# Ensemble learning report - Text Classification with Rakuten France Product Data

Quentin Nativel[1]
quentin.nativel@student.ecp.fr

Sunjidmaa Shagdarsuren[2]
sunjidmaa.shagdarsuren@student-cs.fr

Marine Sobas[1]
marine.sobas@student.ecp.fr

[1]Msc in AI - OSY
[2]DSBA

March 26, 2020

## 1 Problem definition

### 1.1 Introduction

In this paper, we aim at classifying a product from Rakuten France product database into categories of product type. To do so, we use the product designation which is the name given to the product, along of a couple of words and often very descriptive.

As we only have the id of the category to which each product belong, we don't know what each category represents. Although it could be probably deduce from the data, the actual label has little interest for us. As a matter of fact, the underlying goal of this problem is to automatically predict the category of a product by using machine learning techniques.

To achieve this multi label classification, we test several methods of ensemble learning and data processing methods and we conclude by comparing these methods.

### 1.2 Notations

Given n, the number of samples, let's note $(x_1, x_2, .., x_n)$, the input data points and $(y_1, y_2, ..y_i, .., y_n)$ the labels that we want to predict. $x_i$ is the tfidf vector corresponding to the designation field after tokenization. The vocabulary size is 63,795, which explains why $x_i$'s size is a 63,795. A classifier is a function $f_\theta : X \rightarrow Y$ which given an input vector of size 63,795 outputs a label for it.

The metric used to compare the models is the weighted F1 score.

## 2 Dataset description

The dataset counts 84,916 points listing products from Rakuten France's catalogue. Only the field "designation" and "product type code" are considered in the modelling step. The designation field consists of strings which summarize the product's characteristics. In average, each designation counts 89 tokens, mostly French words. The designations were represented by TFIDF vectors. The size of the vocabulary is 63,795.

There are 27 different product type codes, the classes that we want to predict. Their distributions are imbalanced as the figure 1 illustrates.

For instance, class 2,583 represents 12% of the dataset, while five classes account for less than 1% of the data. More worringly though, the boxplot of the classes' frequencies, plotted in figure 2 highlights a dramatic imbalance.

There are a few classes which are overrepresented. That is why an oversampling technique was sometimes applied, namely SMOTE, synthetic minority oversampling. After this transformation, the classes are evenly distributed, respectively accounting for 3,7% of the dataset.

The analysis of the data mostly reveal a major imbalance. How does it impact on the predictive models?

**Histogram of the classes ”product type codes”**

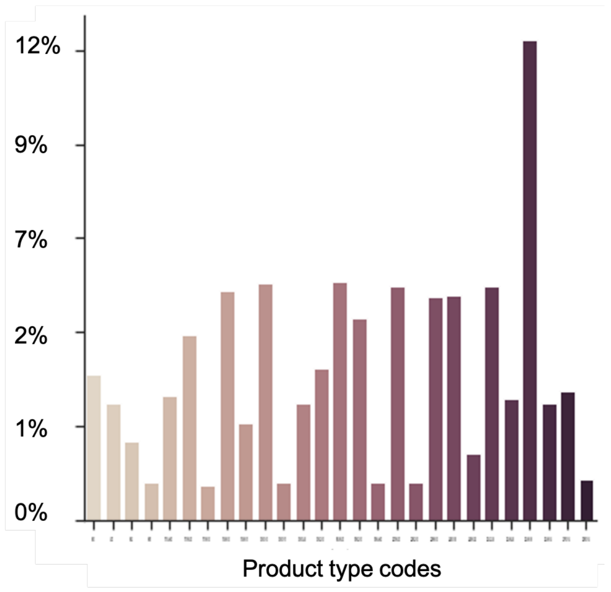

Figure 1: The distribution of the classes points out a significant asymmetry in the dataset. This may mislead the modelling step. The x-axis represents the product type codes, the y-axis indicate the frequencies.

