FOUNDATION OF MACHINE LEARNING

Master in Data Sciences and Business Analytics

CENTRALESUPELEC

Assignment 2 - Kaggle Competition

Team name:
Group 007

Team members:
Bryan ATOK A KIKI
Sauvik CHATTERJEE
Alban VEAUTE
MIAO WANG

Feature Engineering

Preprocessing

Before starting feature engineering, we did following major steps to clean data:

1. Fill null values: since we noticed that both in *train_df* and in *test_df*, there are null values in `'tld'`, `'org'`, `'mail_type'` and `'chars_in_subject'`. For categorical features, we filled with `'none'` while for numerical features, we filled with its mean value.
2. Check duplicates: namely, they have the same values for all parameters but with different labels attached. 19626 rows from the original train set are duplicated twice, 395 rows are duplicated three times. We assume these duplicates happen either because we need more information to distinguish or because they are ambiguous to be categorized into one single class. One email can be both Personal and Updates. To take care of this, we implemented this trick.

	row_id	Feature 1	Feature 2	Feature 3	label
train_df	1	a	b	c	1
	2	a	b	c	2
	3	a	b	c	3
test_df	1	a	b	c	unknown

After transformation

	row_id	Other features	label_1	label_2	label_3	label
train_df	1	....	0	1	1	1
	2	.....	1	0	1	2
	3	.....	1	1	0	3
test_df	1	.....	1	1	1	unknown

0: because its actual label is 1, so we put 0 here
1: this row has possibility of being label 2&3, so we put 1 here
1: for the same record of test_df, we will put 1 in each dummy variable to tell our model this record can belong to all three labels

3. Clean outliers: To avoid being biased by outliers, for numerical features, we replaced those outside 95% quantiles with their respective 95% quantile.

Extracting more features:

Based on provided features, we found below information might be interesting to help us identify email classes:

1. Date time relevant information: `append_date_time()` & `append_date()`
   We assume some underlying patterns can be revealed by more detailed info from `'date'`, like day_of_week, day_of_month. For example, probably those promotion emails tend to be sent at weekends while forum emails from professional groups appear more on weekdays.
   `'date'` is originally a string, thus we used `date()` and `append_date_time()` functions to transform it into *datetime* format and then applied `append_date()`.
2. More binary features: `create_binary()`
   For `'ccs'`,`'images'`,`'urls'`,`'chars_in_subject'`, instead of using them in numericals, a binary column may better help recognize those special cases from others. For example, an email with 0 char in the subject is more likely to be spam. Thus, we added `"Has_cc"`, `"has_image"`, `"has_url"`,`"has_char_in_subject"`.
   Following what we have extracted from `'date'`, we have `'if_weekend'`
3. More candidate features: `extract_more()`
   Here, we took into account the interaction between parameters, and created more ratio features.

For example, we assume the ratio between `'chars_in_body'` and `'images'` may matter, because promotions (rich media) tend to be lower in this ratio while personal emails should score higher.
We have `'bodychar_image_ratio','image_subject_ratio''bodychar_url_ratio', 'body_subject_ratio', 'url_subject_ratio'`

Normalization

To remove the influence of different scales, we should normalize all numerical features. Plus, considering the existence of binary features, we'd better choose min-max normalization instead of `standscaler()` here to cast all numerical features into [0,1].

One-hot encoding on categorical features

Before applying one-hot encoding, we would like to give a closer look at `‘org’,’tld’`.
There are 1023 and 287 distinct values for `‘org’,’tld’`, while those with low frequency won't contribute much to predictions and for the sake of reducing dimensions, we labelled them with 'others' by `process_org_tld()`.
Meanwhile, we realized that there are some capitalization issues and spaces which lead to redundant values in `‘mail_type’`. We applied `lower()` and `strip()` to fix it.
When these prerequisite actions were done, one-hot encoding returned us 260 columns in total.

Feature selection

Before proceeding to model training, we tried to create more train sets with lower dimensions.

1. With chi2 criterion

Initially, we want to use variance threshold to filter out less significant features, however, in our case, one-hot encoded features should be much sparse, so nearly 0 variance is quite normal.
The chi-square test measures dependence between stochastic variables, so using this function "weeds out" the features that are the most likely to be independent of class and therefore irrelevant for classification. We narrowed down to 120 features by this step

2. RFE selection

Based on the result of chi2 criterion selection, we experiment with another prevalent feature selection tool called recursive feature elimination (RFE), the goal of which is to select features by recursively considering smaller and smaller sets of features. However, due to less optimal predictive performance, we skipped it in our final version.

