Report: Kaggle competition

Nowadays, one can receive an aberrant number of emails of all kinds which can be confusing when checking one's inbox. To remedy this situation, we propose a predictive model in order to classify emails in four categories : update mails, social mails, personal mails and promotional mails. The proposed model is trained on a training dataset of around 25066 emails, each represented with 12 different attributes such as the number of images in the mail, Then the model is tested on a dataset of around 10475 emails, based on the F1-score. On the leaderboard, we are ranked 50 over 72 and our score is: 0.86226.

A. Features engineering

First, working with data requires cleaning it : dealing with missing, redundant or aberrant values and get rid of useless features.
Among problems we face, some values of certain features are missing, such as *org*, *tld*, *mail_type*. In order to deal with this issue, we fill the missing values with a 'NA' value that doesn't disturb the distribution. Furthermore, the *mail_type* feature has redundant values (difference in some characters), we combine those redundant values into one.
We estimate that *id* is not important to classify emails so we drop it.

Now, we can work on the features. To do so, we've tried many combinations of our features (quotient, product, sum, simple labelization of one feature, etc.) using intuition and then checking whether the empirical correlation with respect to "label" has improved, by displaying the correlation matrix. Our main work was :

1. Combining the *images* and *char_in_body* features we propose the "***frequency***" feature : $frequency = \frac{\#images^2}{\#char\ in\ body}$. This provides a more important empirical correlation (with respect to labels) : around 0.2, than the number of characters (around 0.04), our intuition behind this is that comparing the number of images to the text size provides more insight about the mail ;

2. Creating a new feature in order to take into account the rarity of some *org* and *tld* values since some organisations appear once in the data. These features were called *count_org* and *count_tld* and return the occurrence of each feature. We verified they are empirically way more correlated to the 'labels' by displaying the correlation matrix (available on the code as corr) ;

3. Create a new feature consisting of the date string's length because it turns out *length_date* is mildly correlated to the labels ;

4. Displaying a histogram (*Fig.1*) of the *ccs* (and *images*, *urls* too) feature shows us that some new categories can help classifying the emails. We divide values of the feature *css* to 3 subcategories: 0, [1,2] interval, 3+ range. Some labels only relate to some specific categories.

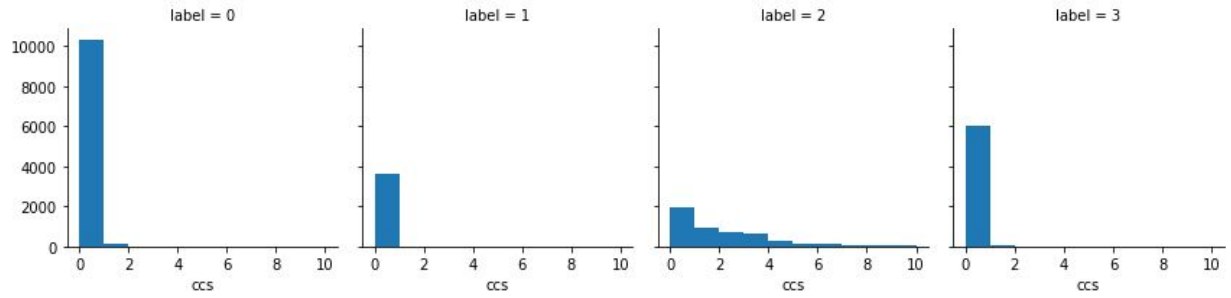

Fig.1 Distribution of number of ccs for each label

5. Convert the following features : *org*, *tld*, *mail_first* and *mail_second* with the one hot encoding method, which gives us a dataset of 560 features.

6. The *mail_type* feature contains 2 informations: the format used and the sub-type of this format. We extracted those 2 informations into 2 different features that we encoded later.

As we can see, some attributes were conserved others were modified, and some were even discarded: Id, date, org, tld, ccs, bcced, mail_type, images, urls, salutations, designations, chars_in_subject and chars_in_body. In addition to the new proposed features.
The next step is normalizing our improved dataset using a StandardScaler provided by the sklearn library.The normalization is necessary because the difference in scales of features can cause some problems for models such as SVM.

B. Model tuning and comparison

We have had to choose one classifier among the ones we know. So, we had to test different classifiers on the dataset and evaluate their performances with F1-score. Plus, all the classifiers have parameters to choose (For example, SVC needs one kernel to choose: rbf, poly, etc..).

Our approach is clear:

1. Do a cross validation of different models using a ShuffleSplit method, it is a randomized K-fold method and we choose K=10. That operation will give us the accuracy with the score method we choose (F1-score) and the fit time.

2. Apply the best model in term of accuracy to our test set. To improve the results, we do a hyperparameter search using the Grid Search method for the best models we found.

	Model	test_score_mean	fit_time_mean
0	LogisticRegression	0.893191	50.369799
1	RandomForestClassifier	0.936330	13.218765
2	AdaBoostClassifier	0.561835	23.950612
3	SVC	0.905878	94.762048
4	KNeighborsClassifier	0.923644	6.289141
5	XGBClassifier	0.893138	83.539692

Fig.2 Models comparison based on F1-score

3. The Random Forest Classifier proved to be the most efficient after the cross validation. In order to avoid overfitting, we tuned a parameter relative to the number of features that are chosen to grow each tree. A cross validation is used to tune this parameter, we chose the one minimizing sample prediction error. We choose a parameter range to search in: [number of nodes: 100, 200, 500, 800, 1100, criterions:gini,entropy] and we found the optimal hyperparameters which are number of nodes=800 and criterion=gini.

4. We also tested the SVM model because it may cause less over-fitting than RandomForestClassifier. Using Grid Search we found the hyperparameters C=0.1 with a rbf kernel to be the most optimal.

C. Additional ideas I have worked on

In the end, I have also tried to compute an efficient **Neural Network** with keras. Since, we have 4-class problem, the last layer has 4 nodes. To avoid any gradient evanescence problem, I have chosen 'relu' in all hidden layers and I use softmax in the last to have probabilities of labels in output. I have chosen to take N nodes in the first layer and N/2 in the second with N the number of features in the one hot encoded dataset.
We test several parameters with the following model: $batch\ size = 2^p$ and $epochs = 50.q$ for multiple (p, q) natural integers (based on many neural networks seen on internet) respectively from 5 to 10 and from 1 to 4. But for large values, there was clearly an overfitting, I could find within the lasting time, that the optimal couple was: $(5, 2)$.
On the test set for optimal couple, we had F1-score~0.85 (2nd best submission on Kaggle)

For the **date feature**, I understood late how important the data is. Since adding length of date has improved Random Forest and Neural Network, I tried to exploit it smartly such as taking the moment of the day (morning, middle of the day, night), moment of the week (working days, weekend), moment of the month (beginning, middle and end of the month). Unfortunately, I had issues with computing well times considering time zone and I couldn't implement errorless date reading.