latent features in the model. The model is transformed and normalized during each iteration in order to find the best fit for the topics in the vector space.

The LDA model in SciKit has settings for the number of latent topics and number of iterations over the entire dataset, which are set to 300 and five respectively. Just as with the GenSim models, these numbers were selected after testing different values, and the same values are selected across tools and models in the AMR system so that the results can be compared. The models only need to be created once per dataset and can then be used to compare query documents without needing to be recreated.

4.6 Getting Query Email From Inbox

With the topic models created for the corpus, the AMR system is finished with the preparatory phase and is ready to be used for its main purpose: to find an email from the corpus matching a new query email. In order to make this comparison, first the query email needs to be retrieved.

For the proof-of-concept AMR system, the retrieval of the query email from the user's inbox is done with the email protocol IMAP and the Python library imaplib [29]. IMAP is a protocol for allowing services to access an email account and retrieve messages. When AMR is started it first asks the user for their email login information and attempts to connect to the user's inbox. The login information is stored securely only while the program is running and is deleted when exiting the program.

After successfully accessing the user's inbox, AMR checks for a new message every 30 seconds. If a new message is found, the email is opened and saved to a text file just as all of the other emails from the user's archive are stored. AMR then creates a new Message class instance of the query email and extracts the tokens as mentioned in a previous section.

In a future full product built with this AMR system, the program would run on a server and could stay logged in to the user’s account permanently. It would constantly check for new messages and wait patiently until a new one arrives. The new query message would also be stored in a database instead of as an individual file.

4.7 Comparing a Query to a Model

Now that the query email has been saved and its tokens extracted, AMR can finally compare these tokens to one of the topic models created previously. In the current AMR system, one model is used at a time. However, it is possible for the user to select the model to be used. First the query email’s tokens are added to the corpus. This is done so that like terms and concepts can be matched from the query to the previous archive. This also adds the query information to the model so that it can be used with future queries. This step is completed in the Python classes for the two tools.

In GenSim the new query document is indexed into the model. This indexing essentially determines the query’s place in the topic model. The similarity of the query to another document in the model is checked and a list is created of each of the documents in the model along with the percentage of how well the document matches the query. This percentage is the cosine similarity, as discussed in Section 2.5, multiplied by 100.

The documents that match the closest have values close to 100% such as 99% or 91% and documents that are the farthest from matching are given negative values down to 0%. A value of 100% should be given to the newly added tokens, as they will match the query perfectly since they are the same. This list is then sorted in descending order, ignoring the twin of the query that was just created in the corpus. The result is the top five matching emails ranked according to the percentage of how well they match the query.

SciKit similarly adds the new query document to the model and indexes it so that the keywords or topics that exist in the model can be searched for in the query. The query tokens are also turned into a vector representation to better match the tf-idf, LSA, and LDA models. Finally the cosine similarity is computed between the query and the model. Just like with the GenSim comparison, a list of the documents is created along with their individual corresponding percentage of how similar they are to the query.

In both SciKit and GenSim, the comparison results are formatted in the same way so that it does not matter which tool is used. The output from the comparison is a list of the top five document names along with their similarity percentage. In the AMR system these document names are actually keys to find where the email text file is stored by looking at the corresponding corpus’ index file that was created along with the keyword text in a previous section.

4.8 Creating a Reply

Now that there is a set of five emails that match the query document, helpful information can be given to the user. For the proof-of-concept AMR system, the top matching email of these five is selected and the original body of the email is extracted and then sent to the user. In the case that the match is a standard email, the original body will contain the keywords and/or topics that the model found to match the query. If the match is a message with a reply, the original body is used because the reply will be in the top section of the email with the quoted question in the bottom “replied-to” section.

It is important to note that there is a threshold percentage that can be set for the user by the AMR system. If none of the matching documents have a similarity percentage that exceeds this threshold, the system will not send a response. This is to ensure that the user is not sent random emails that have nothing to do with the query just because no other messages seem to fit. This threshold is initially set to 50%, but could be lowered to get more experimental results, or heightened to remove unhelpful responses.

Along with the original body of the matching email, some additional information is included at the top of the reply. This includes the percentage indicating how closely the model predicted this reply matches the query as well as the original reply’s header information including the subject, sender, recipient, and date. This response email is given the subject name “AMR Response: “ along with the query subject so that it is easy for the user to find what message the response is supposed to match.

4.9 Sending Response to the User

Now that the new response has been created, the final step is to send this response back to the user. AMR uses the Python library smtplib [30] to send this response to the user. The user can then use this information to construct their response to the original query email by either copying the entire message or selecting pieces of the message. Even if the user does not use parts of the automatically created response, it is possible that by seeing old messages the user will recall some information that they can then use to create their reply. Once a response is sent to the user, the AMR system goes back into its waiting mode, checking every 30 seconds to see if a new message is received and the comparison process runs again with the new query email.

Chapter 5

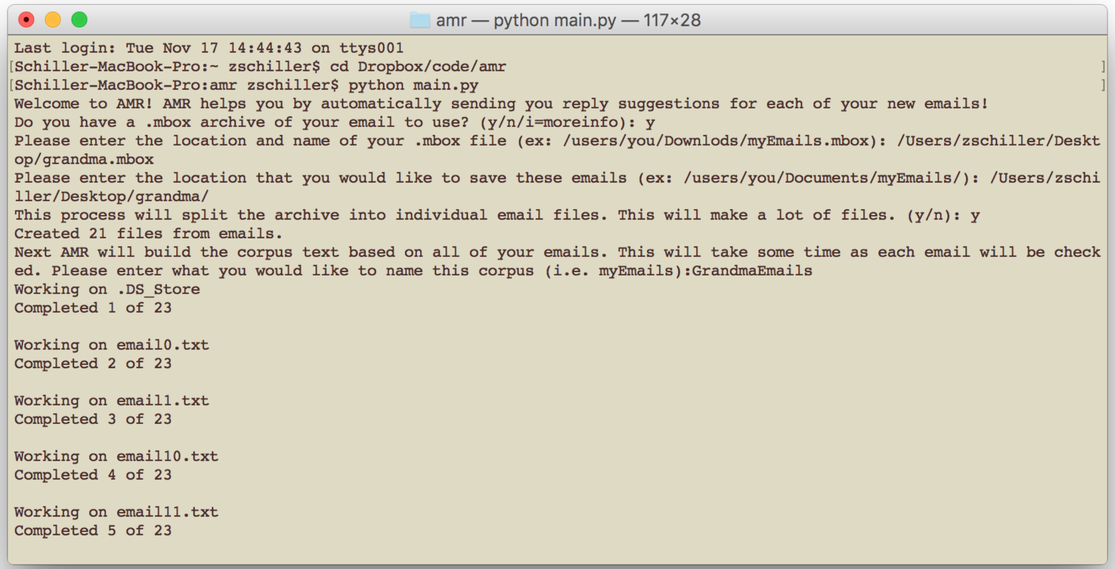
RESULTS AND ANALYSIS

5.1 Results of Building and Testing AMR

The AMR proof-of-concept system was built as designed in Chapter 3 and is functioning correctly as demonstrated in Figures 5.1 and 5.2, with examples in Figures 5.3, 5.4, 5.5, and 5.6 below. The system runs in the background and continuously checks for new emails. When a new email is received, an AMR response is created and sent to the user's inbox in less than a minute. The AMR system, in its current form, can run from a command line, import a user's .mbox archive and run locally on a computer. The only dependencies are the Python language and the extra tools discussed in the implementation chapter, NLTK, SciKit and GenSim.

