Linagora Data Challenge: Email Recipient Recommendation
Advanced Learning for Text and Graph Data

Team 450euros: Salma El Alaoui Talibi Chia-Man Hung Zhengying Liu
Mario Ynocente Castro

March 19, 2017

Abstract

In this challenge, we adopted a rather sophisticated preprocessor and utilized different feature extractors such as naive bag of words (BoW), tf-idf, word2vec, etc. Then we applied several multilabel classifier/regressor including (one-vs-all) logistic regression, linear SVC, decision tree, gradient boosting, k-nearest neighbors (kNN), etc. Sometimes we also combined the predicted results of ML algorithms with information in address book. We got a (public) leader board score of 0.17 for BoW + kNN, 0.24 for tf-idf + LinearSVC, 0.31 for word2vec + decision tree + address book and 0.35 for tf-idf + LinearSVC + address book.

Contents

1 Introduction

In this data challenge, we are given a collection of emails along with their sender, date, content, recipients and we are asked to provide a list of ten recipients ranked by decreasing order of relevance for each new email along with its sender, date and content. The performance is evaluated by the Mean Average Precision @10[1].

We consider the problem as a **multilabel classification** problem, where the target is the columns of the 0-1 encoding of each possible recipient. The value is 1 if the recipient received this message and 0 if not.

Another important question is: should we apply machine learning algorithms for each sender or for all senders. Or equivalently, should we construct a model for each sender (discriminative approach) or just one single model for all senders (generative approach)? As it turns out, we'll use a generative approach when applying the k-Nearest Neighbor algorithm and a discriminative approach when applying other multilabel classifiers. This is the consequence of a trade-off between performance and efficiency.

This report is organized as follows. First, we will explain how the feature engineering is done and how we tackle the task of feature selection. Then, we will propose several models that we experimented with. We will compare our different approaches and develop how the parameters are tuned. Last, we will show the results we obtained by following our different approaches.

2 Feature Engineering

2.1 Preprocessing

In our code[2], we used extensively Pandas data frame (the class `pandas.DataFrame`). We constructed 4 main Pandas data frames: `training, training_info, test, test_info`, containing respectively the information in their corresponding `.csv` file. We also added several features (columns): a column `list_of_mids` in `training` (and `test`); a column `address_book` of python dictionaries in `training` where for each sender, the keys are email addresses of recipients and values the number of emails sent to this recipient from the sender; a column `sender` in `training_info` to indicate the id of the sender of the message.

By manually reading some emails, we noticed that in some forwarded or replying emails, information like "—-Original Message—-" and lists of senders and recipients below it might cause some inconvenience to feature extractor (e.g. tf-idf or simple bag of words). So in the preprocessor function `clean_raw_text` in `utils.py`, we use some regular expressions to delete these parts. Here is a list of what we cleaned in `body`: 1. URL; 2. Inline CSS/JavaScript; 3. HTML tags; 4. Forward/Sender/Date information (but we kept the subject part following "Subject:"); 5. Long non-character string (that are hardly to give useful information) like "713/853-4218", "$80,400.00". We also found many "\t" in certain emails and we decided to replace them by a space.

In addition, there are many words that are sticked together due to mistyping or some other technical reasons, e.g. "...Subject: Gas PartyKim, please invite the Following Texas Eastern folks to the party.Leah MossJanice DeversBerk DonaldsonThe address is5400 Westheimer CourtHouston, TX 77056-5310". So we decided to add a space before every uppercase letter that is preceded by a lowercase letter or a number. Notice that we did this before transforming all letters to lowercase.

[1] More details can be found at `https://www.kaggle.com/wiki/MeanAveragePrecision`

[2] The source code and instructions for running the code of different models can be found at `https://github.com/zhengying-liu/email-recipient-recommendation`

2.2 Text feature

2.2.1 Bag of Words

After applying the preprocessor, we tried several different feature extractors. The first one is Bag of Words (BoW). For each document, BoW counts the number of occurrence of each word and associates a sparse vector of positive integers to this document. BoW is very naive but very simple to implement and often serves as a baseline feature extractor.

In our implementation, we use the class `CountVectorizer` in `sklearn.feature_extraction.text`.

2.2.2 TF-IDF

The abbreviation tf-idf stands for "term frequency - inverse document frequency", which is defined as the product of the logarithm of term frequency times the logarithm of the inverse document frequency, for a given word (or "term"). This is also a bag of words method (which ignores the order of words in a document) and returns a sparse vector of floating numbers. tf-idf is overall better than simple BoW because it characterizes better the amount of information carried by each word.

In our implementation, we use the class `TfidfVectorizer` in `sklearn.feature_extraction.text`.

2.2.3 Word2vec

Word2vec allows us to use pre-trained weights in a shallow neural network to have a relatively good embedding for each word even if we don't have a large corpus (about 46000 documents in our case). As the dimension of such embedding is often less than 300 (in our case dim=200), we can apply any ML classifiers like decision tree, random forest and gradient boosting without worrying about poor efficiency, which is the case if we use high-dimensional features (even if they are sparse).

In our implementation, we use the class `KeyedVectors` in `gensim.models.keyedvectors` and we set the embedding dimension to be 200.

2.3 Recipient feature

2.3.1 CharVect (Sum over received messages)

From all 3 of the bag of words features above (BoW, tf-idf and word2vec), we can construct a characteristic vector (CharVect) for each recipient too, where we simply compute the sum of the feature vector of all received messages of this recipient (probably from different senders). We can do this because all these three text features are bag of words methods, which ignore the order of words and thus are additive.