**Boxplot of the classes’ frequencies**

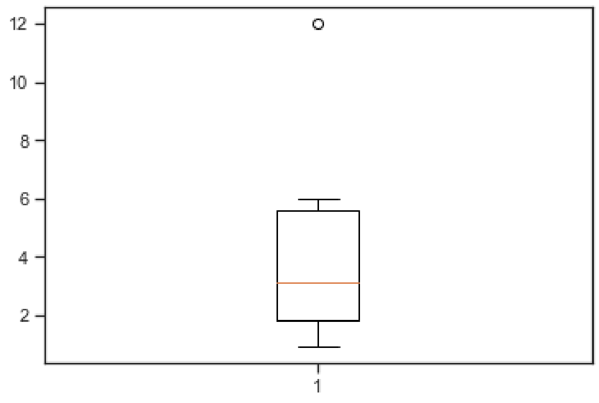

Figure 2: The distribution of the classes points out a significant asymmetry in the dataset. This may mislead the modelling step.

## 3 Models used

### 3.1 Baseline : a single learner

**Decision tree** Decision tree algorithms are weak learners and often referred to as Classification and Regression Tree (CART) models. This model uses recursive binary partition and choose split based on the loss function with the minimum loss. But we have to know when to stop, otherwise, we will overfit. Stopping criteria include max depth - the maximum number of length from root to the farthest leaf; minimum samples leaf - number of samples required for each leaf etc. We can further increase the performance of the model using pruning techniques, which is to eliminate some unnecessary branches. We only used the base decision tree model here. Hyperparameter tuning has been done in the random forest part.

### 3.2 Bagging

Bagging (Bootstrap Aggregating) is an aggregation of weak learners with bootstrap sampling. If it's classification, we can predict the class based on the majority vote of trees. If it's regression, the prediction will be the average of decision trees' results. We used the previous base decision tree in the bagging, with max samples of 0.3, which means 30 percent of the samples should be used for each tree. The most famous bagging algorithm is the random forest model.

**Random forest** The main idea of the random forest model is the majority vote from uncorrelated trees are usually perform better than a single tree. But because we are using bootstrap sampling, trees are correlated with each other. To decrease correlation, random forest uses random variables for each tree.

**Hyperparameters of random forest**

- **Bootstrap** : If the sampling should be with replacement
- **Number of estimators** : the number of trees in the forest
- **Minimum sample split** : the minimum number of samples needed to split an internal node.
- **Maximum depth** : the maximal depth of a tree
- **Maximum features** : the number of features that are considered on a per-split level

### 3.3 Boosting

Bagging and boosting both combine weak learners trained on randomly-picked samples to get more significant predictions. While for bagging, learners are independantly trained, in the case of boosting, each decision tree learns from the errors of the previous models. The latter results in a model with a lower bias, since training focuses on misclassified data. Yet, it may also infer too strong hypothesis, which might cause overfitting. We doomed that boosting was relevant to increase the score of the baseline model, whose F1 score is only 0.69. Two boosting algorithms were considered, gradient boosting and Adaboost. For both boosting methods, let's note $H_t(x)$, the weighted sum of each learner, given t, the number of iterations. An iteration occurs when a new weak learner is added to the model. After t-1 iteration, the predicted value is equal to :

$$H_{t-1}(x) = \sum_{\tau=1}^{t-1} \alpha_\tau h_\tau(x)$$

**Gradient boosting classifier** For gradient boosting classifier, every weak learner is weighted by a constant coefficient $\alpha$ so $\forall \tau, \alpha = \alpha_\tau$. $\alpha$ is equal to a small value such as 0.1 to prevent overfitting. We initiate the tree with a leaf which averages the likelihood to belong to a class on the entire dataset. Then, for each iteration, the goal is to find $h_\tau$, a weak learner, which minimizes the loss between the observations and the predictions :

$$h_\tau = \arg\min_h \frac{1}{n} \sum_{i=1}^{n} L(y_i, H_{t-1}(x_i) + \alpha h(x_i). = \arg\min_h J(H, h)$$

For gradient boosting classifier, we seek for the local best learner which minimizes the loss vector J. So we use the functionnal gradient descent and compute -g, the step direction towards local optimum.

$$-g = -\nabla_H J(H)$$

We then compute h which best approximates -g :

$$h_\tau = \arg\min_h g \cdot h$$

.

The key hyperparameters are as follow :

**Boosting-based hyperparameters**

- **$\alpha$, the learning rate** : it represents the contribution of each tree. It is therefore a lever against overfitting. As a matter of fact, if a model contributes too much, it may become too specific. Yet a lower learning rate increases the computational cost.
- **Number of estimators** : the number of base learners to compute, which amounts to the number of iterations. A too high value may foster overfitting. This risk will depend on the contribution of each tree, that is adjusted thanks to the learning rate. That is why, these two hyperparameters should be conjointly tuned.
- **Subsample** : the fraction of samples which are used to fit each tree. This helps prevent overfitting.

