Oscar MEURER, Javier GONZALEZ, Peter KESZTHELYI, Alix VERMEULEN
TheKNNBest

Kaggle Challenge Report

Feature engineering

Below, we note what each feature is intended to capture:

● **Date**: This is an important feature because it allows us to detect whether an email was sent at an abnormal time or not. We could consider abnormal to be between 2 to 5AM.

● **Org & tld**: These are both very important features because they state the source of the mail which can accurately reflect the type of mail it is.

● **Mail type**: This is the type of data included in the email (html / plain text / html with attachments, etc.) which could be an important indicator.

● **is.CCS**: the number of CCS is also important : for example, spams could be sent to only 1 person (no CCS) or sent to 30 people at the same time.

● **Images & url**: If there is an abnormal amount of images or urls attached to an email, this might raise multiple questions about the validity of the email.

● **Chars_in_subject & chars_in_body:** From the experience we can intuitively say that promotional emails tend to be long in body, however most of them are becoming more images nowadays, we still expect to get some differentiation between categories.

How did you come up with new features?

Remark: the encoding was not done for our final Deep Learning model used.

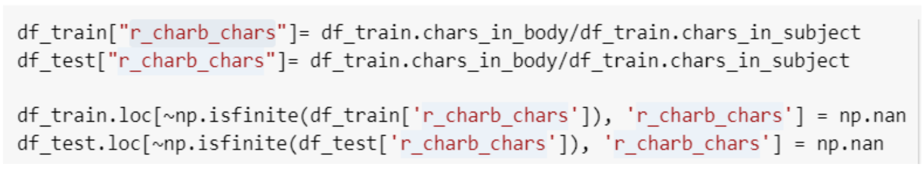
● **Dates**: We discarded the original 'date' column and added 6 new columns 'hour', 'day', 'week_day', 'week', 'month' and 'time_zone_number'. Even if they are ordinal features, we considered them as categorical, and one hot encoded them. We came up with this as we considered the mails from one source can be related to the day of the week, possibly the time zone difference with the actual could relate to spam, or also a close friend in another country.

● **Classic Encoding**: The 6 columns above and 'mail_type' were one-hot encoded.

● **Org And TLD**: As we wanted to keep the "energy" of those features, we computed One Hot Encoding on those columns (which gave us a really sparse matrix) and we performed dimensionality reduction with SVD (Singular value decomposition). We kept the 20 first components.

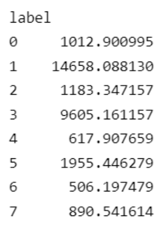
● **Quantify missing values in org & tld**: We added during (mode) imputation of those columns 2 new columns [org_imputed] and [tld_imputed] that have 1 if they have been filled in or not.

● **Relation chars_in_body/chars_in_subject:** Having a long subject but a short content could probably relate to some category, we created a new feature with such a relation:



```
df_train["r_charb_chars"]= df_train.chars_in_body/df_train.chars_in_subject
df_test["r_charb_chars"]= df_train.chars_in_body/df_train.chars_in_subject

df_train.loc[~np.isfinite(df_train['r_charb_chars']), 'r_charb_chars'] = np.nan
df_test.loc[~np.isfinite(df_test['r_charb_chars']), 'r_charb_chars'] = np.nan
```



```
label
0      1012.900995
1     14658.088130
2      1183.347157
3      9605.161157
4       617.907659
5      1955.446279
6       506.197479
7       890.541614
```

Here is the mean value per category of this new feature.

Did you discard other features?

For our XGBoost classifier after encoding, we decided to keep the 20 best features in our model, discarding all the other ones. We didn't have time to put this k-best feature selection.

Cluster	Columns	Analysis
Original Data	ccs, urls, salutation, designation	
Org and tld after SVD	0,1,2,5,9,10,11,13	These are the eigenvectors that keep the energy of org and tld.
Missing values	tld_imputed, org_imputed	The fact that it was missing was important.
Time Series	time_zone_num_-7.0 time_zone_num_0.0 time_zone_num_5.5	Time zones were indeed important.
Mail type	mail_type_multipart/alternative,mail_type_text/html	

We can see that a lot of features we created are important.

