Advanced Learning for Text and Graph
Linagora challenge report
Not1stYet

Hugues Tavenard, Nathan Rouxel
hugues.tavenard@eleves.enpc.fr, nathan.rouxel@polytechnique.edu

16th March 2017

Contents

1 Feature engineering

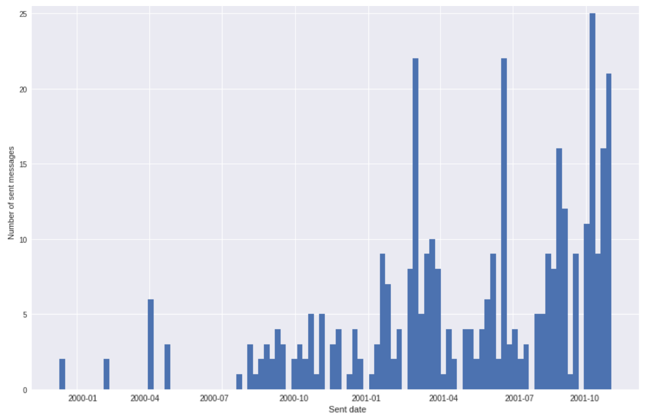
For convenience, we define the *address book* of a sender as the users to whom the sender has already sent a message. Besides, we will call a *contact* of a sender A a user in address book of A. We will also call *sent frequency* from a user A to a user B the number of messages sent from A to B divided by the number of messages sent by A in total.

We first joined the given csv files in order to obtain a unique csv with columns [”Date”, ”Mid”, ”Sender”, ”Body”] for test csv, and the same columns in addition to [”Recipients”] for training csv. The challenge frequency baseline predicts the 10 users with the largest sent frequency in the sender address book, whatever the message is, and performs 0.34 on the private leader-board.

1.1 Date preprocessing and date selection

Some dates were initially invalid, as the associated mails were sent in year 0001 and 0002, so we changed them to 2001 and 2002 respectively. We noticed that this leads initial train and test csv to overlap : the more recent message in train is 127 days after the oldest message in test but only 141 messages from train have been sent after the oldest message in test.

Figure 1: Messages sent from james.d.steffes to richard.shapiro over time



We observed a kind of concept drift after 2001 in sent frequencies between preponderant senders. For example, Figure 1 represents the number of messages sent from james.d.steffes (who is the user who sent the biggest number of messages) to richard.shapiro (who received 551 mails from james in total) over time. We eventually decided to focus on recent data and to only select the messages sent after June 2001, reducing the size of train csv from 43613 to 17378 messages. This led to a significant increase of the validation score of the baseline, from 0.34 to 0.37. This date selection also helped every model ever tried after, and led to a significant speed up in training and validation execution time during the challenge.

1.2 Building of the machine learning training set

To be able to use machine learning algorithms on the data, we built a training set following a similar idea to [1]. For each sender and each message sent by this sender, we consider the recipient prediction as a binary classification problem. For each contact of the sender, we build a line in the training set where the columns are the features described in the following sections and associated to the contact, in addition to a label which is 1 when the contact is a recipient to the message, 0 otherwise.

Hence, a sender associated training set has a length of $M_s|AB_s|$, with M_s the number of messages sent by the sender in the train csv and $|AB_s|$ the length of its address book. Moreover, there are 125 different training set, one for each sender in the train csv.

1.3 Time-period sending frequency features

Following the baseline idea, the first feature used was the ”global sent frequency” from the sender to the user in the train csv. For example, if Suzy has sent half of her messages in the train csv to James, then for each message sent by Suzy, the line in the training set associated with Suzy and corresponding to James will have the value 0.5 for feature ”global sent frequency”. Generalising this approach leads us to finally consider ”last 9 months sent frequency”, ”last 6 months frequency” and ”last 3 months frequency” features, in order to give to our model an idea of the evolution of the sent frequency from a user to another in her training set.

1.4 Message-period sending frequency features

In addition to the previous features, we added ”last 30 messages sent frequency”, ”last 50 messages sent frequency” and ”last 100 messages sent frequency”, which are the sent frequencies only computed on respectively the last 30, 50 and 100 messages sent by the user considered.