**Tree-specific hyperparameters**

- **Minimum sample leaf** : Contrary to Adaboost, the decisions trees are not stumps but have several branches and leafs. Yet, their size are fixed by this hyperparameter which corresponds to minimum samples required in a leaf.
- **Minimum sample split** : the minimum number of samples needed to split an internal node.
- **Maximum depth** : the maximal depth of a tree

As these hyperparameters are related to the definition of the weak learners, it seems more consistent to tune them simultaneously.

**Adaboost** As the gradient boost, Adaboost uses a gradient descent method by choosing the next weak learner $h_t$ the minimize the error over the training set $E = \sum E[H_{t-1}(x_i) + \alpha_t h_t(x)]$. $\alpha_t$ the coefficient is computed with the error as well.

**Hyperparameters**

- **Base Learner** : Usually the weak learners for Adaboost are simply splits on the data, so decision trees of depth 1, but it gives poor result on Rakuten data because the input space has a lot of dimensions, so we use instead deeper trees. The decision tree classifier that we fitted on the data has a depth of around 3000, so we decided to test smaller values for the depth of the weak learners of Adaboost.
- **Number of estimators** : This is the number of estimators in our final Adaboost classifier, so it's also the number of time the Adaboost algorithm runs, the higher it is the better are the results but also the slower is the training.
- **Learning rate** : The learning rate shrinks the contribution of each new weak learner and therefore it has an impact on the convergence speed.

## 4 Experimental strategy

Regarding the experimental protocol, we performed different evaluations. We calculated the accuracies of the model. More importantly however, the weighted F1 score was computed. As classes are not evenly distributed within the data, both the precisions and the recalls are needed here. For instance, the class 2583 represents 12% of the data while 1180 accounts for 0.9%. To compare models, the choice of hyperparameters is significant. Thus, we implemented grid and random search based on 3-fold cross validations.

### 4.1 Training-validation split

A first evaluation relies on splitting the data into a train and a test set. The test set accounts for 33% of the dataset. Once the model has fitted the train set, relevant metrics are computed on the test set, namely the weighted F1 score and the accuracy.

## 4.2 3-fold cross validation

A more robust evaluation was then deployed using k-fold cross-validation. The data were split into 3 samples, for the dataset was quite large and some models had a training time greater than 3 hours. One of these three samples is randomly picked as a validation set, while the other two constitute the train set. The model is then trained and evaluated based on its weighted f1 score. This process is performed for each sample and thus returns three scores which assess the models.

## 4.3 Grid search

To find the best hyperparameters for the different models, we used a grid search which try to optimize the weighted f1 score over the range given for each parameter. We used the implementation from sklearn. The process is very on our computers, which put some restrictions over the hyperparameters we tried.

For the Adaboost classifier, we had 2 hyperparameters, the learning rate in $[10^{-4} : 1]$ and the number of estimators in $[20 : 200]$. Due to computational power we couldn't use this method to find the optimal depth of the base estimator.

Due to computational power, we used 2 hyperparameters for Random forest, number of trees $[100, 120, 140, 160, 180, 200]$, minimum samples split with $[8, 10, 12]$.

## 4.4 Random search

As Bergstra[1] showed, a random search performed both empirically and theoretically better than grid search for hyper-parameter optimization. We implemented it for tuning the gradient boosting classifier, for its training was slow and its hyperparameters dependant. We first performed it on boosting hyperparameters assuming that the learning rate alpha followed a uniform distribution $U([0.01, 0.199])$ and the number of estimators' distribution was discrete and uniform : $U([\![100 ; 200]\!])$. Five combinations were then compared using 3 fold cross-validation relying on the F1 weighted score. The same process was repeated for the minimum samples leaf, the minimum sample split and the subsample hyperparameters, leading to a 12% increase in the weighted F1 score (from 62% to 74%).

