Email recipient recommendation

Report for Master Data Science/MVA
data competition 2017

ZHENG Chengyi, TIAN Yu, ZHOU Peikun (team: Daft.py)

March 19, 2017

1 Competition Description

It was shown that at work, employees frequently forget to include one or more recipient(s) before sending a message. Conversely, it is common that some recipients of a given message were actually not intended to receive the message. To increase productivity and prevent information leakage, the needs for effective email recipient recommendation systems are thus pressing.
In this challenge, we are asked to develop such a system, which, given the content and the date of a message, recommends a list of 10 recipients ranked by decreasing order of relevance.

2 Method Part

In this section we introduce you the detailed methods we used. The first part is data cleaning, we removed some wired data. The second part is text pro-processing, we tried several approaches to extract effective text for further training step. The main idea is computing a ranking result by the aggregation of time stamp and content, our methods mainly refered *Towards Building Effective Email Recipient Recommendation Service*

2.1 Data Cleansing

This part we concentrated on data cleaning. Firstly, we want to make sure that the mail in training_info and training_set are bijiection, that is to say, all of the recipients can be found in senders. We can clean then use effective training data set in this way. So we defined a function named check_info_set, which compares training_info with training_set according to the attribute "mids". Then we know that there are 43613 mails either in training_set or training_info, and they are in bejection.
In order to obtain a clearly data set, we merge training_info and traing_set into one single file named megered_train. In merged_train.csv file, there are five columns(attributes), mid, sender, data, body, and recipients. Table.1 presents a head of it.

mid	sender	data(time)	body(mail content)	recipients
9716	michelle.cash@enron.com	21/12/1998...	Brent,Attached is...	brent.hendry@enron.com...

Table 1: Merged_train.csv

There are a few mails whose time stamp are in 1775, while the others are around 1998 to 2001, so we thought that 1775 is a weird date data need to be removed. In recipients, there are some invalid mail addresses where is not included, we also remove them. We also cared about the distribution of recipients' number, most of mails have less than 10 recipients, a little of mails have 25 to 175 recipients. Considering that big recipients only count a little over all, we thought those mails who have more than 25 recipients are noisy mail and decided to remove them. We also do the same cleaing for test data, at last we recived a megeres_test.csv file.

2.2 Text Preprocessing

In this section we introduce our text processing method, especially for mail content text. With the help of NLTK library, we firstly cleaned stop words like "the", "is", etc. We also extracted stemming to reduce text complexity and improve efficiency.We save text pre-processing result into merged_train_textClean.csv and merged_test_textClean.csv.
According to mail content, we found that many of them are made up of RE and CC, in other words, the content may be superimposed by forwarding and CC which also produces some noisy information. So another method is under considering to extract the real effective content.

2.3 Ranking

In this section we combined time stamp with content, then received a ranking result as recommendation recipients. For time stamp, we computed recency score of message then received a recency ranking. For content, we used tf-idf method as weighting factor to compute another content ranking. At last, we aggregated recency ranking and content ranking with specific merging method and received the final result.

2.3.1 Recency

Each mail has a time stamp, we use power function to measure the recency of message. Assuming the timestamp of the new message m_{new} is $t(m_{new})$, the recency score of each message m_i is calculated by the power function, like

$$S_{rcy}(m_i) = (t(m_{new} - t(m_i)))^{-\lambda}$$

where $t(m_i)$ presents the timestamp of the message $m_i \in M_s \cup M_r$, M_s and M_r present the sent message and received messages in the mail history respecitivity. λ is the smoothing factor for the power function.

The next step is to compute the weight of the social network. For paticipants v_i and v_j in this social network, $w_r(v_iv_j)$ which is the weight of the edge v_iv_j is calculated as follows,

$$w_r(v_iv_j) = \omega \sum_{m \in M_s(v_iv_j)} S_{rcy}(m) + \sum_{m \in M_r(v_iv_j)} S_{rcy}(m)$$

where $M_s(v_iv_j)$ presents the sent message set, in which the message contain both v_i and v_j. Simillarly, $M_r(v_iv_j)$ presents the received message set, in which the message contain both v_i and v_j. ω presents the relative importance of sent messages versus received messages. $S_{rcy}(m)$ presents the recency score of messages m, which is computed by the last formula. Following this, we defined two functions $recency(t\_new, t\_m)$ and Recency_rank(new_mail, address_books, contact_books, df) to compute recency.