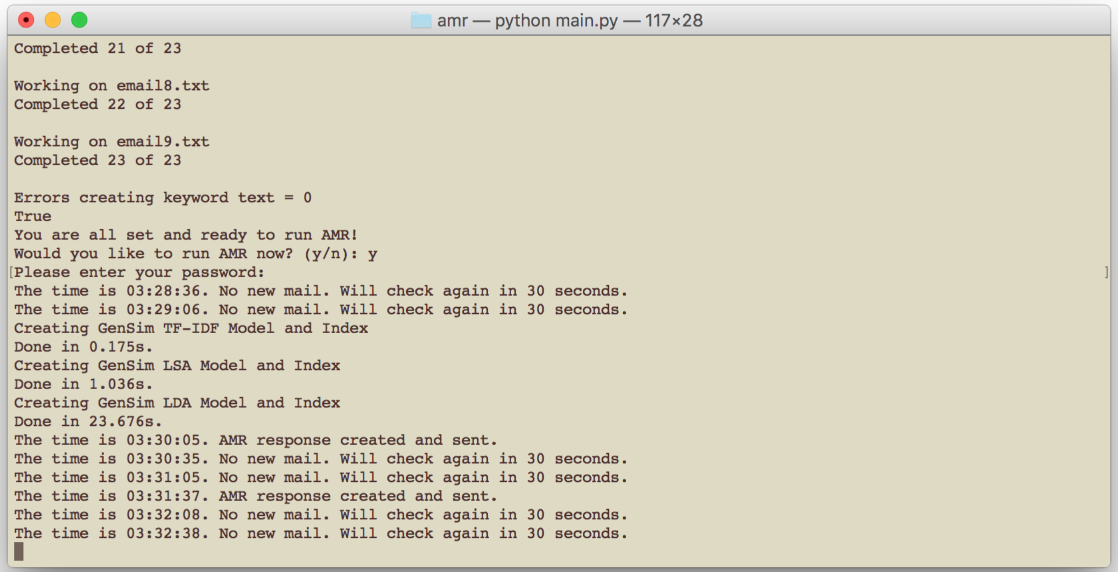
Figures 5.1 and 5.2 below show an example of the setup process for AMR. The user imports their .mbox email archive and the program builds the corpus and models for them. Once these steps are completed, the user is prompted to start the email processing side of AMR and login to their email account with their username and password. After this is done, AMR finds a new matching email, then creates and sends an email response. Next, the system waits before creating and sending a new response after another new email comes in.

In the final version of AMR the Python tool and model to use are set in the code of the program rather than allowing the user to make the choice. This is in order to remove an added confusing step for the user. However, these settings



```
Last login: Tue Nov 17 14:44:43 on ttys001
Schiller-MacBook-Pro:~ zschiller$ cd Dropbox/code/amr
Schiller-MacBook-Pro:amr zschiller$ python main.py
Welcome to AMR! AMR helps you by automatically sending you reply suggestions for each of your new emails!
Do you have a .mbox archive of your email to use? (y/n/i=moreinfo): y
Please enter the location and name of your .mbox file (ex: /users/you/Downlods/myEmails.mbox): /Users/zschiller/Deskt
op/grandma.mbox
Please enter the location that you would like to save these emails (ex: /users/you/Documents/myEmails/): /Users/zschi
ller/Desktop/grandma/
This process will split the archive into individual email files. This will make a lot of files. (y/n): y
Created 21 files from emails.
Next AMR will build the corpus text based on all of your emails. This will take some time as each email will be check
ed. Please enter what you would like to name this corpus (i.e. myEmails):GrandmaEmails
Working on .DS_Store
Completed 1 of 23

Working on email0.txt
Completed 2 of 23

Working on email1.txt
Completed 3 of 23

Working on email10.txt
Completed 4 of 23

Working on email11.txt
Completed 5 of 23
```

Figure 5.1. AMR Setup Example 1



```
Completed 21 of 23

Working on email8.txt
Completed 22 of 23

Working on email9.txt
Completed 23 of 23

Errors creating keyword text = 0
True
You are all set and ready to run AMR!
Would you like to run AMR now? (y/n): y
Please enter your password:
The time is 03:28:36. No new mail. Will check again in 30 seconds.
The time is 03:29:06. No new mail. Will check again in 30 seconds.
Creating GenSim TF-IDF Model and Index
Done in 0.175s.
Creating GenSim LSA Model and Index
Done in 1.036s.
Creating GenSim LDA Model and Index
Done in 23.676s.
The time is 03:30:05. AMR response created and sent.
The time is 03:30:35. No new mail. Will check again in 30 seconds.
The time is 03:31:05. No new mail. Will check again in 30 seconds.
The time is 03:31:37. AMR response created and sent.
The time is 03:32:08. No new mail. Will check again in 30 seconds.
The time is 03:32:38. No new mail. Will check again in 30 seconds.
```

Figure 5.2. AMR Setup Example 2

are easily changed, as AMR is setup to work with any combination of the tools and statistical modeling approaches discussed in this thesis.

Although AMR is creating responses and sending them to the user successfully, the usefulness of these responses is inconsistent. As an anecdotal test, a set of email questions were sent to AMR with the two public datasets to see if the responses created were useful or related to the questions. A response was deemed related if most of the keywords in the email were the same or similar to the query email. If there were only a few matching terms the response was marked as partially related. Finally, if the response could reasonably be used to aid the user in writing a response to the question, the response was marked as useful. Table 5.1 below shows the results of this short test. Some of these query emails were specifically created with keywords found in emails in the datasets to see how well the match would be found.

In another example, seen in Figure 5.4, the query email contains the keywords “party”, “drinks”, “Friday” and “cupcakes” and, although the matching email that AMR found only contains one of the words, “Friday”, the matching email contains the keywords “birthday”, “goodies” and “breakfast”. These terms are similar to the keywords in the query email so the topic model, SciKit’s LSA in this case, finds this a relatively good match at 55% and gives the information to the user. This is a good example of using the latent concepts in the corpus to help in finding a match for the query, even if the terms do not match. However this response is not particularly helpful for creating a response to the query email.

Query Subject	20 News Response Match %	20 News Response Related	20 News Response Useful	Sec. Clinton Response Match %	Sec. Clinton Response Related	Sec. Clinton Response Useful
Book	70.0	Partially	No	99.2	Yes	Yes
Friday	67.3	No	No	55.2	Yes	No
IP Address?	82.3	Partially	No	58.1	No	No
Groups in Gmail	50.3	Yes	Yes	73.4	No	No
Capital Questions	60.9	No	No	59.5	No	No
Recommendation	60.9	Partially	No	71.3	No	No
Afghanistan	71.5	No	No	62.9	Yes	No
Computer Speed	62.5	No	No	49.5	No	No
Movies	62.9	No	No	52.2	Partially	No

Table 5.1. Relevance and usefulness of the AMR System with the 20 Newsgroups and Secretary Clinton datasets

