Final Report - Smart and Fast Email Sorting

Antonin Bas - Clement Mennesson

1 Project's Description

Some people receive hundreds of emails a week and sorting all of them into different categories (e.g. Stanford, Studies, Holidays, Next Week, Internship, Friends, Graduate Activities) can be time-consuming. Most email clients provide a sorting mechanism based on rules specified by the user : sender's email address, key words for the subject... The aim of this project is to develop a machine learning algorithm which would learn from the manual sorting of emails by the user in order to quickly be able to handle itself the processing of new emails.

We do not want to have to wait for the user to label a sufficiently high number of emails in each category before being able to make a prediction. We are therefore looking to implement an online learning algorithm, which will make a prediction for each incoming email as soon as it arrives (even for the first ones, even though the tentative labelling is likely to be erroneous). However, the algorithm will not display its prediction to the user and move the email to a specific folder unless it is confident in its prediction. Indeed, an application which frequently mislabels email or worse, "loses" important emails by mistakenly placing them in a "garbage" folder rarely consulted by the customer, is just not worth using. An interesting part of the project is therefore to evaluate the degree of confidence of the algorithm in its prediction.

It is interesting to note that at least one open-source solution (POPFile) already exits which fullfils this task. POPFile is an email proxy running Naive Bayes, which inserts itself between the email server and the email client. However, it is our opinion that POPFile is not well adapted to webmail clients : the user has to run POPFile as a daemon on a machine which remains "on". If the machine is turned off, then the email does not get sorted anymore, which prevents mobile-only Internet users to sort their email with POPFile. Therefore, one of our goals will be to develop an email sorter which uses as few computing ressources as possible. Such a product may be able to run directly on web servers, and could be used directly by webmail providers.

2 Email Processing

2.1 Tokenization

An important part of our work was to obtain well-formated data on which to test our algorithms. We have developped an Email Retriever / Tokenizer / Labelling tool, with a user interface (UI).

Email Retriever We connect to a given IMAP email server (in our case Gmail) using Oracle's JavaMail API, and retrieve the Inbox content.

Labelling The downloaded emails are displayed one-by-one via the UI to the user, which can choose to give them a label or to leave them unlabelled

Tokenizer An email can have different content types : *text/plain*, *text/html* or *multipart/alternative*, when for example it combines a *text/plain* and a *text/html* representation of the same message. Our tool only supports emails with at least one *text/plain* part, which is the part we use for the tokenization, and is sufficient for our tests. Furthermore, most email clients tend to include a copy of the original email when you choose the Reply option. We have choosen to discard this copy from our input : for now, we consider that having the same subject is a good enough indication that two emails are related, and that the extra information is not needed. For the tokenization, we use the Stanford Log-linear Part-Of-Speech Tagger, which includes a stemming algorithm. We discard ponctuation tokens. Web addresses and email addresses are

respectively replaced by tokens $HTTPADDR\_domain$ and $EMAILADDR\_domain$, where *domain* is the domain name for the web / email address. The mail's subject, as well as the recipient and sender's addresses are also tokenized. The final output is a one-line text file (per mail) containing the label, and the number of occurences of each known token. Since we want to design an online learning algorithm, the representation of an email does not have a fixed length : each incoming email is likely to increase the number of known tokens and thus the length of an email representaion.

2.2 Other Features

There are several features one could select instead to lower the dimension of the problem. Such features include for instance : the size of the email, the number of receivers, the time period when it was sent... Using these features as regressors, one could expect to obtain results using supervised learning techniques. However, the loss of information is huge since the actual content of the email is never taken into account. Email Tokenization seems a better approach. Besides, it is always possible to identify the most relevant tokens for a particular category, and only feed those to the algorithm.

3 Description of the data sets

The algorithms presented are tested with two different data sets obtained from one of our Gmail boxes.

The table below shows the distribution of the data sets : number of categories, total number of emails, proportion of unlabelled emails (no label given by the user), average number of emails per category, lower / higher number of emails in one category.

Data Set	Nb of categories	Nb of emails	Unlabelled	Average number	Lower	Higher
1	10	565	118 (20.9%)	44.7	6	212
2	4	715	68 (9.5%)	161.8	18	500

TABLE 1 – Distribution of the data sets

The first data set specifies 10 very specific categories, with sometimes very few emails per category. For the second data set, we have tried to simplify the task of the classifier : there are fewer categories, and more emails per category. For both data sets, it would be impossible to define objective criteria to sort the emails (i.e. Gmail current sorting model using user-defined categories could not be applied)

Given the Gmail box we consider, these data sets are both reasonable email allocations. We do not try to fool the algorithm by wrongly labelling some emails. This way, we make sure there is no bias in data selection and that we are able to analyze the different behaviours of our algorithm accurately. We want to insist on the fact that there are unlabelled emails each time, which means we do not want to sort the inbox completely, as we assume most users do not.