2.3.2 Content

Content is another important component in the prediction for the email network. In our method we used tf-idf, short for term frequency-inverse document frequency, to analyze the content of the email.
Term frequency is number of times a term occurs in a email. It is calculated by count the number of times each term occurs in each document and sum them all together.
But tf is not sufficient in many casese. For example, some terms are too common, solely using term frequency will cause incorrectly emphasis on documents which use the these words too frequently, without giving enough weight to other perhaps more meaningful terms. These terms are not a good keyword to distinguish relevant and non-relevant information. Therefore, an inverse document frequency (idf) factor is incorporated which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely. As Karen Sprck Jones (1972) pointed out: the specificity of a term can be quantified as an inverse function of the number of documents in which it occurs.
Our implementation of tf-idf is directly using *TfidfVectorizer* of *scikit-learn* library to analyze the pre-processed content of emails.

2.3.3 Rank Merging

The distribution of the recency score is very different from the result of the content score especially in terms of order of magnitude. How to weight and merge the two results is quite vital to the final result. Our approach combines the rank of two results based on mean reciprocal rank.
For each candidate vertex v_i, the final rank is the weighted reciprocal sum of the rank of recency and rank of content:

$$Rank_{final} = \frac{\alpha}{Rank_{content}(v_i)} + \frac{1-\alpha}{Rank_{recency}(v_i)}$$

in which α is the parameter that adjusts the impact of each component. Higher the aggregated rank is, more recommended the recipient will be.

2.4 Evaluation

The final evaluation is based not only on correctness, but also on priority. A list of less than ten ranked recipients will be recommended for each email. The actual lists of recipients of emails has no ranking. But if a correct recommendation has a higher ranking, the methodology will get a better evaluation.
In detail, from the beginning of a recommendation list and onward, the evaluation will first check if a recommendation is in the actual list of recommendation. If it is correct, the parameter *hits*, which is set 0 as default for each list of recommendation, will increase by 1. And the parameter *score*, which is also set 0 as default for each email, will add the quotient of hits divided by the ranking of the recommendation.
Mathematically speaking, for each match in a list:

$$hits = hits + 1$$
$$score = score + \frac{hits}{ranking}$$

where *hits* and *score* are set to 0 for every email. In the end of matching process of each email:

$$score = \frac{score}{min(number_{of\_actual\_recipients}, number_{of\_recommended\_recipients})}$$

The final score for all recommendation lists is the arithmetic mean of the scores of each list. Our methodology scored 37.49 in the final evaluation.

3 Conclusion and Future Extension

Using various techniques such as text-cleaning, tf-idf and time-correlation analysis, the *Daft.py* analyzed and measured a social network connected by more than 43,000 emails. The final product of this project is a email recommendation system that generates a list of recommended recipients based on the content and time of an email. The recommendation system scored 37.49 in the quantitative evaluation.
The model we proposed excels in time-relevance because of its focus on recency of communication social network. Its performance is also satisfactory in terms of content analysis due to the compliance of text-cleaning strategy and the choice of tf-idf method. However, its lack of weighting on communication frequency may negatively affected the precision of recommendation. A better content extraction, which distinguishes the main body and forwarded part of an email was also part of our plan. But we failed to implement it in time.
Several extensions that may significantly improve the email recommendation system also exist. The centrality and network analysis based on the knowledge of typical company's infrastructure and client structure, for example, can be an interesting extension for the project. We also plan to continue on content computation using topic model to alleviate term mismatch problem. The full vectorization of words and sentence structure (*wordvectorizor* for example) is also worth exploring. We would also like to consider the concept

drift problem of the model to deal with the complexity of data and related events. Moreover, We also found errors in send time of 200 emails from the same sender. Most of these emails were recorded as sent in year "0001" instead of "2001". Since our methodology is very time-relevant and sensitive, fixing this error may also imporve model.