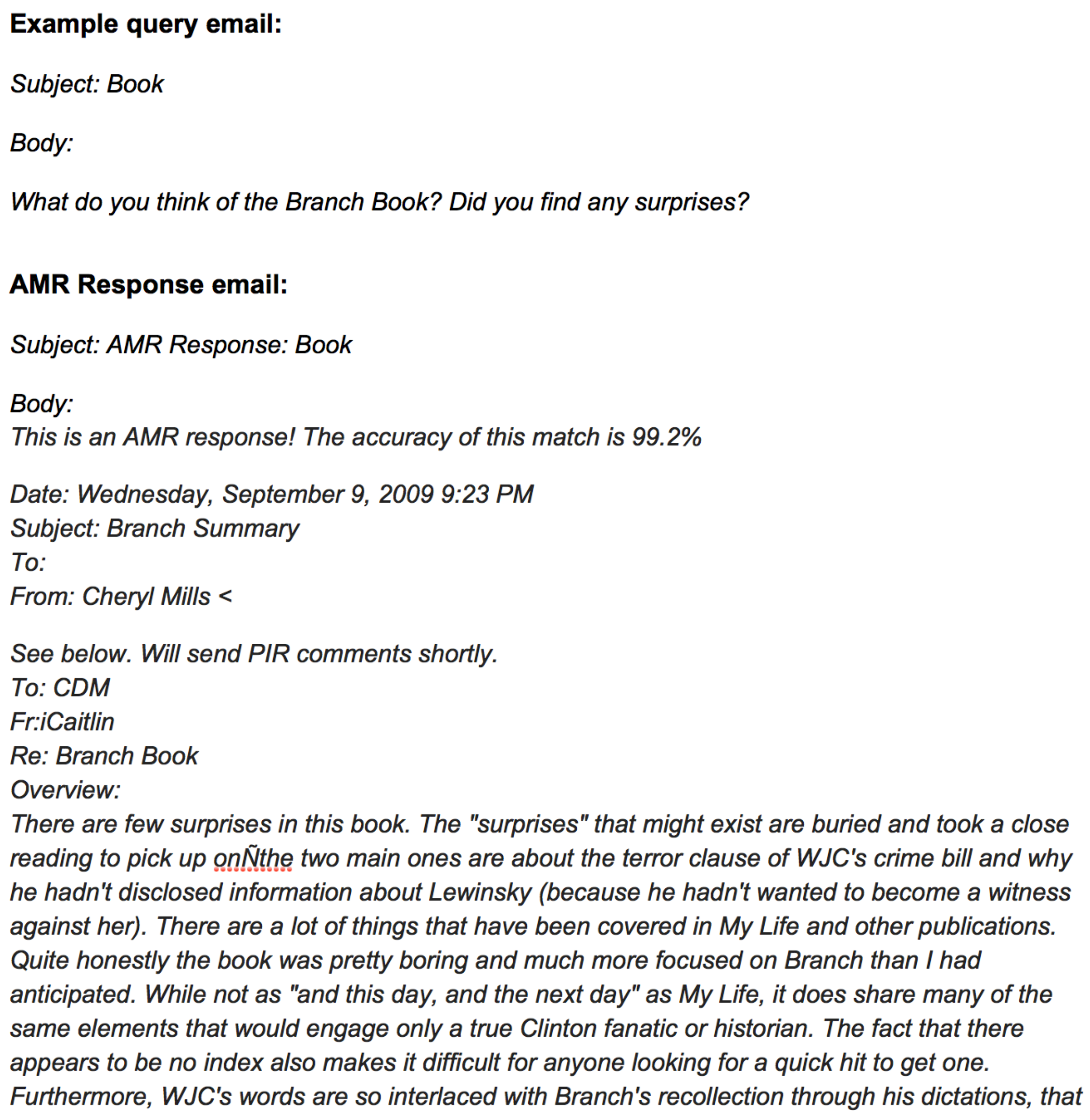
Figure 5.5 shows an example AMR response to a query asking how to make email groups in Gmail. The response does not mention Gmail, but contains instructions on how to set up a listserv for sending emails to groups of people. Although this is not exactly the response that is being searched for, it is information that would help in the reply to the query. Without having a dataset and set of queries that are specifically made for each other, this is a promising result.

As seen in Table 5.1, there are a few examples of AMR matching similar emails, but there are many examples of the system failing to produce related or useful results. Having a threshold for the response set to 50% alleviates some of this, but occasionally some nonsense email is brought up as a result of a query email. There are a few reasons this could happen which are discussed more thoroughly in the next section.

In the example in Figure 5.6, the query email contained some computer and networking related terms such as “IP Address” and “server”. However, the matching response AMR found does not have a lot to do with the query email despite the model, SciKit’s LSA, giving the match an 82% accuracy rating. This can happen when there simply is not a good match in the dataset, or the topics found by the model do not match the emails very well.

A positive feature demonstrated in Figure 5.6 is the removal of the signature before comparing the message against the corpus. The query email contained a signature with a quote. AMR correctly removed the signature before finding the tokens in the email. This is important, as removing signatures helps to ensure that the keywords that match are actually relevant to the information in the email.

Since the AMR system is working as intended, the remaining question is what tool and model produces the best matching results for email. The next section will discuss selecting and comparing the topic models and tools to find which works best with AMR.



Example query email:

Subject: Book

Body:

What do you think of the Branch Book? Did you find any surprises?

AMR Response email:

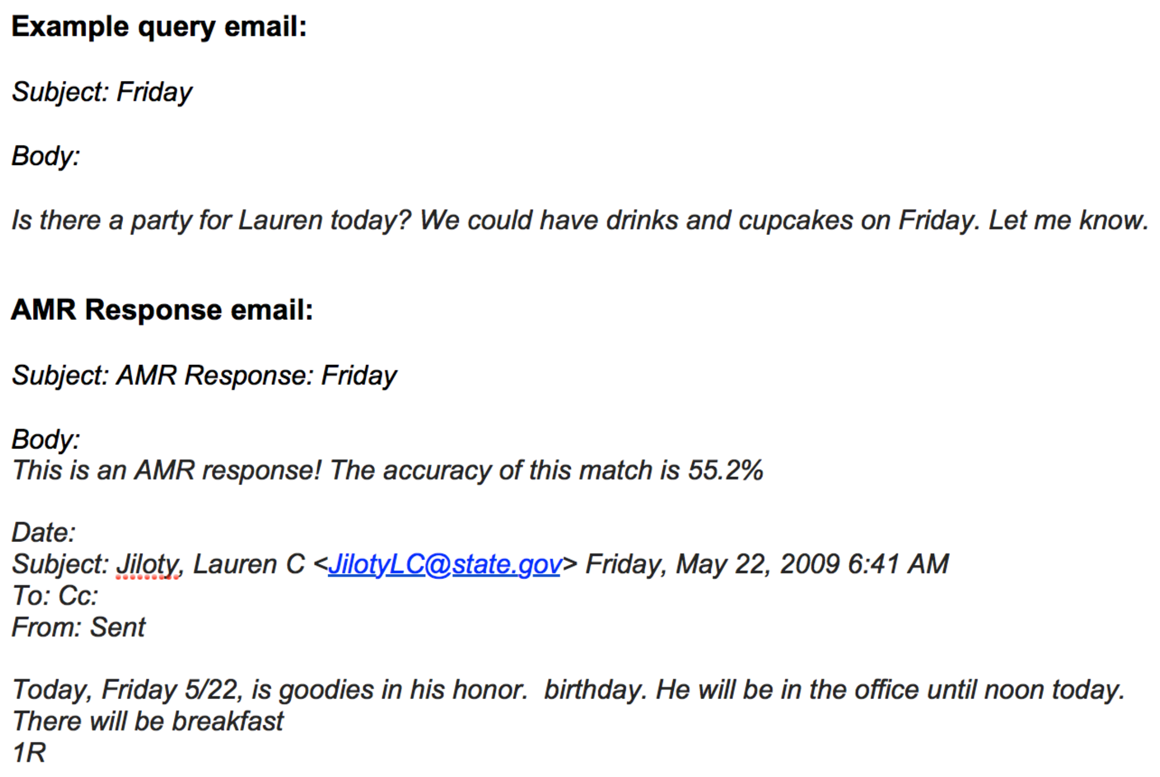
Subject: AMR Response: Book

Body:
This is an AMR response! The accuracy of this match is 99.2%

Date: Wednesday, September 9, 2009 9:23 PM
Subject: Branch Summary
To:
From: Cheryl Mills <

See below. Will send PIR comments shortly.
To: CDM
Fr:iCaitlin
Re: Branch Book
Overview:
There are few surprises in this book. The "surprises" that might exist are buried and took a close reading to pick up onÑthe two main ones are about the terror clause of WJC's crime bill and why he hadn't disclosed information about Lewinsky (because he hadn't wanted to become a witness against her). There are a lot of things that have been covered in My Life and other publications. Quite honestly the book was pretty boring and much more focused on Branch than I had anticipated. While not as "and this day, and the next day" as My Life, it does share many of the same elements that would engage only a true Clinton fanatic or historian. The fact that there appears to be no index also makes it difficult for anyone looking for a quick hit to get one. Furthermore, WJC's words are so interlaced with Branch's recollection through his dictations, that

Figure 5.3. Example AMR Response with Secretary Clinton Dataset, GenSim LDA Model



Example query email:

Subject: Friday

Body:

Is there a party for Lauren today? We could have drinks and cupcakes on Friday. Let me know.

