Report on ELTA project

Table of Contents

SECTION 1: PROBLEM DEFINITION

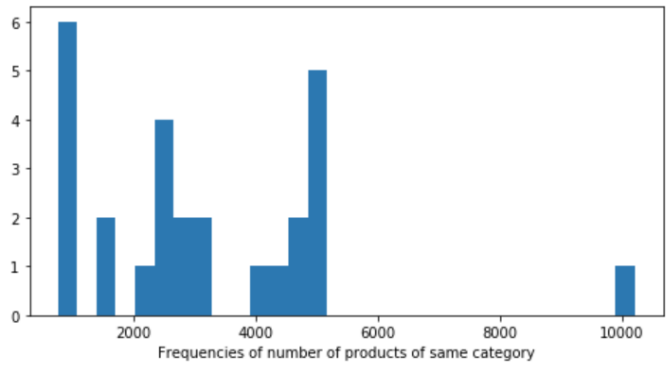
This project is part of a school project, we participate to the Rakuten challenge. This challenge focuses on the topic of large-scale product type code multimodal (text and image) classification where the goal is to predict each product's type code as defined in the catalog of Rakuten France.

SECTION 2: DATASET DESCRIPTION

Our raw dataset is composed of 2 files, one is called the X_train.csv which contains the designation, description, product id and the image id, this is will be taken as the input of the model. Another file is called Y_train.csv which contains the categories of the different products. The data in 2 files are entered in a row-row corresponding way, for example, the first row of the Y_train.csv is the product type code for the products described by the first row of X_train.csv. Here is some example data from X_train.csv and Y_train.csv.

designation	description	Product id	Image id	Product type code
'Grand Stylet Ergonomique Bleu Gamepad Nintendo Wii U - Speedlink Pilot Style'	'PILOT STYLE Touch Pen de marque Speedlink est 1 stylet ergonomique pour GamePad Nintendo Wii U. Pour un confort optimal et une précision maximale sur le GamePad de la Wii U	201115110	938777978	50

In this dataset we have 84916 products of 27 different categories, and the distribution of number of products of same category are highly unbalanced, here is the histogram of frequencies of number of products of same category:



We see that for most categories, we have between 2000-4000 products, while for certain categories, we have far too much (about 10000) or little (about 800) data.

A detailed description of the data preprocessing is given in the annexes. It gives good insights on the data we used to train our models.

SECTION 3: MODELS USED

Random Forest

Random forests are a modification of bootstrap aggregation that builds several uncorrelated trees, and then averages them. Trees are suitable candidates for bagging, as they are able to capture complex structures in the data, and if grown sufficiently deep, have comparatively low bias.
Since the trees generated in bagging are identically distributed, the expectation of an average of B such trees are the same as the expectation of any one of them. The idea in random forests is to reduce the variance by decreasing the correlation between the trees, without increasing the variance too much. This is done in the tree-growing process by randomly selecting the input variables.

Logistic Regression

A classic machine learning model, one of the first to use to get a baseline.

Adaboost classifier

AdaBoost, short for "Adaptive Boosting", focuses on classification problems, aiming to convert a set of weak classifiers into a strong one. The final equation for classification can be represented as

$$F(x)=\text{sign}\{\sum_{m=1}^{M} \alpha_m f_m(x)\}$$

where f_m stands for the m-th weak classifier and α_m is the corresponding weight.
The algorithm is formally described below:

1. Initialize the observation weights, $w_i=1/N$; i=1,2,3,…,N
2. For m=1 to M:
   a) Fit a classifier $G_m(x)$ to the training data using weights w_i
   b) Compute $err_m=\frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
   c) Compute $\alpha_m=\log((1- err_m)/ err_m)$
   d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, i=1,2,3,…,N.
3. Output $G(x)=\text{sign}[\sum_{m=1}^{M} \alpha_m G_m(x)]$.

Multinomial Naïve Bayes

Naïve Bayes is a classification technique based on Bayes' Theorem which works under the assumption that each of the features are pairwise independent.

Gradient boosting

Gradient Boosting, like Adaboost, converts a set of weak classifiers into a small one. The difference is how the two algorithms overcome the shortcomings of the weak learners. GBM generates learners at the time of the learning process. It calculates the loss (difference between actual & predicted values) of the first learners & then builds a second learner to forecast the loss after the first step. The process continues until a certain threshold is reached.