4 Error and Hesitation

Each data set contains emails the user does not want to classify. Consequently, we distinguish between error and hesitation and introduce some criterion to measure the degree of confidence of the algorithm in his prediction.

Error : The algorithm makes a prediction for the email category or box, which it is confident about, but the prediction is wrong : the email ends up in the wrong category or receives a label whereas the user did not want it labelled. The algorithm should try to minimize the number of errors, as an important email could end up in a box of little interest to the user.

Hesitation : The algorithm makes a prediction for the email category or box, which it is not confident about. The email receives no label and is left in the main Inbox. Therefore, the user has to sort the email himself, but there is no risk of losing an important email. An hesitation is considered preferable to an error.

5 First Approach : Multinomial Naive Bayes Framework

5.1 First Implementation

We present here a first implementation of email sorting using the Multinomial Naive Bayes framework with Laplace smoothing. It is a natural extension of the spam/non spam Naive Bayes algorithm used in many spam classifiers, except that we now have several categories, each with its own Naive Bayes classifier. At each step of the algorithm, i.e. for each new incoming email, the training set is increased by one, and the dictionnary / tokens list is extended. As in the spam/non spam approach, we calculate the probability for the new email to belong to each category. We can note that the probabilities found do not necessarily sum up to 1. The most obvious example is when an email does not belong to any category. We describe below the different steps of the algorithm for each incoming email. We implemented it in MATLAB.

1. Train the algorithm with the n sorted emails.
2. Assign the $n+1$ email to a category based on the words contained in current dictionnary and qualify the confidence of the prediction. The category predicted is the maximum classification probability among the categories. The prediction is confident when this probability is higher than some value α, and all the other classification probabilities are lower than some value β. In the final tests, we choose $\alpha = 0.98$ and $\beta = 0.5$. Given the results observed, the choice for β has more impact as classification probabilites often end up being 0 or 1. A high beta means a high level of confidence in your training.
3. Check the correctness of the labelling. In case of mislabelling, add an error and reassign the new email to its correct category.
4. Expand the token list and update the representation of the trainig set ($n+1$ emails) in this new dictionnary .

This algorithm demands much computing : one Naive Bayes classifier is maintained for each category, and has to be updated for each incoming email.

5.2 Adding Relevance

The assignment of an email generally relies on keywords. Consequently, we select the most relevant tokens for each category to compute a relevance filter, excluding high-frequency tokens (which appear in all categories), and tokens which only appear sporadically in a category. Suppose we know $\phi_{j,i} = \mathbb{P}(token = j | y = i)$ for all (i, j), the relevance $r_{i,j}$ of a token j for category i is :

$$r_{i,j} = \log \frac{\phi_{j,i}}{\sum_{l \neq i} \phi_{j,i}}$$

The number of relevant tokens is a key parameter. A low number sharpens the classification but is sensitive to the variabilty of content in a category whereas a large number gives too much credit to each classification thus increasing the hesitation rate. The number of relevant tokens, as well as how they are computed, impacts the algorithm's performance time. Recomputing the list for each new incoming email is very time-consuming. However, when the categories are more "stable", it is not necessary to recompute the list at every step, and classifying incoming emails becomes faster (a reduced corpus means fewer operations).

5.3 Results

Data set 1 : 3.6% error and 13% hesitation, considering the 50 most relevant tokens for each category, $\alpha = 0.98$, $\beta = 0.5$.

Data set 2 3.6% error and 2.94% hesitation, considering the 100 most relevant tokens for each category, $\alpha = 0.98$, $\beta = 0.5$.

The error rate seems to be less sensitive to the number of relevant tokens than the hesitation rate. Actually, after a short training period, the algorithm becomes very accurate and stops mislabelling

emails, which explains the low error rate passed a few hundred emails. What happens when we add more tokens, especially with the first data set (10 boxes), is that the probability of belonging to a category becomes higher than 0.5 for several categories. This result ends up with more hesitation. There is no clear rule for choosing the number of relevant tokens, but we suggest keeping it to a small fraction of the total dictionary (5%).