AMR Response email:

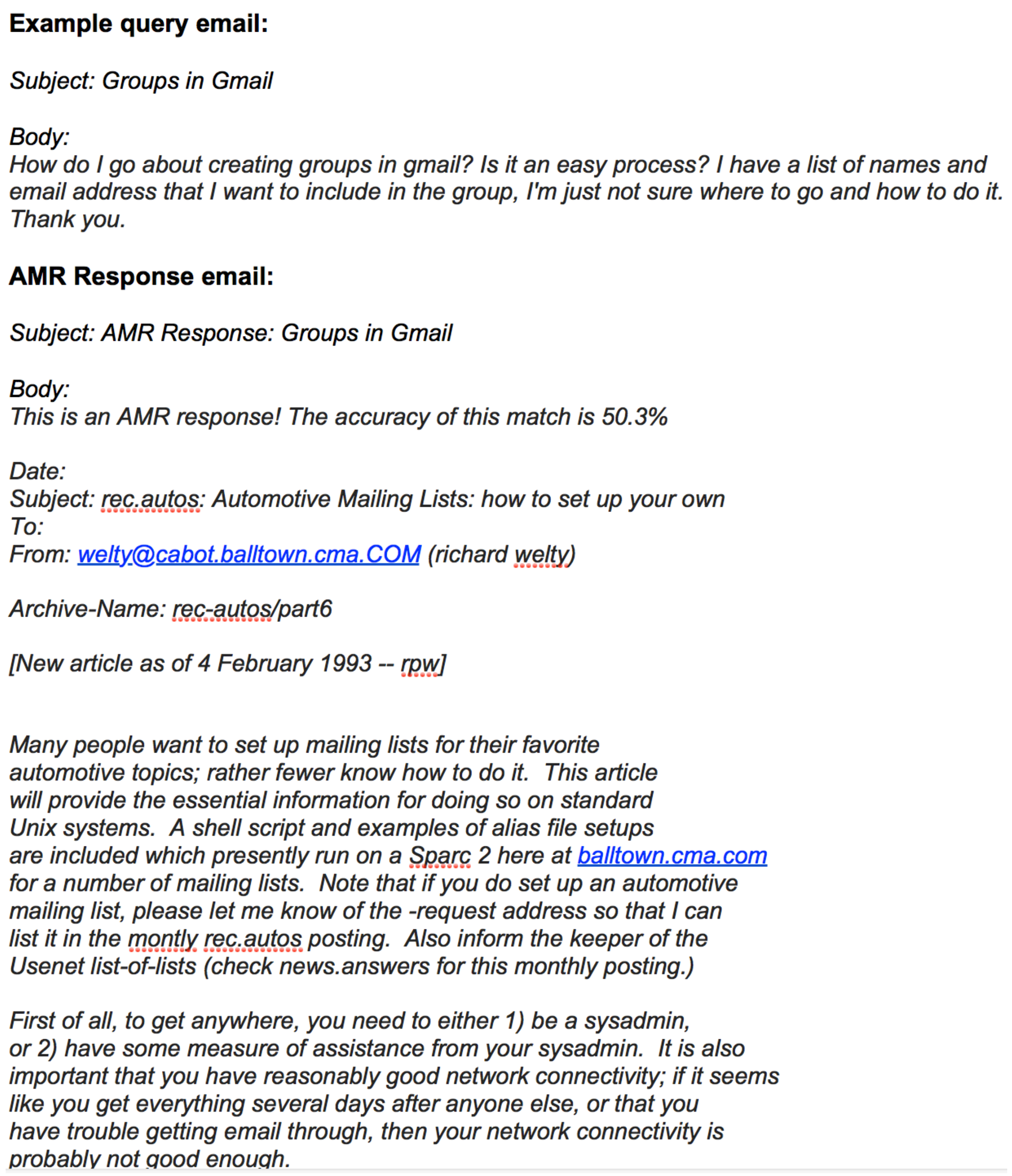
Subject: AMR Response: Friday

Body:
This is an AMR response! The accuracy of this match is 55.2%

Date:
Subject: Jiloty, Lauren C <JilotyLC@state.gov> Friday, May 22, 2009 6:41 AM
To: Cc:
From: Sent

Today, Friday 5/22, is goodies in his honor. birthday. He will be in the office until noon today.
There will be breakfast
1R

Figure 5.4. Example AMR Response with Secretary Clinton Dataset, SciKit LSA Model



Example query email:

Subject: Groups in Gmail

Body:
How do I go about creating groups in gmail? Is it an easy process? I have a list of names and email address that I want to include in the group, I'm just not sure where to go and how to do it. Thank you.

AMR Response email:

Subject: AMR Response: Groups in Gmail

Body:
This is an AMR response! The accuracy of this match is 50.3%

Date:
Subject: rec.autos: Automotive Mailing Lists: how to set up your own
To:
From: welty@cabot.balltown.cma.COM (richard welty)