1.5 Temporal features

We also used the features ”day” (which is 0 if the day is Monday, 6 if the day is Sunday) and ”hour”, which is the hour of the message (from 0 to 23). The intuition behind these features is that the sending patterns during the week-end, during working hours and out of working hours are different. These features should help the model to identify those patterns.

1.6 K-Nearest-Neighbors-30 scores

The idea behind K-Nearest-Neighbors (Knn) scores is to use the information contained in the text of the emails as features, the intuition being that people do not use the same vocabulary to talk to their boss as they do when talking to somebody in the accounting department or to somebody in the IT department.

The Knn-scores method is described in [1]. Since the senders are treated separately, in the following we consider the messages of only one sender. We first compute the TF-IDF representation of the training messages of this sender, therefore obtaining for each message a normalized vector $\vec{m}$. Now, in order to compute the score of each contact for a message that belongs to the test set, we compute the TF-IDF representation of the body of this message relatively to the training messages, obtaining a new normalized vector : $\vec{m}_{test}$. The next step consists in computing the TF-IDF cosine similarity between the test message and each training message by computing the scalar product: $\langle \vec{m}_{test}, \vec{m}_{train} \rangle$ for each train message. The score of a recipient r_i for the message m_{test} is finally obtained by computing the sum of the similarities between $\vec{m}_{test}$ and the messages of which r_i is a recipient among the 30 most similar training messages. In [1] the number 30 is considered to be the best one, which is also the best one on our train set. For smaller values, the cross-validation results were not as good and for larger values the improvement was negligible but the training took more time to run.

This Knn method can be used as described above, in which case for each test message, each contact has a score, but it can also be used using a cross validation procedure to build machine-learning features without overfitting. In this context, the training set is split in 10 folds, we compute the knn-30 score on 90% of the data and we make predictions on the remaining 10%. By doing this on the 10 folds, each message in the training has a score for each recipient and these scores can be given with other features to a machine learning algorithm.

It should also be noted that we did not apply TF-IDF to the raw messages, we preprocessed the email bodies by removing punctuation, converting the text to lowercase and stemming the words using the nltk python library.

1.7 Additional prediction modules

1.7.1 Co-occurrences module

Another intuition we had was that in this challenge, finding the first recipients to include in the prediction of the messages is a (very?) different problem from finding the last recipients to add to the prediction. Indeed, if we consider a stepwise prediction when the first recipient of the final 10-length list is predicted, then the second one, and so on until the last one, then it is possible to use the information of the previous predictions to predict the next one. In other words, we could consider a structure in the output, and predict the first most probable recipient to the message, then the second most probable recipient *knowing that the first one has been predicted*, predict the third one *knowing that the two first ones have been predicted* and so on.

To easily (and quickly) test this intuition, we decided not to include the structured output strategy directly into the algorithm, but to include a module which adapts the final prediction just after the machine learning algorithm output. The idea is to consider the confidence of the classifier that a user label is 1 for a particular message as an estimation of the probability that this user is a recipient of the message. Let say that for a particular message sent by a user u_0, the classifier gives to a user u_1 the probability $p(l_{u_1} = 1)$ that the label of the user is 1 (with l_{u_1} the label associated to u_1, seen as a random variable). Then, we make an estimation of the probability that a user is a co-recipient of u_1 using Bayes rules :

$$p(l_{u_1} = 1, l_{u_2} = 1) = p(l_{u_2} = 1 | t_{u_1} = 1) p(l_{u_1} = 1)$$

And considering that an estimation of $p(l_{u_2} = 1 | l_{u_1} = 1)$ can be given by the empirical co-occurrence of u_2 and u_1 in recipients messages sent from u_0. Our co-occurrences module computes the most probable co-recipients of the 5 first predictions to complete the 10-length prediction list (only users who are unique and different from the previously predicted are selected).

