Email Thread Extractor

Chen Hu, Yinzhi Yu

December 21, 2016

1 Introduction

Everyday communication among people is important and is done through different channels. In modern days an important channel is email. For this reason it is important to analyze it and understand how it works.

One of the main problems while trying to analyze emails is that it comes in different forms. For this reason, it is important to structure its content for its understanding.

In this report, we present the details on a tool we developed during the semester to structure the threads from emails obtained from the Enron corpus. In addition, we describe some of the problems we met during this procedure. At last, we give out some statistical information computed on a sample based on the first 50,000 records in our database.

1.1 Implementation Logic

Our approach to solve the problem is by implementing an extractor based on a state machine which scans each line in the email. By design, we defined that each component of the thread is separated by the headers found in the emails.
The thread header is recognized by five key features:

1. –Original Message–
2. – Forwarded by
3. To:
4. From:
5. Regular Expression(Email_Address on Time)

The thread header is ended by different criteria in different type of headers, the corresponding ending criteria are:

1. The line with “Subject:”. If not, the line with “From:”. If not, the second line in thread head.
2. The line with “Subject:”. If not, the line with “From:”. If not, the second line in thread head.
3. The line with “Subject:” or “X-Mailer”. If not, the line with “Re:”. If not, the second line in thread head.
4. The line with “Subject:” or “Re:” or “Date:”. If not, the line with “Sent:” or “To:”. If not, the second line in thread head.
5. The line with “Subject:”. If not, the line with “To:”. If not, the second line in thread head.

According to these criteria, we separate each email into threads and main bodies. For this we adopted a rule-based framework and implemented it as a finite state machine. The State Machine has two states: "Head" and "Body", which represents the thread header and thread doc respectively. The "Head" state has 5 sub-states: "Original", "Forward", "To", "From", "Email", which correspond to 5 different headers. The state machine has following operations:

1. *transition(line, text, line_no)*: Given current line and current state, transfer from current state

2. *check_ head(line, text, line_ no)*: Check whether current line is the start of header, we use rules check "Original", "Forward", "To" and "From" header and use regular expression to check "Email" header

3. *transFromBody(line, text, line_ no)*: Given current line and context, transfer from "Body" state

4. *transIn[Org/Fwd/To/Fm/Eml](line, text, line_ no)*: Given current line and context, transfer from "Head" state and sub state is "Original", "Forward", "To", "From" or "Email"

5. *find[Org/Fwd/To/Fm/Eml]End(line, text, line_ no)*: Using context to help find the ending of current thread header

When we scan through the email line by line, the state transfers according to current line and line context. Therefore, we get header starting lines and header ending lines, which further produces thread header list and thread doc lists.

1.2 Difficulties

The challenge while we developed this extractor is the diversity of thread headers and their implicit boundaries. The rule-based framework is static and context-free while the thread header is dynamic and context-related. For example, each header has different and unfixed ending criteria. Discovering all the possible ending features a priori is impossible considering the huge amount of emails. Besides, different types of header will interfere with each other. For instance, the feature "From:" is the starting feature of "From" header but also might be the ending feature of "Forward" and "Original" header.

To overcome the difficulties aforementioned we adopted a context-related rule-based framework. We used the Linux based command *find[Org/Fwd/To/Fm/Eml]End* to find the corresponding ending position of current thread header given the email context.

1.3 How to Find New Pattern and Problems

In our experiments, we went through approximately 150 emails and found thread patterns manually. Inspite of the specific patterns, we also found out some key words in a thread: To, From, Subject, Cc, Forward, etc. We first developed the extractor only based on these specific patterns. We ran our program on a larger dataset consisting of 500 emails and we got 2 files, one is formed by all the threads, the other is formed by all the main bodies. We searched those key words we found, as described above, to manuallt check if there were any missing patterns. To be specific, we searched "Subject" in the file containing all the main bodies, we manually checked if this occurence was supposed to be part of a thread. If it was, we easily got a new thread pattern and we added more constraints or adjust some settings in our program. After several iterations like this, we ran our program on a large dataset containing 50,000 emails and got a clear division on the thread and main body.

Nonetheless, our solution is far from perfect since there some patterns may exist that we have not yet discovered from our training corpus. For this reason, we decided to ignore those cases for two reasons, first because they make program less readable, and second is their relatively higher potential to misclassify a part from the main body and be assigned to a different thread. Also, those cases are guaranteed to have low frequency in a substantial sample of 50,000 emails.

2 About the Program

2.1 File Name

We implemented the extractor in Python and can be found under the file:
Extractor_v2.py

2.2 Usage

$ python Extractor_v2 path_to_email_folder path_to_email_usrs

path_to_email_folder: The path of a folder in which we store all the emails as separate text files. The name of each file corresponds to the "id" field of this email in database.

path_to_email_usrs: The path of the file in which we store the user of each email with email ID. The format is "id usr"(separated by a blank space) per line for each email in the folder above.

2.3 Output

The processed emails are stored by default in JSON format within the file `email_threads.json`, the format is the following:

{
id:{
"thread_doc":[[start_index, doc of thread],...],
"thread_head":[[start_index, head of thread],...],
"usr": usr_name
} ...
}

* The type of id is string

Additionally to the information extracted, the program also extracts some basic statistics regarding the threads from the e-mails. The average threads number for each user is stored in email_threads.sta, in the following format: User_name average_number_of_thread

3 Statistical results

Statistical data of each user (18 users in all) based on the first 50,000 records in database can be found in file "userInfo50000", including Avg Email Length, Number of Emails, Number of recipients, Avg Number of Thread. These statistics are also visualized in the following charts.