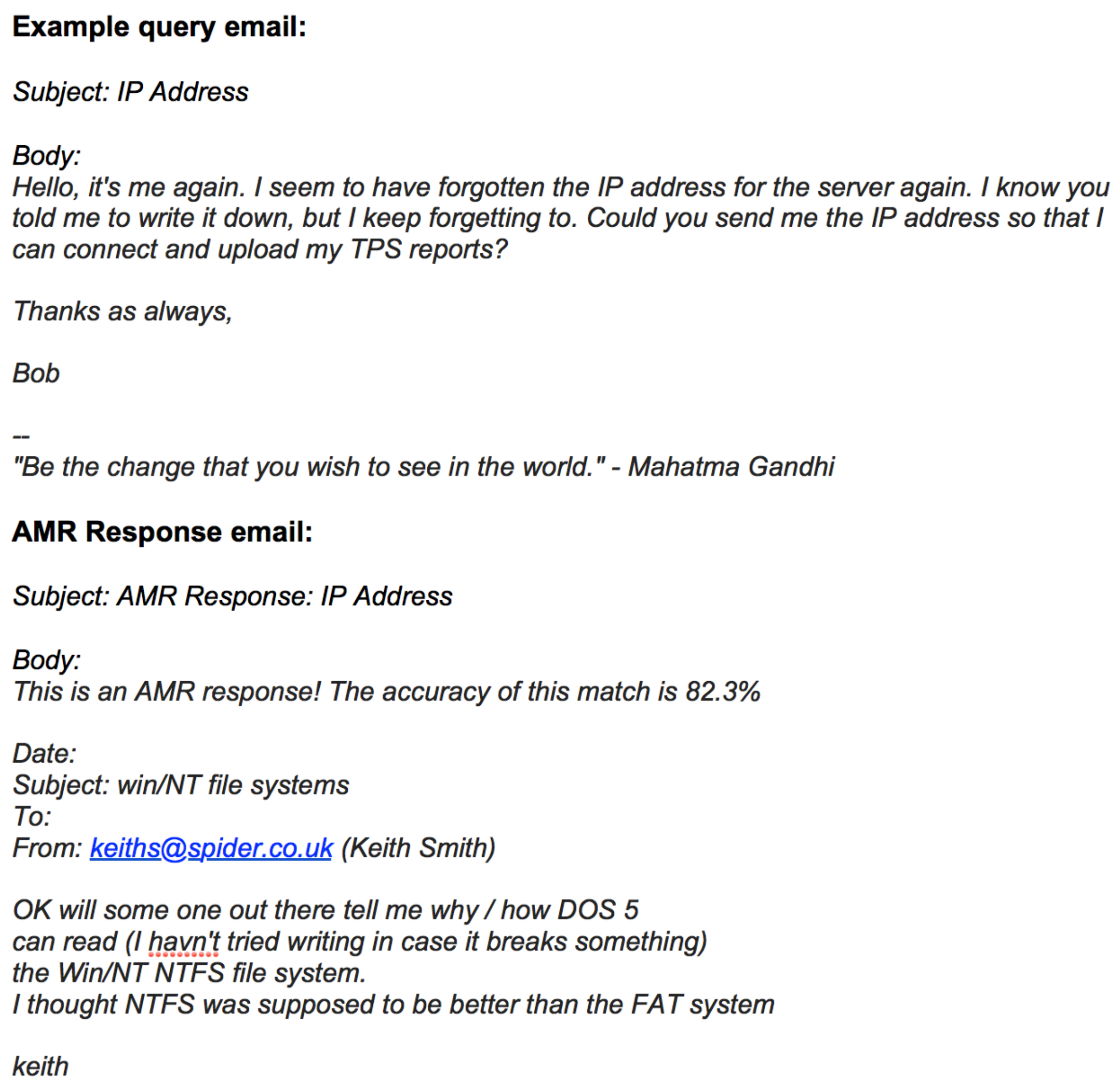
Archive-Name: rec-autos/part6

[New article as of 4 February 1993 -- rpw]

Many people want to set up mailing lists for their favorite automotive topics; rather fewer know how to do it. This article will provide the essential information for doing so on standard Unix systems. A shell script and examples of alias file setups are included which presently run on a Sparc 2 here at balltown.cma.com for a number of mailing lists. Note that if you do set up an automotive mailing list, please let me know of the -request address so that I can list it in the montly rec.autos posting. Also inform the keeper of the Usenet list-of-lists (check news.answers for this monthly posting.)

First of all, to get anywhere, you need to either 1) be a sysadmin, or 2) have some measure of assistance from your sysadmin. It is also important that you have reasonably good network connectivity; if it seems like you get everything several days after anyone else, or that you have trouble getting email through, then your network connectivity is probably not good enough.

Figure 5.5. Example AMR Response with 20 Newsgroups Dataset, SciKit LSA Model



Example query email:

Subject: IP Address

Body:
Hello, it's me again. I seem to have forgotten the IP address for the server again. I know you told me to write it down, but I keep forgetting to. Could you send me the IP address so that I can connect and upload my TPS reports?

Thanks as always,

Bob

--
"Be the change that you wish to see in the world." - Mahatma Gandhi

AMR Response email:

Subject: AMR Response: IP Address

Body:
This is an AMR response! The accuracy of this match is 82.3%

Date:
Subject: win/NT file systems
To:
From: keiths@spider.co.uk (Keith Smith)

OK will some one out there tell me why / how DOS 5
can read (I havn't tried writing in case it breaks something)
the Win/NT NTFS file system.
I thought NTFS was supposed to be better than the FAT system

keith

Figure 5.6. Example AMR Response with 20 Newsgroups Dataset, SciKit LDA Model

5.2 Comparing Topic Models

This subsection describes how the models and tools were compared, shows the comparison results, and discusses what is meant by one approach being “better” than another.

5.2.1 Predicted Outcome

Based on previous research and testing done by others [10, 11], the processes using LSA and LDA were expected to work better than the tf-idf approach. This is mainly due to the concept of grouping messages based on latent topics found in these more advanced methods. Further, it was expected that LDA would create better results than LSA because of the way LDA iterates over the documents multiple times to refine the concepts.

5.2.2 What is Meant by “Best” Method?

When comparing the three statistical modeling approaches as well as the two different tools it is important to understand what is actually being compared. For the purposes of this thesis, the “best” method would give the optimal email match based on the query email. However, optimal could mean different things. First, there is an assumption that there exists a relevant match to the query in the dataset. It is difficult to define the “best” match if there are no relevant matches in the dataset. Another problem with searching for the best method is that users can have different opinions about what makes a response match useful.

Understanding that these limiting factors exist, the remaining sections can now discuss how the comparison was completed.

5.2.3 Finding a Way to Assess the Methods

In order to determine which method worked the best there must be some criteria for determining the effectiveness of the output. The best approach would be to give these tools to many people and have them use the tools for a period of time. These testers could report back on how useful they found the AMR system in general as well as the different methods. Unfortunately, this was not possible due to time constraints, so an alternate approach was needed.

For the two public datasets, one way to compare the methods would be to see how the method's results compare to Google's indexing and ranking system. To do this, these two datasets were listed on a publicly accessible website and the indexes were given to Google for inclusion in their web-crawler. Once the sites were indexed, a simple Google search including the specific folder could be done. Unfortunately, Google only allows for searching with a small number of keywords compared to the many keywords used in the emails from the datasets. Although initial testing worked by searching with two or three terms, using all of the keywords from an email failed to produce results.

Another approach, although more arbitrary than percentages, would be to analyze a set of query emails and look at the results to determine if the results would be useful to someone in creating an answer to the original query. If the results included the exact answer this would be the most helpful. If the results contained part of the answer this would be the next best, followed by results that were similar to the question. Finally the lowest score would be given if the results did not seem to match the query email at all. Unfortunately, this approach would require individually going through and looking at every message to determine whether the method's match fit into one of these categories. Since this would require too much human involvement, this method was not used.

5.2.4 How Results Were Obtained

In an effort to find which approach works best on the four datasets a formula was created that could compare the approaches. The combination of the two tools and three models lead to six different methods to be compared: SciKit-tfidf, SciKit-LSA, SciKit-LDA, GenSim-tfidf, GenSim-LSA, and GenSim-LDA. The following paragraphs explain how these six models were tested on each of the four datasets.

To find a result for this comparison, an assumption had to be made about the effectiveness of the methods. The assumption is that the matching document selected by the majority of the methods is the best match possible for the query. That is to say, if a query is tested against a dataset with each of the six methods, the matching document that is given the highest matching percentage by most of the six methods is considered the best match for the query.

For example if a query is tested against the dataset and four of the methods suggest document A is the best match and two methods suggest document B, it is assumed that A is the correct top match of all the documents in the dataset because it was found to be the best in four of the six cases. It is important to understand that in making this assumption, the results below are more of a comparison of the consistency of the methods rather than their usefulness. With this assumption, it is possible to compare the methods to find how well the methods each pick a match that the majority agree is suitable for the query.

To create results to be used in the comparison, the six models were created for each dataset. Then each dataset was iterated over by the models, treating each individual document as the query against the rest of the dataset. This produced a set of the top five similarity matched documents for each of the six methods. These five documents, along with their match percentages were recorded for each document in each dataset. Table 5.2 below shows an example of the output from using one of the documents as a query against the dataset.

Note that the top match for every comparison would actually be the query document itself with a 100% matching percentage. This 100% match was used as a way to ensure the model was working correctly, but was removed in order to find relevant results.

Method	1st Match	1st Match %	2nd Match	2nd Match %	3rd Match	3rd Match %	4th Match	4th Match %	5th Match	5th Match %
SciKit -tfidf	A	34.95	C	34.46	D	32.38	B	31.79	E	31.74
SciKit -LSA	B	87.39	A	87.28	D	87.10	C	87.07	F	86.84
SciKit -LDA	A	34.95	C	34.46	B	32.38	E	31.79	G	31.74
GenSim-tfidf	A	31.66	B	31.12	C	29.62	D	28.91	E	27.37
GenSim-LSA	B	86.28	D	86.27	A	86.26	E	86.24	C	86.20
GenSim-LDA	C	98.62	B	98.62	D	98.62	H	95.66	F	93.54

Table 5.2. Example Method Comparison Results from Dataset

With these numerical results, a formula to compare them had to be found. Since the methods often found similar documents in the top five matches, the matched documents were ranked depending on how many times they were listed in the results of a specific query as well as the rank in that particular method's results.