This feature of recipient can help to compute the similarity between a message and a recipient. In our kNN approach, we used **cosine similarity**, where we simply compute the dot product of the normalized feature vector of the message and the CharVect of the recipient.

2.3.2 Address book

To reduce the size of the searching space when doing prediction, we used the address book given in `baseline.py`. That is, we suppose that every sender will always send messages to the recipients that he or she had already contacted before. In addition, for each sender, we count the number of messages sent to each recipient and order them by this frequency. The baseline in `baseline.py` will then always predict the top 10 most frequent recipients for a given sender. This gives already a public leader board score of 0.344.

In our implementation, we add a column `address_book` to the pandas data frame `training` to store this information.

3 Model Tuning and Comparison

During the challenge, we tried several combinations of feature extractor and classifier/regressor. To test the performance of each approach, we implemented a train_test_split function for our `training_info` data frame and a metric function `get_validation_score` that gives the MAP10 score of each prediction on validation set.

3.1 BoW + k-Nearest Neighbors

The first model that occured to us is a k-NN model with BoW features. And later we found that our approach is very similar to that of [1] and [2].

By using bag of words embeddings and CharVect, we can embed all the messages and all the recipients in the same space $\mathbb{R}^d$, where $d = 200$ for word2vec and $d \approx 10^5$ for BoW and tf-idf. Then for each new message, we can find the 10-nearest recipients in the **address book** of the sender (too slow if without address book) by cosine similarity and predict them in a decreasing order. With BoW + k-NN, this gives a score of 0.17 on the public leader board.

Score: 0.17148

Advantages: 1. Easy to implement; 2. No hyperparameters to tune; 3. Relatively fast.

Disadvantages: 1. The choice of similarity criterion is debatable; 2. Works not that well with high dimensional features.

3.2 Multilabel classification

For a multilabel classification job, we can naively apply a base binary classifier for each label (one-vs-all), which is similar to Naive Bayes in graphical models. This is definitely not the most ideal way to tackle our problem, where there is obviously an underlying structure between our labels. For example, some recipients might belong to a distribution list and will then always appear together. This naive approach will not detect this structure and will then work in a target space of much higher dimension. So it'd be better to learn some graphical model and reveal this underlying structure between recipients.

Unfortunately, the module `sklearn` doesn't support this feature and we didn't manage to implement an approach using graphical model. So all classifiers/regressors that we used during the challenge are their naive one-vs-all version for this multilabel problem.

3.2.1 word2vec + Logistic Regression + address book

With word2vec features, we work in a space of dimension 200. Thus we are free to use most off-the-shelf base binary classifiers: Decision Tree, Random Forest, Gradient Boosting, SVC, Logistic Regression etc. For simplicity, we tried Logistic Regression. And by combining the result of `predict_proba()` and the frequency in address book, we got 0.34907.

Score: 0.34907

Advantages: 1. Lower dimension with word2vec; 2. Relatively good performance; 3. Not many hyperparameters to tune.

Disadvantages: 1. Not enough samples for each model (there are senders who sent only 60-70 messages); 2. Naively supposes the independence between the labels.

3.2.2 tf-idf + LinearSVC (+ address book)

After getting tf-idf feature, we applied a one-vs-all LinearSVC classifier in a discriminative manner (i.e. a model for each sender). We used a linear kernel instead of other kernel because tf-idf feature is high-dimensional, to which kernels like 'poly' or 'rbf' are not very robust. Another reason is that

linear kernel works very fast with sparse features. Also due to this efficiency issue, we didn't choose models like random forest or gradient boosting for tf-idf.

As the metric function imposes the importance of order of the predicted list, we need to order our results according to their importance. But our LinearSVC will only give 0-1 information with `.predict()` method. To overcome this, we use the `LinearSVC.decision_function()` and then a softmax function.

To improve the performance of this model, we combined the "probability" predicted by the model with the frequency information in address book. This can be regarded as an ensemble.

Score: 0.24312 and 0.35150 with address book

Advantages: 1. Easy to implement; 2. Relatively good performance;

Disadvantages: 1. Hyperparameter tuning (e.g. `C`) for 125 models; 2. The choice of kernel is not flexible; 3. Not enough samples for each model (there are senders who sent only 60-70 messages); 4. A bit slow (30min for prediction).

3.3 Some other experiments

Before implementing the BoW + kNN approach to predict the 10-nearest recipients, we implemented a similar model which predict the exact list of recipients of the nearest message. This gave a very poor score: 0.01044. This is not very surprising as the variance of this model is huge.

We alse tried word2vec + Decision Tree + address book, which gave 0.31633.

4 Conclusion

By using the same preprocessor, we tried different combinations of feature extractor and classifier/regressor, and the highest score (0.35150) is obtained with tf-idf + LinearSVC + address book.

The improvements that we could make in the future are: 1. Use more robust structured output machine learning algorithms; 2. Construct a model that uses the underlying structure between the labels, instead of using Naive Bayes method; 3. Keep the deleted part in preprocessing as feature to give more useful information.

References

[1] V. R. Carvalho. *Modeling intention in email*. Springer, 2008.

[2] V. R. Carvalho and W. Cohen. Recommending recipients in the enron email corpus. *Machine Learning*, 2007.