## 5 Comparison and analysis of the results

We have used a decision tree, bagging classifier, random forest, gradient boosting and AdaBoost to predict the correct product type for each designation. Hyperparameter tuning with grid-search and the randomized search takes a lot of time to train. Based on the below results gathered in table 1, random forest model performs the best for the classification in our dataset.

| | **Smote** | **Decision tree** | **Bagging** | **Random forest** | **Gradient boosting** | **Adaboost** |
|---|---|---|---|---|---|---|
| **Training validation split - Scores computed on a test set (33% of the dataset)** | | | | | | |
| **F1 score** | False | 0,699 | 0,699 | 0,784 | 0,738 | 0,731 |
| **Accuracy** | False | 0,699 | 0,699 | 0,788 | 0,733 | 0,722 |
| **F1 score** | True | 0,708 | 0,707 | 0,787 | 0,763 | 0,728 |
| **Accuracy** | True | 0,705 | 0,705 | 0,789 | 0,754 | 0,719 |
| **3-fold cross validation** | | | | | | |
| **F1 score** | False | 0,692 | 0,688 | 0,785 | 0,740 | |
| | False | 0,688 | 0,686 | 0,785 | 0,734 | |
| | False | 0,679 | 0,682 | 0,784 | 0,733 | |

Table 1: The results show that a random forest is more relevant to predict the product categories. Data upsampling has also a positive impact on the accuracy and the F1 score for most of the models.

**Comparison between the models** It appears that random forest returns the best results, namely a 0.79 weighted F1 score. Gradient boosting and adaboost with SMOTE also return satisfying scores, respectively, 0.76 and 0.72. We observed that the major challenge in this problem is to prevent the models from overfitting so reducing the variance. Bagging is more effective than boosting and random forest has better scores because averaging over several weak learners lower the variance, while boosting may maintain a certain level of overfitting. Gradient boosting trees perform better than adaboost because data include 63,795 fields. While gradient boosting use trees of fixed depth, able to learn from several features, an adaboost model only generate stumps. That is why, adaboost models need extra estimators to use all the features, which may lead to overfitting. So adaboost is not perfectly fitted to solve this problem. That's also why in order to have good results with Adaboost we decided to use deeper trees. The gab between the test scores and the validation scores is higher for bagging than more boosting techniques. Thus, it is now clear than boosting may trigger more overfitting.

**Comparison between normal samples and upsampled training data** Upsampling techniques had a positive impact on the scores, whatever the model may be. It especially improved the scores of the boosting models. It is consistent because these models are also more likely to overfit, so SMOTE help them making up for the imbalance.

**Hyperparameters** To previous results were obtained using the hyperparameters written in the table 2. Random forest and gradient boosting trees have a similar number of estimators. Through grid search and random search, we tried to obtain the best F1 scores for each model to compared them on the same basis.

| | **Decision tree** | **Bagging** | **Random forest** | **Gradient boosting** | **Adaboost** |
|---|---|---|---|---|---|
| **Method** | Default | Default | Gridsearch | Randomsearch | Gridsearch |
| **Hyperparameters** | | | | | |
| **n_estimator** | | 10 | 160 | 162 | 100 |
| **max depth** | None | | | 15 | 300 |
| **max_samples** | | 1 | | | |
| **min_samples split** | 2 | | 10 | 341 | |
| **max features** | None | 1 | 2 | None | |
| **learning rate** | | | | 0,08 | 0,1 |
| **min samples leaf** | 1 | | | 31 | |
| **subsample** | | | | 0,87 | |

Table 2: The table give the hyperparameters which were used to obtain the previous scores

## 6 Conclusion

In this case, predicting product type from the data will save time and money for the business. However we only used the designation field texts to predict the product type. Our dataset has 85k samples and 27 different type of product classes and then we transformed texts into TFIDF vectors. Using these vectors we did ensemble learning techniques and got the best result of 0.79 weighted F1 score with random forest model. There are limitations of computational power and time in our algorithms. If we want to further increase classification performance, we should include other variables, use hyperparameter tuning on more parameters or use other preprocessing techniques.

In this assignment, our group members trained each models and contributed in the process. Each contribution is described in table 3.

| Team member | Quentin Nativel | Sunjidmaa Shagdarsuren | Marine Sobas |
|---|---|---|---|
| Models | Grid search on adaboost | Grid search on random forest | Random search on gradient boosting tree |
| Contribution | Problem definition | Comparison of the results | Dataset description |
| | Experimental strategy | Experimental strategy | Experimental strategy |

Table 3: This table illustrates the contribution of each team member.

All in all, interesting results were obtained but the F1 scores indicate that more work is needed to exclusively use these predictions to automate product categorization at Rakuten. They may also be combined with human help when the predictions are uncertain. Active learning may be therefore a relevant technique to industrialize this approach.

## References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.