A point system was used where a document matching the query would get five points if it was the first match, four for the second, three for the third, two for the fourth and one for being the fifth matching document. Table 5.3 below shows a demonstration of this point assignment continuing the example from Table 5.2. The points for each document are summed to create a score for that document.

Method	1st Match	2nd Match	3rd Match	4th Match	5th Match
SciKit-tfidf	A	C	D	B	E
SciKit-LSA	B	A	D	C	F
SciKit-LDA	A	C	B	E	G
GenSim-tfidf	A	B	C	D	E
GenSim-LSA	B	D	A	E	C
GenSim-LDA	C	B	D	H	F
Score for documents:	**{ A:22, B:23, C:19, 15, E:6, F:2, G:1, H:2 }**				

Table 5.3. Example assigning points given to matching documents of one query

In order to determine the accuracy of the model's choice for the best match to the query, this score is divided by the maximum number of points that a matching document can get. If a match were selected as the best for each of the six methods it would earn five points for each, giving 30 points. Dividing by 30 gives a proportional rating for how well a specific document matches the query based on all six models.

With this score for each matching document, the top rated match for each method was multiplied by the corresponding percentage the model gave to that match. Continuing the example in the previous tables, Table 5.4 shows this final match value for each method. This finally gives one matching rating for each method and each query document in the corpus. These final values were averaged across each dataset and the means were compared to find which tool and model produced the best results.

Method	1st Match	1st Match %	Match score	Final Score
SciKit-tfidf	A	34.95	(22/30)	0.256
SciKit-LSA	B	87.39	(23/30)	0.670
SciKit-LDA	A	34.95	(22/30)	0.256
GenSim-tfidf	A	31.66	(22/30)	0.232
GenSim-LSA	B	86.28	(23/30)	0.662
GenSim-LDA	C	98.62	(19/30)	0.953

Table 5.4. Example final scores given to methods for one query

5.2.5 Analysis of Results

With the resulting scores, there were two factors to compare: the Python tool used to create the models and the models themselves. For the evaluation of the Python tools, the results were separated into the SciKit models' scores and the GenSim models' scores. Similarly, for the evaluation of the models, the results were separated based on the model used: tf-idf, LSA, and LDA.

Before the averages were compared, a one-way analysis of variance (ANOVA) [31] was completed to determine whether the difference between the resulting scores was significant. The null hypothesis was that there is no difference between the results based on the tool or model used. Table 5.5 below shows the results of the ANOVA for the two factors. The results indicate significance ($p < 0.05$) for both comparison factors. Therefore, the null-hypothesis can be rejected.

	GenSim & SciKit ANOVA Test Statistic	p value	tf-idf, LSA & LDA ANOVA Test Statistic	p value
Personal Email Dataset	308.23	< 0.05	156.34	< 0.05
Work Email Dataset	889.73	< 0.05	123.40	< 0.05
Secretary Clinton Dataset	172.59	< 0.05	40.08	< 0.05
20 Newsgroups Dataset	2,004.23	< 0.05	603.92	< 0.05

Table 5.5. ANOVA results for Python tool comparisons and model comparisons

With the ANOVA completed, a t-test [32] comparison of means was run for each set of variables to ensure the differences in the means of the scores were statistically significant. This mean corresponds to how often the method picks the highest ranked matching document out of the list of matching documents found by all the methods in this research.

The results of the Python tool comparisons and corresponding t-tests are below in Table 5.6. For each of the datasets, the results show that the mean of the SciKit scores was greater than that of the GenSim scores. This indicates that SciKit performs better on the datasets than GenSim with statistical significance ($p < 0.05$). The difference in the means range from 0.015 for the author's work email dataset to 0.020 for both the author's personal email dataset and the 20 Newsgroups dataset.

	SciKit Mean (SD)	GenSim Mean (SD)	Test Statistic	p value
Personal Email Dataset	0.152 (0.083)	0.132 (0.094)	17.56	< 0.05
Work Email Dataset	0.137 (0.075)	0.122 (0.085)	29.83	< 0.05
Secretary Clinton Dataset	0.186 (0.105)	0.170 (0.115)	13.14	< 0.05
20 Newsgroups Dataset	0.103 (0.071)	0.083 (0.073)	44.77	< 0.05

Table 5.6. SciKit and GenSim comparisons with standard deviations and t-tests

The means of the three topic models were compared in sets of two in order to determine the significance of the results. First tf-idf and LSA were compared, then tf-idf and LDA, and finally LSA and LDA. The results of these comparisons, along with the corresponding t-tests are below in Tables 5.7, 5.8, and 5.9. In three of the four datasets, Latent Semantic Analysis produced the results with the highest score, indicating that LSA is the best model for the datasets. Only one dataset, the author's work email, had a higher mean for the tf-idf model as seen in Table 5.7. The comparisons with LSA were all statistically significant ($p < 0.05$).

For the comparisons between the tf-idf and LDA models, tf-idf consistently scored higher. The only comparison that did not produce a statistically significant result ($p = 0.62$) was the comparison of these models in the 20 Newsgroups dataset. For every other comparison, LDA scored the lowest among the topic models tested in this thesis.

	tf-idf Mean (SD)	LSA Mean (SD)	Test Statistic	p value
Personal Email Dataset	0.139 (0.081)	0.155 (0.093)	11.97	< 0.05
Work Email Dataset	0.133 (0.071)	0.131 (0.088)	4.79	< 0.05
Secretary Clinton Dataset	0.179 (0.106)	0.184 (0.110)	3.53	< 0.05
20 Newsgroups Dataset	0.087 (0.069)	0.104 (0.077)	29.87	< 0.05

Table 5.7. tf-idf and LSA model comparisons with standard deviations and t-tests

	tf-idf Mean (SD)	LDA Mean (SD)	Test Statistic	p value
Personal Email Dataset	0.139 (0.081)	0.132 (0.091)	5.66	< 0.05
Work Email Dataset	0.133 (0.071)	0.124 (0.082)	16.11	< 0.05
Secretary Clinton Dataset	0.179 (0.106)	0.171 (0.114)	5.42	< 0.05
20 Newsgroups Dataset	0.087 (0.069)	0.088 (0.070)	0.50	0.62

Table 5.8. tf-idf and LDA model comparisons with standard deviations and t-tests

	LSA Mean (SD)	LDA Mean (SD)	Test Statistic	p value
Personal Email Dataset	0.155 (0.093)	0.132 (0.091)	16.65	< 0.05
Work Email Dataset	0.131 (0.088)	0.124 (0.082)	10.06	< 0.05
Secretary Clinton Dataset	0.184 (0.110)	0.171 (0.114)	8.71	< 0.05
20 Newsgroups Dataset	0.104 (0.077)	0.088 (0.070)	29.32	< 0.05

Table 5.9. LSA and LDA model comparisons with standard deviations and t-tests

5.2.6 Best Topic Modeling Approach for Emails

Based on the results in Tables 5.6, 5.7, 5.8, and 5.9 above, the best topic modeling approach according to this scoring method is SciKit-LSA. It is important to remember the assumption made to get this result. The assumption was that the document considered to be the best match by the majority of the methods is indeed the best match for the query. If this assumption is valid, the results show that SciKit-LSA is the best approach to use for AMR.

