Kaggle Competition

Maxime et les garçons à table - Théau Pihouée, Alexis Carpier, Victor Perroux, Maxime Seince

Feburary 7, 2021

1 Feature Engineering

The goal of this project was to process the data given and build a multiclassifier model that offers the best performances possible. Here is how we proceeded.

1.1 Data Processing

The first step in all DataScience projects is to study the data set and highlight its strengths and weaknesses, which could be done using the .**describe**() function.

The data set contains about **40 thousands samples**, which is quite a lot. On top of that, we had already at our disposal a few relevant features. However, we quickly noticed that the data set was **unbalanced**. Indeed, when one looks at the labels, some are over represented (personal e.g.) and others are under represented (spam e.g.). In fact, each class had very different numbers of elements and we read that this unbalance made it difficult for algorithms to be performant.

We tried different techniques of **under** and **oversampling** with no significant results. For example, we decided to do the following : for classes over 5000 samples, we reduced them to 5000 by deleting rows. For classes under 500 samples, we oversampled to this number. We chose these numbers because a 1/10 ratio was acceptable (based on our research). In order to oversample, we did some random oversampling, i.e. taking random samples from the minority class and duplicate them into the training dataset in order to have more samples of this minority class (until this number reached 500). This was our first way to compensate manually the unbalance of the data, but it didn't perform well because it led to an important reduction of the training dataset size.

We didn't touch the numerical binary features, but we tried to **scale our data** to a [-1,1] range with a mean equal to zero and a standard deviation equal to 1 using Standard Scaler.

However, for most of the data, it didn't improve our result. Indeed, we think that the range of the features was quite important. If you contract the hour to a [-1, 1] array, it is not correlated to its original signification which is to give the time of the day between 0 to 84 600 seconds.

At first, we tried to scale our features in order to avoid a biased importance of some values due to their size, like we said previously (for instance with a great difference between the values of integers representing the days of the week and the values of integers of the column *chars in body*). After some tries of our models without scaling our features, we realised that it worked out way better with the Random Forest, which was our most performant model. Therefore we abandoned the scaling of the values.

1.2 Creation of New Features

For our first models, we used only the numerical data, but we quickly realised that if we wand to gain more accuracy, we would have to **encode more features**. We tried different types of encoders (OneHot, Ordinal and Get Dummies). Thus, we decided to look at mail type, org, tld and date features.

First, we exploited the *mail type* column as it was the easiest column to process according to us. We first tried to encode it using a OneHotEncoder which resulted in the creation of 14 features/columns (which

corresponds to the number of different categories of mail). We also tried to encode it using an Ordinal Encoder which attributed a number to each category, thus creating a single column. Finally, we tried to encode it using the dummies encoder and that method gave us the best results.

In fact, when looking closely at the different categories of mail types, we saw that some types were only different in writing but represented the same category (html and Html) so we decided to capitalize every label of mail_type. This reduced the number of categories and improved our score.

We then tried to incorporate categorical features in our training set. However, when looking at columns such as 'tld' and 'org', we saw that there were hundreds of different categories. To do it in an easy way as a first approach, we encoded them using the Ordinal Encoder but the predictions were less accurate than without these features. Indeed, we lost all interpretation of the data when encoding these two columns because there were two many categories. By paying a closer look to these columns, we selected only a few specific domains (the ones that were the most numerous) and we grouped them together. For example, we had a group *Education*, composed of *Coursera*, *Quora*, etc. We had another group called *Promotion* composed of *Netflix*, *Ebay*, etc. (These big companies sent a majority of promotion emails in our dataset).

Our final approach was to encode both tld and org using the dummies encoder, after having capitalized their labels. Then, we only kept the featurized labels that were both in the training and testing set, and we dropped the rest.

We extracted quite a lot of features from the date column. Among them are the **day** (Mon, Tue, etc.), the **month** (Sep, Oct, etc.), the **hour** in seconds, the **day date** and the **year**. We even created a feature that indicates if the mail was sent during the $\mathbf{week-end}$ or not, but this last one was redundant and very correlated with the 'day' feature, so we decided not to use it.