Dimensionality reduction

To avoid overfitting and curse of dimensionality, we would like to further reduce the dimension of our input parameters. Here, due to the nature of our problem, LDA is a more appropriate tool to deal with supervised problems (We also considered PCA in the beginning).

As of now, we have 9 pairs of train sets and test sets prepared. And in the model tuning part, we would apply different algorithms in them and see which combination can lead us to the best result.
#without one-hot encoding and dimensionality reduction `train_norm`
# without any dimensionality reduction `Train_temp1`, with 260 features &`Train_temp2`, with 120 features
# with lda only `train_lda` # with pca only `train_pca`
# with lda and pca `train_lda_pca`

Model tuning and comparison

Prior to model implementation, we noticed that the train dataset has a class imbalance issue as labels 4/5/6 appear in small proportion compared to the unbalanced size of label 0 to 7. To prevent our models from over predicting majority classes, we tried both `SMOTE()` to synthesize new examples of the minority and `auto_class_weights` parameters.

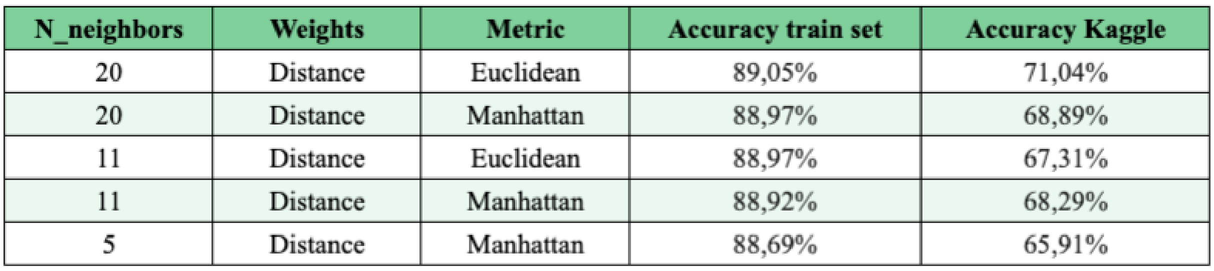
For the purpose of this assignment, we tried six different classification algorithms. For each classifier, we decided to use every pair of dataset to observe the relevance of the dimensionality reduction. We observed that our models yielded to better results with the LDA dataset.

Please find below a quick summary of the performances obtained with the six different models:

Classifier	KNN	Random Forest	Catboost	SVM	Naive Bayes	XGboost
Accuracy train set	89.05%	77.9%	87.7%	85.12%	79.62%	89.45%
Accuracy Kaggle	71.04%	62.1%	68.0%	61.39%	57.86%	69.14%

In terms of accuracy on Kaggle, we can see above that our results were significantly better with KNN, XGBoost and Catboost compared with SVM, Random Forest or Naive Bayes.

The KNN classifiers showed the best performance with 71.04% on the test dataset, therefore we will focus on this algorithm for our project.
You can see below the results for different parameters on the KNN models :



N_neighbors	Weights	Metric	Accuracy train set	Accuracy Kaggle
20	Distance	Euclidean	89,05%	71,04%
20	Distance	Manhattan	88,97%	68,89%
11	Distance	Euclidean	88,97%	67,31%
11	Distance	Manhattan	88,92%	68,29%
5	Distance	Manhattan	88,69%	65,91%

Regarding the hyperparameter tuning, we first used `GridSearchCV()` algorithm on our models to find the best combination of hyperparameters. Then, for KNN, XGBoost and Catboost we used trial and error to fine tune our classifiers.

Moreover, we also realized that the Catboost model includes a parameter allowing the algorithm to handle categorical features by one-hot encoding (default choice) and skip this pre-processing segment. We performed Catboost using `train_lda` and `train_norm`, we also tried to close the structural imbalance problem of our dataset both manually or with the `auto_class_weights` parameter but this did not lead to conclusive results.

In our search to find the best algorithm for this task, we noticed that the classification between mail and spam was the Naive Bayes even though the mail content was unavailable in our case. However, this algorithm performed poorly on the classification.