5.3 What These Results Mean

There are two interesting points in these results. First is that the SciKit models consistently beat their GenSim model counterparts in all four datasets. This indicates that the SciKit tool is a better tool to use with the AMR system. However, there are two caveats to this conclusion. First is that the assumption made to find this data favors results that match the other methods and second that the difference in the average matching percentage is still not very large. Further testing would need to be completed in order to validate the results, but it is predicted that the SciKit tool will lead to better results for similarity matching over GenSim.

Another interesting point is that the LSA method beat the LDA approach. This result was surprising because the past research suggests the opposite. This could be due to the assumption made to do the comparison. It is possible that the datasets did not meet some requirement, either in terms of number of documents or content in those documents, to allow the iterative LDA approach to improve over the LSA model. Since the key difference between LSA and LDA is the constant improving of the concept groups, another possible reason could be that the similarity matching does not benefit from this improvement.

Anecdotally, when comparing a few responses by hand, simply looking at the results, LDA seemed to give more relevant results for the query. LDA sometimes recommends a match that is different than the match suggested by tf-idf and LSA, however this match may actually be more useful to a person. It is possible that the results of the comparison in this thesis are not consistent with the results that would come from having human testers. This would also explain why the expected result, LDA giving the best matches, is not reflected here.

As mentioned in Section 5.2.2, it is difficult to rate the true usefulness of this outcome. These results were obtained by comparing the methods to each other and not to an outside standard, since this idea has not been implemented publicly before. More research will need to be completed including tests with multiple users' inboxes.

Chapter 6

CONCLUSION

6.1 Completion of AMR

The main goal of this thesis, as stated in Section 1.2, was to build a working proof-of-concept Auto Mail Reply system. This goal was accomplished as was shown in Chapter 5. As an outline for a larger product, AMR was also a success, as it represents the core of what a full product would need to do. At the start of this research it was uncertain whether a system like this, once built, would be useful. The system now works and can be helpful in showing the user relevant information and replies based on query emails.

The AMR system is publicly available and can be downloaded and used by anyone with an email account. The only extra tools needed by the user are the GenSim and SciKit Python tools as well as the Python language itself. AMR will be open sourced and will be provided on the author's Github page[2] for anyone to use or add to.

In addition to using AMR in order to build a full email based product, the work in this thesis could be used in many different applications. The core of AMR is the processing of email, unsupervised learning based on topic models, and applying that knowledge to make a prediction. These pieces could be used as a toolkit for many different purposes. It could be used for searching emails, comparing other matching algorithms, or as a basis for a recommendation engine for a user's non-email documents.

[2] https://github.com/zacheryschiller/amr

As an example, AMR could be used to help with matching descriptions of one product on a website to a different description of the same product on a different site. There are many possibilities for continued work with AMR since it is freely available on the Internet.

6.2 Comparisons

While there are conclusions that can be drawn from the comparisons in this thesis, the overall best topic modeling approach and Python tool for this system is not yet known. As mentioned in Section 5.2.3, there are many ways that the methods could be tested and compared, and the approach in this research is just one of them. The results presented in Chapter 5 show the effectiveness of the models on the included datasets, but additional testing on different datasets and with different approaches would need to be completed before definitively concluding that one tool and model are the best for the AMR system.

With more time for this research, the next step would have been to have the system tested by many human users. They would each have a version of the AMR system that used the different methods and could provide feedback on the success or failure of these methods. Using a service like Amazon's Mechanical Turk [33], many different approaches could be tested on a public dataset. By having the system tested by many individuals with their own very different email archives, a more definitive conclusion could be made about which topic modeling approach is the best for AMR.

Although the best topic modeling approach may need more testing to determine, one result from the testing was that the SciKit tool was superior to the GenSim tool for this purpose. Consistently across all of the datasets, SciKit gave more accurate results as shown in Table 5.6. This result leads to the suggestion that future versions of AMR be built using the SciKit tool rather than the GenSim tool.

6.3 Usefulness

The AMR system works the best when the user gets many emails that are similar in their content. There are many professions where this would be useful. For example, AMR would be an ideal tool for positions in information technology or human resources. The author's work dataset produced much more useful responses than the author's personal dataset. One reason for is the repetitive questions and similar answer topics likely to be found in the email archives of an IT professional.

If the AMR system were used at a company or business where multiple people worked in the same position, the archives of multiple people could be combined to create a better dataset. This would also be beneficial for a team in which people come and go. When someone leaves a team, the answers that that person could provide are then lost to the team. It is possible that the person left their email account with the team, but the remaining team members may not have time to go back and look through a past colleague's email. By adding the person's email archive to the team's dataset, their old responses can be brought up if there is a new query that they had already answered.

Dataset sharing could be taken much further. For example, anyone using the AMR system could include his or her email archive in a general dataset to be used by everyone. There are obviously negative implications of this, as people's email could be given to others. However, instead of using the additional data as matches for query searches, the archives could instead be used as additional training data for the topic model to help in finding latent topical links between keywords.

Although the goal of this thesis is to make email more efficient, increasing the efficiency of something does not always make the situation better. It is possible that by cutting down the amount of time it takes to read and respond to email, society will get used to the efficiency and instead expect people to respond to more email. This is known as Jevon's paradox[3] [34]. However, if AMR were expanded upon, it may be possible to create a system that would respond automatically. This would completely remove the user from having to spend time rewriting messages.

6.4 Building a Full Product

In order to take the work on the AMR system and expand it into a fully functional product, first the system would need to be set up on a server rather than run locally on the user's computer. This would ensure that the system could run continuously and provide responses to the user as quickly as possible. The

[3] Thanks to George Markowsky for bringing this to my attention.

content could be stored in a database to improve searching and recall time as well as provide support for as many topic models as desired.

Another important addition would be to support OAuth or another form of authentication protocol linking the user's inbox to the AMR system. Having the user import his or her email as a .mbox file works and is not very difficult, but it is an unfamiliar process to someone who does not have experience with system tools. Additionally, it could be a small but important barrier keeping some from using the system. Syncing AMR with the user's email would also allow the models to be updated if the user changes their archive by deleting emails or re-categorizing them.

As mentioned previously, the current AMR system attempts to separate multipart messages into the original message and the reply. Further work could expand on this to include searching each piece of the message and providing the user with only the reply part of the message. Since some people will reply separately to an email, the thread of message interactions may need to be traced to find a specific response. In some situations, the user may prefer the entire message, but the future system would benefit from this option.

The next big improvement would come from the way that the automatically generated response is given to the user. In the proof-of-concept AMR system, the generated response is emailed to the user separately. The best solution to this problem would be to build the system into an add-on for the email service or as an extension for the Internet browser. In either of these cases, the automatically generated response could be shown to the user as an overlay of the original query email. This would make the system easier to use and increase the efficiency of the tool.

One way that the suggestion model could be improved is by incorporating feedback from the user into a supervised learning model as mentioned in the related work section of Chapter 1 in addition to the unsupervised learning that is already happening in AMR. First the model would be built and trained with the previous archive of email then, as the user is presented with multiple automatically generated responses, the system would remember which ones that the user actually picked and use this as part of its learning process. This supervised learning approach would refine the model as it is being used to continuously improve it.

The final addition that would be beneficial to the AMR system would be to use the top matching replies to actually build new sentences for the reply as mentioned in Section 3.7, instead of just giving the user the full raw replies as the system currently does. If the messages were annotated with information using the Semantic Web standard [35] this semantic knowledge could be used to help with the creation of these sentences. Additionally, there has been some research into generating text responses by learning from user input. This has been done most famously by Cleverbot.com [36] and also in a recent Google research paper [37]. In fact, combining the conversation model used in the paper by Google with the research in this thesis, a better version of the Automated Mail Reply system could be created that would be even more useful.