We didn't pay attention to the time zone, since we considered that the important thing was the hour of the sender, therefore adapted to its own time zone. It happened that the hour was a very important feature to add: our models performed way better, and we noticed an important reduction of the log-loss.

To select the most relevant features, we used the feature_importances_ function that give information about the importance of each features based on the training data set.

2 Model Tuning and Comparison

2.1 Transformation of the problem

Because we know how difficult a mutliclassification problem is, we had the idea to **transform the problem**.

Our first idea was to identify each **unique combination of labels** and to transform it into its own class. For example, we identified all the mails labelled "personal" and "*updates*" and labelled them as "*personal and updates*". This way, all the samples belonged to only one class.

However, we quickly realized that this method was going to be problematic. First of all, some categories were **under represented** compared to others that were predominant (3 samples vs. 9 thousands for the biggest differences), and on top of that, these new categories we created were **not exclusive**. For example, the class "*Personal and Spam and Travel*" could not be in the training data set, but could be in the test data set without any way to recognize it as the model was had never seen it before.

We then tried to simplify the problem into **8 different single classification problems**. For each samples, the goal was to determine, if whether or not it was going to be labelled 0 or 1 for each labels, and then combine the results to come back to the original problem. This idea was more efficient and gave us a few good results using logistic regression, even though it was not our most efficient model.

2.2 The different models we tried

Logistic Regression

We started with a logistic regression applied to the multiclass data that we had obtained after processing the data. We obtained mediocre results.

Then we made some researches about methods to handle imbalanced data and it appeared that bagging and random forests were algorithms that fitted pretty well to this type of data. So we implemented a random forest algorithm.

Adaboost and XGBoost

We tried two boosting prediction models on our datasets with a multiclass prediction for the 8 possible outputs. Therefore each time we applied a prediction model on each and every one of the outputs to determine the probability of each mail being a social for example. The metrics weren't convincing compared to our previous results and even trying to play on our parameters we weren't able to join the results of our other more efficient models.

MLP

We tried to implement a MultiLayerPerceptron, without success. We used standardized data. It was hard to determine the optimal number of neurons per layers and the total number of layers as there doesn't seem to be a theory behind this optimization. However, after trial and error, we managed to find the best architecture for our network. But it performed badly, probably due to the fact that our dataset was unbalanced.

kNN

We implemented a kNN model for a k in a range of [1, 15]. The model did not perform well compared to the others, certainly because we had less time to work on it.

Random Forest

We implemented our random forest algorithm with the following elements. :

- Grid Search to find the best hyperparameters: tuning the number of trees, the minimum samples of split, the maximum number of branches per tree.

- We added the class weight attribute to show our algorithm how imbalanced our data is. Actually, we read that a simple technique for modifying a decision tree for imbalanced classification was to change the weight that each class has when calculating the impurity score of a chosen split point. Since the RF classifier would tend to be biased towards the majority class, we should have a heavier penalty when misclassifying the minority class. We then set a *class weight* argument. However, it ended lowering the performance of our model but we were not able to identify why.

2.3 Tuning and Improving of our best model

In order to improve our best model, we worked on two aspects of our model the parameters and the data set with feature engineering.

Thanks to Scikit-Learn tools we were able to do model hyperparameter tuning. Our goal was to estimate those parameters using the GridSearchCV function therefore we estimated "by hand" the range of those parameters on which we were going to play. Those parameters were 'n estimators', 'min samples split' and 'max depth'. The GridSearchCV function gave us the best parameters for our random forest model, the parameters that we therefore use for our final model and our final prediction.

For each model, we prevent overfitting by making sure we computed cross validation on our different models.

On top of that, a lot of our work was centered around the feature engineering. Indeed, to improve our model we relied a lot on the extraction of relevant features that would bring additional information on how to

classify the test samples. To select the best features, we used the .feature_importances_ function mentioned above.

Remark :

- We made a few late submissions to the Kaggle that were not taken into account in the final leader board. Indeed, we believe we lost ourselves in the feature engineering by trying to do a lot of complex things and by losing sight of what could have been done to be truly efficient. For example, rather than encoding the date, the month or the year which was quite basic, we directly focus ourselves on the fact that the data set was unbalanced. This project taught us how important it was to work on the data and exploit it to get as many relevant features as possible.