Email Recipient Recommendation

Marc Sanselme
MVA
ENS Cachan/Ecole Polytechnique
marc.sanselme@polytechnique.edu

Christophe Lanternier
MVA
ENS Cachan/Ecole Polytechnique
christophe.lanternier@polytechnique.edu

Abstract

This project was realized in the frame of the class "Advanced learning for text and graph data" taught by M. VAZIRGIANNIS, during the spring 2017 semester of the MSc "MVA" at ENS-Cachan. The name of our team was **Sanselme-Lanternier**.

Introduction

Our work on this project was mainly inspired by paper [1], and [2], which study the same problem and solved it with good results. After a main focus on implementing and improving the method called *K-Nearest-Neighbors* presented in 3.1.2 of paper [1], which we will refer to as the hand-crafted approach, we tried to get the best out of methods using learning algorithms presented in both [1] and [2].

1 Feature engineering

1.1 Feature extraction

As a first step, we need to extract information from the raw data, to be able to build a predictive model on it. The first set of features we decided to extract from emails are :

- tf-idf representation of the body of the email
- date (and more precisely temporal proximity)
- name contained in the first 10 characters

For the first point, we use scikit-learn library to train a tf-idf model representation of the whole training set. Then, for every email of the training and testing set, we compute its tf-idf representation. We use it to compute cosine similarity between emails and measure their resemblance in vocabulary. For a given email to predict, we compute the similarity of all the emails of the training set with the same sender and keep the k highest scores, let's call this set of emails kNN. We define the *kNN-score* of a person in the address book as the sum of the scores of all the emails that are in kNN and where this person is recipient [1].

$$\textit{kNN-score}(person, mail) = \sum_{m \in kNN} \mathbf{1}_{recipient(m)}(person) \times similarity(m, mail)$$

For the date, we rank emails in kNN from the oldest to the most recent. Let *rank* be their normalized ranking positions, same ways as the *KNN-score*, we can compute the *recency* feature:

$$\textit{recency}(person, mail) = \sum_{m \in kNN} \mathbf{1}_{recipient(m)}(person) \times rank(m, mail)$$

For the name, we can parse the email addresses to get first and last name and try to find these strings in the beginning of the email, hoping to see the name of the recipient in the greeting. We define the header value of an address as 0 if none of the strings are present in the beginning of the email, 1 otherwise. We will refer to this feature as *header-value*.

In addition to those three features, we implemented features that allow to capture information from the sender's networks ([1] and [2]): who does he receives the most emails from, and who does he sends the most emails to. To do so, for a couple (sender, recipient) we extract network features of two kinds :

- for a given recipient, the normalized sent frequency, which is the number of messages sent to this recipient by the sender divided by the total number of messages sent by the sender.We will refer to this feature as *NSF*.
- for a given person, the normalized received frequency, which is computed by dividing the number of messages received from this person by the sender divided by the total number of messages received by the sender. We will refer ti this feature as *NRF*.

1.2 Data representation

To create our handcrafted train features, we browsed the training set, taking for each email the ground truth (in order to create labels), and computing the five previously described features based on the k closest emails ("closest" meaning having the highest cosine similarity on tf-idf representation to the considered email). Here is an example of those features:

sender: jaime.lannister	kNN-score	Recency	NSF	NRF	Header-Value	Ground Truth
john.snow	0.6	0.4	0.2	0	1	1
tyrion.lannister	0.3	0.4	0.1	0	0	1
cersei.lannister	0.1	0.2	0.05	0	0	0

We observe that the *NRF* column is filled with zeros, which doesn't seem very insightful. This is a problem we had with this feature: given that this enron database contains only 125 senders for 9870 recipients, we have relatively few informations concerning the emails that were sent to the considered sender. In this particular example it is very likely that John, Tyrion and Cersei aren't in the list of the 125 senders.

2 Model tuning and comparison

A big problem we had when it came to build the best prediction model as possible was that the results estimated on our local test/train split were very different from the results we had after submission, because of over-fitting. We solved this problem the following way:

- First, we realized the importance of chronology in this challenge: all emails from the given test set were posterior to the ones in the training set. After solving some formatting error (part of the mail for sent in year "0001" according to the original training set, which we fixed to "2001") we organized our local train/test split accordingly, and split at a chosen date (which allowed to still have a good amount of emails in the training set).
- Second, we had to introduce randomness in the way we chose our test emails, in order to make sure to have the same diversity of emails between the train and the test. We did that by selecting a thousand emails randomly, dated from after the split date (which was August 24th, 2001). Our models were then tested on several of different draws of this train/test repartition, and we looked at the mean performance.