XGBoost

XGBOOST is an ensemble tree method that applies the principle of boosting weak learners using the gradient descent architecture.

The main reasons for the effectiveness of XGBoost are as follows:

- Parallelization: XGBoost handles sequential tree building using parallelized implementation, which is possible due to the interchangeable nature of loops used for building base learners. The outer loop enumerates the leaf nodes of a tree, while the features are calculated by the inner loop.
- Tree Pruning: XGBoost uses maximum depth parameter and starts pruning trees backward. This approach highly improves computational performance.
- Hardware Optimization: XGBoost makes efficient use of hardware resources cache awareness, allocating internal buffers in each thread to store gradient statistics.
- Regularization: Penalizes complex models through Ridge & Lasso regularization to prevent overfitting.
- Sparsity Awareness: XGBoost allows sparse features for inputs by learning best missing value depending on training loss and handles different types of sparsity patterns in the data.
- Weighted Quantile Sketch: XGBoost uses weighted quantile sketch for finding optimal split points.

SECTION 4: EXPERIMENTAL STRATEGY

Splitting the dataset in train/validation/test

In order to train the models, we splat the data into a training, validation and test set. The training set is used to train the model, and the validation set is here to tune the hyperparameters of the model. The test set is never seen by the model and is just here to have an idea of the generalization capability of the model.

Using k-fold cross validation

In order to efficiently train the hyperparameters, we use the k-fold cross validation strategy. It consists in splitting in k parts the dataset and using k-1 parts to train and 1 to validate. Thanks to this we can fine-tune the parameters and help the model to be robust.

Using Grid search

Another way to find the good hyperparameters is to combine the previous methods with the grid search method. For each model we tried different ranges of values for the most important hyperparameters. However, it takes a lot of time to try all the possible parameters, it took us a full day to find relatively good hyperparameters.

For trees boosting algorithms we decided to put a high number of classifiers since it was an important factor, the models were still generalizing well thanks to the k-fold cross validation.

SECTION 5: COMPARISON AND ANALYSIS OF RESULTS

Descriptions of the models used

Here is the description of the hyperparameters of our models

- **Random Forest:** We used 1000 estimators without limiting the depth and the min sample split value was set to 2.
- **Logistic regression:** We use the l2 regularization function and a regularization parameter C of 2, the regularization function is necessary to get a model able to generalize.
- **Multinomial Naive Bayes:** We use the regularization parameter alpha with a value of 0.1.
- **Gradient Boosting classifier:** We used 1000 classifiers, a learning rate of 0.2 and a max depth for the trees of 3.
- **Adaboost classifier:** Using a base estimator with a depth of 2 instead of 1 was helping a lot, it seems that using a too simple base classifier is decreasing the score because the task is a more complex. We used also a big number of estimators in order to compensate the simple structure of the base classifier, the value we kept is of 3000 estimators which is quite big compared to the number of labels to predict, yet, this is not making the model to overfit too much thanks to our regularization strategy and the amount of data available.
- **XGBoost:** We kept the default parameters for this model because the training time was very long. The model had a result close to 0.7 of weighted F1 score without tuning the hyperparameters. It could have been interesting to try other values to see the influence.

We got those hyperparameters after training the gridsearch wrapper of scikit-learn with a 5-fold cross-validation strategy.

Scores and performance

	Weighted F1 score	Accuracy	Training time
Random Forest	0.780	0.779	1489.3 seconds
Logistic regression	**0.795**	**0.796**	13.6 seconds
Multinomial Naive Bayes	0.752	0.758	0.1 seconds
Gradient Boosting	0.745	0.746	4440.8 seconds
Adaboost classifier	0.429	0.419	360.5 seconds
XGBoost	0.709	0.702	214.8 seconds

We use big parameters for some models, that is why the training is long.

SECTION 6 : CONCLUSION

Team members contribution

Chu Haotian:

- Data preprocessing and cleaning
- Creating bag of word and TF-IDF vectors
- Stats on the dataset
- Section 1-2 of the report and annexes (detailed explanation on preprocessing)

Debdeep Roy

- Code to train the different models without hyperparameter tuning
- Section 3 of the report

Victor Paltz

- Gathering the code in a Github + cleaning and structuring
- Documentation and final script
- Hyper-parameter finetuning of the models
- Section 4-5-6 of the report

Conclusion

Most of the models have a weighted F1-scores which between 0.75 and 0.81, these scores are close to the baseline model F1 score, which is 0.8113. The best score is obtained with the logistic regression, it might be because it is well adapted to use it after a TF-IDF.