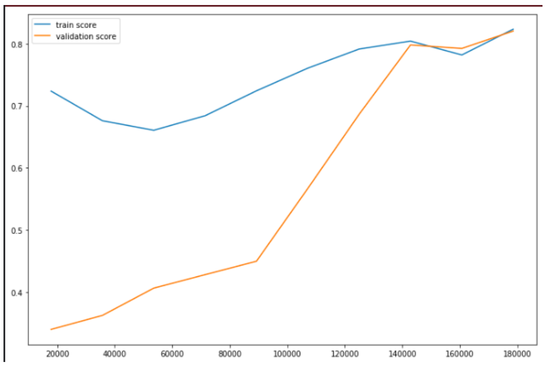
Model tuning and comparison

For this project, after the shared preprocessing phase, we assigned one type of model to every member: **KNN, Naive Bayes, Random Forest, SVM.** After seeing that only RandomForest was performing well, we decided to try **Deep Learning** models and **XGBoost**.

For every model, we first did a **RandomizedGridSearch** to have an idea of the best parameters, and then focused on a smaller range with **GridSearchCV**. First we put cv=5, but it was taking too long, so we reduced to 5. We also computed some learning curves so see if the model generalizes well.

KNN
We made the number of clusters k range from 2 to 18 and CV returned 18 clusters. We obtained an accuracy of 41% which is significantly lower than the other models.

Random Forest
On Kaggle the best result was 54%, and the score on the training set was 80%.
NB: as some labels were underrepresented, we tried Over-Sampling with ROS (that means that we added rows so that every label has the same number of instances). This gave this learning curve which is impressive, but didn't do any good results on the predictions.

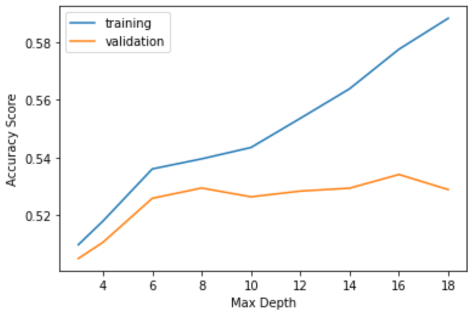


XGBoost

For XGBoost, the best performance obtained was 57% on the Kaggle test set. We used Grid Search Cross Validation and the accuracy score to choose the best model, whilst tuning the hyperparameters. Our optimal parameters were: learning_rate = 0.1, max_depth = 5, min_child_weight = 3 and n_estimators = 140.

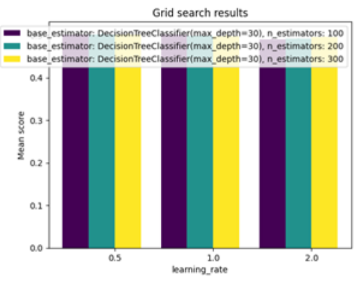
Decision Tree Classifier

Our model was not able to meaningfully capture the relationships in our dataset, possibly due to non-linear decision boundaries that cannot be created through this classifier. As seen below, the performance was subpar to our final classifier.

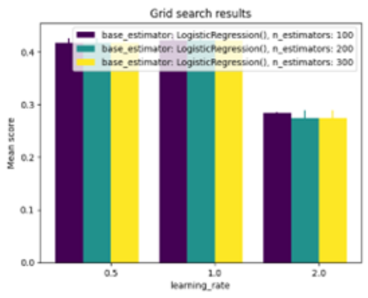


Adaboost:

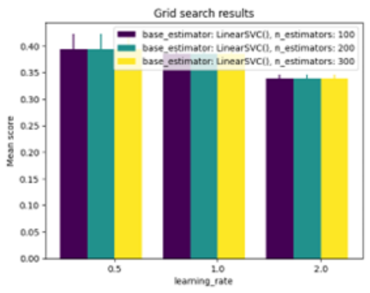
One of the trials was the use of boost technique adaboost with some weak classifiers. In this test, we used Decision Trees, Logistic Regression and Linear SVM.



Decision tree



Logistic Regression



LinearSVM with softmax

As a result, for the three models, the number of estimators in the adaboost did not improve the performance of the classification. It seems linear SVC is a robust classifier so each of the estimators will probably have the same score therefore in the end the classification will be the same no matter the number of estimators. From the results of this model test, it seems there is no kernel function appropriate to linearize the data to make the SVM algorithm perform better.

Tensorflow and gradient boosted decision trees

Finally our best model was a decision tree trained with the library keras from tensorflow. We obtained an accuracy of 58%. For this task, the big issue was the learning curve to use this kind of library and learning the concepts behind it. The data preprocessing to train this model was similar to the one used previously, except for the categorical string features which were passed directly to a tensor - this one created an intermediate layer from string data to a dictionary. The best parameters for the gradient boosted decision tree were:

No of trees	**Min Examples**	**Max Depth**	**Subsample**	**Sampling method**	**Validation ratio**
250	6	5	0,65	Random	0,1