2.1 Hand-crafted approach

Our first try was to just return the 10 highest *kNN-scores*. The result is slightly better than the baseline : the public score was 0.34 with $k = 70$.
To improve this, we tried to rank potential recipients according to the product of a function f of the *kNN-score* and a function g of the *recency*. We tried different functions : linear, exponential, logarithmic.

$$score = f(kNNscore) \times g(recency)$$

The objective of trying those various functions was to find the best balance between both of these features. Here are a few examples of the results we got :

k	f	g	Score of submission
30	linear	linear	0.38403
70	linear	linear	0.39959
70	exp	exp	0.36216
70	linear	exp	0.339
70	exp	linear	0.35
70	log	linear	0.39420
70	log	log	0.38157

By trying and changing these parameters, we got an idea of the optimal value for k and realized that the linear functions were more efficients for f and g. Also, we see that both features are very important and increasing the importance of one of them with the exponential gives worse results.
The next try was to increase the score (by adding a constant) when the name of a recipient appeared in the beginning of the mail :

$$score = f(kNNscore) \times g(recency) + header \times C$$

We didn't manage to find a right value for C that would improve the result but we kept this feature for the learning approach.

2.2 Learning approach

In the first attempt of learning, we didn't use our features, but rigorously the ones depicted in section 3.2 of paper [1], which included *kNN-score*, *NRF*, *NSF*, and a different version of the *recency*, in the sense that instead of ranking emails contained in the kNN set as we did and taking the k first, it ranks all recent emails sent by the sender (so not necessarily close in terms of cosine similarity) and selects the k first. First attempt didn't include the *Header-Value*. The same types of features could also be depicted in paper [2].

For this first attempt we tried four different algorithm: Logistic Regression, Adaboost [4] on Decision Trees, Random Forest [6] and SVM. While Decision Trees models (Adaboost and Random Forest) are pretty low performers (score ranged from 0.21 to 0.25), SVM gave acceptable scores (around 0.35), and Logistic Regression yielded the best performances (submission gave **0.37924**).

In the second attempt with used all five features described in part 1, and focused on the Logistic Regression, which had the effect of improving performances (ranged from **0.38** to **0.395**). Changing k from 30 (as suggested by paper [1]) to 70 (which gave us our best performances in the hand-crafted approach) gave us the high score for the learning approach, which was **0.39564**, our second best submission overall.

In an attempt to improve this score, we made the observation that the algorithm was learning on an unbalanced set (200,000 positive samples over 2,000,000 total samples), so we implemented an over-sampling method, called SMOTE [3] (Synthetic Over Sampling TEchnique), which creates new synthetic positive examples in the features space, close to authentic ones, allowing the algorithm to learn on global region of the feature space and not isolated examples. More details on this technique are given in the cited paper. This technique did not improve the results.

Further exploration and Conclusion

A lead for further exploration could be to work on the text preprocessing and vectorization. Indeed, lots of the emails, the ones that are a long chained of forwarded email for instance, contain more noise due to the mail client (all tags like "—-FORWARDED TO——", etc...), than actual content. This has the effect that if we are trying to predict recipients on a forwarded email, cosine similarity will only yield other forwarded emails from the same sender, regardless of the topic and real content of the mail. Being able to "clean" all those emails could improve performance.

Paper [2] mentions that instead of using the classical tf-idf, it uses the Okapi BM25 textual similarity measure (which was the one we used in lab 4). We tried to implement this method, but couldn't go through with it because of timing issues: our implementation was to slow to process the amount of emails we had, and we were getting pretty close to the deadline.

To conclude, the most efficient methods were after all, the simplest ones. The similarity between texts of different emails, with a tf-idf model, was a way to select emails with similar subjects and the proximity in time was a decent proxy for the probability of sending a similar email. All the other methods gave us, sometimes, some improvement, but didn't change radically the score.

Note: the `readme.txt` file provides explanations on how to use the code.

References

[1] Vitor R. Carvalho and William Cohen. Recommending recipients in the Enron email corpus. *Technical Report CMU-LTI-07-005, Carnegie Mellon University*, 2007.

[2] Zvi Sofershtein , Sara Cohen, Predicting Email Recipients, *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, August 25-28, 2015, Paris, France

[3] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Philip Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research 16, 321357*, 2002.

[4] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences , 55(1):119-139*, August 1997.

[5] Robert E. Schapire. *Explaining Adaboost*

[6] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, *An Introduction to Statistical Learning*, 2013.