6 K-Means Clustering

While Naive Bayes produces good results, we are also considering another approach, inspired by unsupervised learning, which may lead to a reduced performing time. Each mail category is represented by a a vector (centroid) in a m dimensional space, where m is the size of the dictionnary. m grows for each incoming email we add to the training set. We represent incoming emails by binary vectors (token's presence / absence). We use two different metrics to calculate the distance between a new mail and a centroid. The first one is the traditional L_2 norm, the second one is the scalar product of the new email vector and the centroid.

6.1 Algorithm

1. Label the incoming email by finding the "closest" centroid, based on the tokens currently contained in the dictionnary. The prediction is confident when the mail is really closer to a centroid than to the others, we use the ratio of the distances and some threshold value α to decide whether the mail will actually be classified or not.
2. Check the correctness of the labelling. In case of mislabelling, add an error and reassign correctly the new email.
3. Expand the token list and update the centroids (new tokens, new dimension) . The space dimension is increased by one for each new token.

This algorithm is a little faster than our Naive Bayes approach since for each new incoming email, we only have to update one centroid, and this update requires few operations.

6.2 Adding Relevance

As for Naive Bayes, we can only use the most significant tokens in our predictions to reduce the dimension across which we make a prediction : μ is the centroids matrix. Given a category i and a token j, we define :

$$r_{i,j} = \frac{\mu(i,j)}{\sum_{l \neq i} \mu(i,j)}$$

6.3 Results

6.3.1 Choice of Distance

The L_2 norm is the first natural choice for a distance but it gives poor results. The scalar product of the new email vector and a centroid is a projection rather than a distance because the closest you are to a centroid, the higher is the norm of the projection. However, the second approach gives better results both in terms of error and hesitation, and this is the measure used for plotting the curves presented below. We can note that using the scalar product makes the algorithm look a lot like Naive Bayes.

6.3.2 Analysis

The two curves show the Error-Hesitation curves for the two data sets when we vary the parameter α. A low error rate comes at the cost of a high hesitation rate and vice-versa. The algorithm produces satisfying results, especially for the second data set : 3% error for 10% hesitation when α is set "correctly". However we do not see how to provide an heuristic for the choice of α.

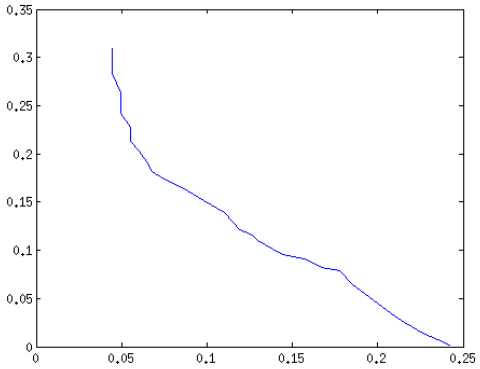

FIGURE 1 – Error Hesitation Curve for First Data Set

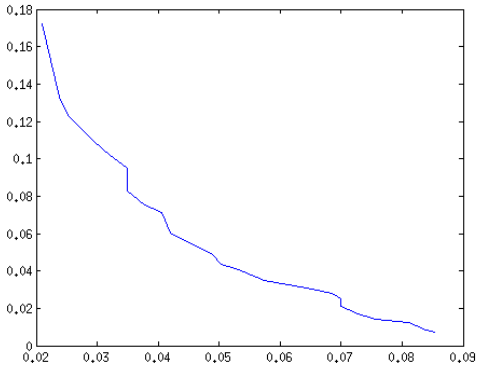

FIGURE 2 – Error Hesitation Curve for Second Data Set

6.4 Additional Remark : Giving more weight to recent emails

In some cases (especially for the second data set), for a given category, the lexical field of the emails can change with a new topic of discussion for instance... We thought this might impact the precision of the algorithm. Therefore, we modified the algorithm to give more weight (in the centroids) to the most recent emails in the category. This slightly lowered the hesitation rate, but we did not judge this improvement enough to justify the additional computing time.

7 Conclusion

We obtained very good performances with Naive Bayes. However, we could not complete our secondary objective, which was to devise an alternate algorithm, with similar (or slightly inferior) results than Naive Bayes, but with a lower running time. Our K-means adaptation does take less time to execute than our Naive Bayes implementation, but is not as accurate and requires a finer tuning of parameters. Actually, we realize now that an online machine learning algorithm will always require some "heavy" computation at each step, since the model needs to be updated for each new example. In the absence of a fast algorithm, one can always try to reduce the frequency of the updates, and see if the accuracy is significantly reduced.