REFERENCES

[1] M. Chui. *The social economy: Unlocking value and productivity through social technologies*. 2010. [Online]. Available: http://www.mckinsey.com/insights/high_tech_telecoms_internet/the_social_economy

[2] The Radicati Group, Inc. *Email Statistics Report 2011-2015*. [Online]. Available: http://www.radicati.com/wp/wp-content/uploads/2011/05/Email-Statistics-Report-2011-2015-Executive-Summary.pdf

[3] Official Gmail Blog. *Email overload? Try Priority Inbox*. 2015. [Online]. Available: http://gmailblog.blogspot.com/2010/08/email-overload-try-priority-inbox.html.

[4] D. Aberdeen, O. Pacovsky, and A. Slater. *The Learning Behind Gmail Priority Inbox*. NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds. 2010.

[5] Official Google Blog. *An inbox that works for you*. 2015. [Online]. Available: http://googleblog.blogspot.com/2014/10/an-inbox-that-works-for-you.html.

[6] T. Ayeodele and Z. Shikun. *Applying Machine learning Algorithms for Email Management*. Pervasive Computing and Applications, 2008. ICPCA 2008. Third International Conference. pp 339-344. 2008.

[7] M. Dredze. *Intelligent Email: Aiding Users with AI*. PhD thesis, University of Pennsylvania. 2009.

[8] M. Dredze, B. Schilit, and P. Norvig. *Suggesting Email View Filters for Triage and Search*. [Online]. Available: https://www.cs.jhu.edu/~mdredze/publications/dredze_ijcai_09.pdf

[9] E. Minkov, B. Charrow, J. Ledlie, S. Tellar, and T. Jaakkola. *Collaborative Future Event Recommendation*. 2010. [Online]. Available: http://rvsn.csail.mit.edu/location/Minkov_collaborative_recommendation.pdf

[10] M. Kantardzic. *Data Mining: Concepts, Models, Methods and Algorithms*. John Wiley & Sons, Inc. 2002.

[11] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, and R. Harshman. *Indexing by latent semantic analysis*. Journal of the American Society of Information Science. 41(6). pp. 391–407. 1990.

[12] D. Blei, A. Ng, and M. Jordan. *Latent Dirichlet Allocation*. Journal of Machine Learning Research. 2003.

[13] G. Salton, A. Wong, and C. S. Yang. *A Vector Space Model for Automatic Indexing*. Communications of the ACM. vol. 18. nr. 11. pp. 613–620. 1975.

[14] K. Pearson. *On Lines and Planes of Closest Fit to Systems of Points in Space*. Philosophical Magazine Volume 2. pp. 559–572. 1901.

[15] B. Taskar. *Dimensionality Reduction and Principal Component Analysis*. University of Pennsylvania CIS 520: Machine Learning. [Online]. Available: http://learning.cis.upenn.edu/cis520_fall2009/index.php?n=Lectures.PCA

[16] T. Landauer, D. Laham, and P. Foltz. *Learning human-like knowledge by singular value decomposition: A progress report*. Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference. Vol. 10. MIT Press. 1998.

[17] J. Pearl. *Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning*. UCLA Technical Report CSD-850017. pp. 329–334. 1985.

[18] S. Bird, E. Loper and E. Klein. *Natural Language Processing with Python.* O'Reilly Media Inc. 2009.

[19] The Open Group - UNIX. *mbox*. Version 6 Unix. 1975. [Online]. Available: http://wwwlehre.dhbw-stuttgart.de/~helbig/os/v6/doc/I/mail.html

[20] A. Jaffe. CNN.com *First Round of Hillary Clinton State Department Emails Released* [Online]. Available: http://www.cnn.com/2015/05/22/politics/hillary-clinton-emails-release-benghazi/

[21] US Department of State. *Freedom of Information Act*. [Online]. Available: https://foia.state.gov/Default.aspx

[22] A. McGill. *Your Friendly Neighborhood Journalist*. 2015. [Online]. Available: http://andrewmcgill.me/hillary-clinton-benghazi-emails-scraping/

[23] Z. Schiller. *AMR*. 2015. [Online]. Available: https://github.com/zacheryschiller/amr

[24] Adobe Systems Incorporated. *Adobe Acrobat Pro*. [Online]. Available: https://acrobat.adobe.com/us/en/products/acrobat-pro.html

[25] *20 Newsgroups Dataset*. [Online]. Available: http://qwone.com/~jason/20Newsgroups/

[26] R. Řehůřek. *Gensim: Topic Modelling for Humans*. 2015. [Online]. Available: https://radimrehurek.com/gensim/

[27] Pedregosa et al. *Scikit-learn: Machine Learning in Python*. JMLR 12, pp. 2825-2830. 2011. [Online]. Available: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[28] T. Landauer, P. Foltz, and D. Laham. *Introduction to Latent Semantic Analysis*. Discourse Processes. 25. 259-284. 1998. [Online]. Available: http://lsa.colorado.edu/papers/dp1.LSAintro.pdf

[29] Python Software Foundation. *imaplib - IMAP4 Protocol Client*. 2015. [Online]. Available: https://docs.python.org/2/library/imaplib.html

[30] Python Software Foundation. *smtplib - SMTP Protocol Client*. 2015. [Online]. Available: https://docs.python.org/2/library/smtplib.html

[31] J. Neyman. *Statistical Problems in Agricultural Experimentation*. Supplement to the Journal of the Royal Statistical Society. Vol. 2. No. 2 pp. 107-180. 1935.

[32] Student (W. Gosset). *The Probable Error of a Mean*. Biometrika. Vol. 6. No. 1. pp. 1-25. 1908.

[33] Amazon.com, Inc. *Mechanical Turk*. 2015. [Online]. Available: https://www.mturk.com/mturk/welcome

[34] W. Jevons. *Of the economy of fuel [excerpt from The Coal Question]*. Organization & Environment 14(1). pp. 99-104.

[35] W3C. *Semantic Web.* [Online]. Available: http://www.w3.org/standards/semanticweb/

[36] R. Carpenter. *Cleverbot*. 2015. [Online]. Available: http://www.cleverbot.com/

[37] O. Vinyals and Q. Le. *A Neural Conversational Model*. Google. [Online]. Available: http://arxiv.org/pdf/1506.05869v3.pdf

[38] D. Jurafsky and C. Manning. *Ranked Retrieval-Stanford NLP*. [YouTube Video Series]. Available: https://www.youtube.com/playlist?list=PLt2REbl0sREOHKudF45g5cbhA4SHBHtyc

[39] Norsys Software Corp. *Introduction to Bayes Nets*. 2015. [Online]. Available: https://www.norsys.com/tutorials/netica/secA/tut_A1.htm

APPENDIX

Code

All of the code written for this thesis is publicly available and can be found on the author’s GitHub page: https://github.com/zacheryschiller/amr.

BIOGRAPHY OF THE AUTHOR

Zachery L. Schiller was born in Newburyport, Massachusetts on September 1st, 1988 where he lived for five years until moving to Bangor, Maine and then again to Calais, Maine at the age of 10. He grew up in Calais attending Robbinston Grade School and Calais High School where he graduated third in his class in 2006.

Zack attended the University of Maine in Orono, on a full scholarship and graduated in May 2011 with a B.S. in Physics minoring in Math and Astronomy. During his undergraduate career he got the opportunity to intern for two Summers at the Jet Propulsion Laboratory (JPL), a National Aeronautics and Space Administration (NASA) field center in Pasadena, California. While working at JPL he fell in love with computer science and returned to the University of Maine pursuing a Master of Science degree in Computer Science.

While working towards his Master of Science degree, Zack has worked full time as the Technology Director for the RSU 26 school district in Orono, Maine. At this position he has transformed the technology program at the district creating a more cohesive technological experience for the students and staff.

He is a candidate for the Master of Science degree in Computer Science from the University of Maine in December 2015.