These scores are acceptable, but it could be possible to increase these scores by

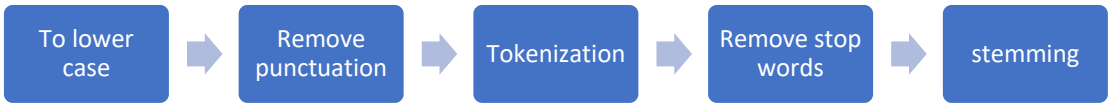
- Spending more time to fine-tune the hyper-parameters
- Embedding the sentences with another or additional method than TF-IDF (Glove embedding, word2vec, …)
- Using the image information (using the description vector generated by a standard deep-learning model as an input of our machine learning models).

Annexes

SECTION 2 BIS: DATA PREPROCESSING

To start with, our model concentrates on predicting product type code using their designation, so first step of data preprocessing will be deleting useless rows: description and the product id. We leave the image id in case that we want to use images for further improvement.

The designation is pure text data, and to preform classification on text data, the preprocessing on text is necessary. Here is our preprocessing pipeline:



We take the following designation as example:

'Modèle De Voiture 4pcs Alliage Métallique 1.9in Jante De Roue Pour 1/10 Traxxas Trx4 Hsp Scx10 D90 Rc Voiture'

To lower case

'modèle de voiture 4pcs alliage métallique 1.9in jante de roue pour 1/10 traxxas trx4 hsp scx10 d90 rc voiture'

Remove punctuation
Punctuation would introduce problems when tokenizing, so we preform the remove_punctuation befor the tokenization, here is the result.
'modèle de voiture 4pcs alliage métallique 19in jante de roue pour 110 traxxas trx4 hsp scx10 d90 rc voiture'
We see that the points and slash are well removed.

Tokenization
Tokenization transform the sentence to be a list of words, to do this we have adopted the regex tokenizer proposed by the nltk libarary, and the regex rule is *r'[a-z]\w+'* which means all words that start with lower case alphabet, this is for filtering words start with numbers, here is the result:
['modèle', 'de', 'voiture', 'pcs', 'alliage', 'métallique', 'in', 'jante', 'de', 'roue', 'pour', 'traxxas', 'trx4', 'hsp', 'scx10', 'd90', 'rc', 'voiture']

Remove stop words
This step is to remove words which are 'useless' for classification, like 'de' 'pour' of French, because these words doesn't provide any information about the object. To remove the stop words, we've adopted the stopwords corpus of nltk, and here is the result:
['modèle', 'voiture', 'pcs', 'alliage', 'métallique', 'in', 'jante', 'roue', 'traxxas', 'trx4', 'hsp', 'scx10', 'd90', 'rc', 'voiture']

Stemming

Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. This can reduce the number of different words in the dataset which will be the input dimension of the model.
To perform stemming, we've applied the PorterStemmer proposed by nltk library, here is the result:
modèl voitur pc alliag métalliqu in jant roue traxxa trx4 hsp scx10 d90 rc voitur
We see that the words in the sentence are well stemmed.

Well with this pipeline, there are about 68000 different words for the whole corpus of designation, this is too big to be the input dimension of a 27-class classification model.
To reduce this word corpus size, we've adopted 2 techniques:

Replace numbers in sentence by space
As a fact, numbers in the designation doesn't provide too much information for the classification task, like the 10 in scx10 for the example used previously. So we've considered to replace numbers in sentence by space, and the space will be deleted in the tokenization step, here is the result:
modèl voitur pc alliag métalliqu in jant roue traxxa trx hsp scx rc voitur
With this technique, the word corpus size is reduced to 57000, still too large.

Remove infrequent words
After inspecting the word corpus, most of them is of extremely low frequencies, also these words may provide information for the classification, model with input size too large won't work too well (overfitting, slow training and testing). So we've chose to remove all word that appears less than a threshold, like 10. In the result, the size of word corpus for all products is reduced to 7400, which is appropriate for the classification task.

Others
Since the preprocessing would delete words in the sentence, there would be some designations which become empty string, in this case we just delete these rows of empty string and the according rows in the Y_train.csv. As summary, 485 among the 84916 (about 0.6%) rows are deleted due to the preprocessing which is an acceptable result.