1.7.2 First name recognition module

We noticed that a lot of email bodies in the train csv contained the first names or surnames of some recipients of the message. Typically, an email whose body begins by "Bob, Sarah–Could you please have a look at [...]" is (at least) sent to Robert and Sarah. We were able to use the format of enron email addresses *firstname.lastname@enron.com* to identify some obvious recipients. As the classifier was already pretty slow to train and predict, we decided not to add sender address book first names and surnames as features, but to add instead a module which changes the final prediction after the classifier output. The person name taggers were not performing very good in Python (we mainly tried Stanford NER Tagger), so we only used custom regex to look for address book compatible first names or surnames in the beginning of the email body written by the sender. The strategy was simply to add at the top of the 10-length list prediction the address of the user whose first name matches the regex, and for whom the classifier has the more confidence to classify him as a recipient.

2 Model tuning and comparison

In order to compare our models, we have used two validation strategies, the first one is a simple 3-fold cross-validation but the scores it returned were quite far from the leader-board scores. The reason is that the dates in the training set are mostly before the dates of the test set which is not the case in a random 3-folds cross-validation. However, this validation method gave good results on the variations of the score. The second validation strategy reproduces the temporal relation between train and test csv. In this strategy, the validation test set consists in the most recent messages from train csv, and the predictor is trained on the older messages. The time overlap between train and test csv, the time period of test csv and the proportion between the number of messages in test and in train csv are reproduced. This validation method led to evaluations with 10^{-3} precision of private leader-board error on our last submissions.

Our best supervised classifier is a gradient boosting algorithm (LightGBM) which minimizes a log loss during classification and uses Knn-scores and time-period sent frequencies features with first name recognition module at the end to reorganise the output. It gave a leader-board score (private score) of 0.39046. This is significantly less than our best performing model but it is much better than the initial baseline.

Finally, our best model consists in combining different features that gave good results separately but without using machine learning. For a given sender and a given test message, we compute the Knn-30 score of each contact for this message, which yields the 10 best contacts according to this criteria (there can be less than 10 contacts if the address book of the sender is small). Now, if our first name recognition module recognizes first names (maximum 2), it puts them at the top of the list of the 10 contacts. The last step consists in using frequencies in the address book to make predictions for the 9th and 10th spots in the predictions list. This model gave a leader-board score (private score) of 0.40261 which ranked us 7th overall. However, in the last night of the challenge, we left a bit of code that we should have removed when running our best submission (we used the square of the similarities instead of the similarities in the Knn). Without this mistake we would have had a score of 0.411.

Sender train errors analysis were a good way to identify specific users and to guide reading of training or test emails.

3 Way of improvement

We used LightGBM feature importance to guide our intuitions on feature selection. Surprisingly, we noticed that temporal features were used sometimes instead of time-period sent frequencies by gradient boosting, so we are convinced there is a way to use them better than what we do.

A promising way of improvement would be to improve name recognition module to make the difference between bodies "Leslie–Please help me with" and "We talked with Leslie yesterday and ..." where the first one suggests Leslie is recipient, while the last one does not. Name recognition best improvement would be to include first names in train set features, in order to leave the gradient boosting decides which features are important to consider to optimize the loss.

Another way of improvement we were about to explore was to allow to predict recipients out of the sender address book. In some context, this is not conceivable (we will find very weird if Gmail suggests us as recipient the email address of a person we never sent a message to before), but in the context of a company, that might totally make sense. The risk of this approach is to degrade the prediction performance because of the too large possibilities of prediction. But this will handle incomplete predictions problem: sometimes, we are not able to suggest 10 predictions in our approach when the address book of the sender is very small. In this case, the users who are the most contacted in general or the Knn feature apply to a larger set of recipients than address book could give more prediction options. These predictions out of the address book could be restricted to the first name recognition module for example.

Finally, a last idea would be to stack multiple models with different set of features, or possibly different losses. Minimizing a log loss and then rerank the predictions as we do certainly does not minimize a MAP@10 metric. Using ranker instead of classifier greedily minimizing MAP@10 metric could be another way to go.

References

[1] Vitor R. Carvalho and William W. Cohen. Recommending Recipients in the Enron Email Corpus. 2008.