Ensemble Learning Theory to Application Project

Rakuten France Multimodal Product Data Classification

March 26, 2020

Niccolo BORGIOLI
DSBA
niccolo.borgioli@student-cs.fr

Yingqiang WANG
DSBA
yingqiang.wang@student-cs.fr

Xinxin LU
DSBA
xinxin.lu@student-cs.fr

1 PROBLEM INTRODUCTION

Defining the Problem

This challenge is started by Rakuten, a worldwide famous e-commerce platform. The cataloging of product listings through title categorization is a fundamental problem for any e-commerce marketplace, with applications ranging from personalized search and recommendations to query understanding. Manual and rule-based approaches to categorization are not scalable since commercial products are organized in many classes. Deploying multimodal approaches would be a useful technique for e-commerce companies as they have trouble categorizing products given labels from merchants and avoid duplication, especially when selling both new and used products from professional and non-professional merchants.

The Goal

The aim of the challenge is to classify products into corresponding product type codes based on their designation. The designation contains the product title and a short summary of the product. We will use text preprocessing tools and apply different classification models on the data. The accuracy of prediction will be based on the weighted F1 scores.

2 DATA DESCRIPTION

The train data contains 84916 samples and each sample has 2 features - an integer id and product designation, and 1 target value – product type code marked as 'prdtypecode'. The product designation contains the product title and a short summary of the product. This feature is textual and will be applied with preprocessing tools before use. The target value is categorical, different number stands for different product type.

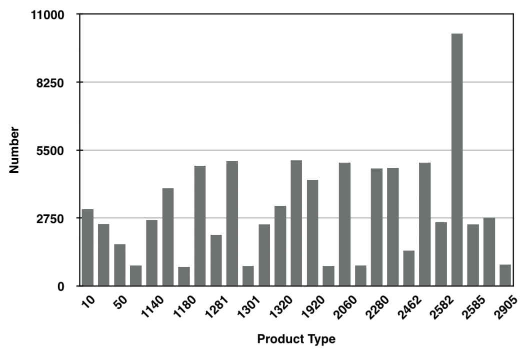


Table 1: Product Type Distribution

3 MODELS USED

Decision Tree Classifier

It is a non-parametric supervised learning method used for classification and the goal of this model is to predict the value of a target variable by learning simple decision rules inferred from the data features. This model is a basic classification model which is simple to understand and easy to interpret. It requires little data preprocessing, accepting both numerical and categorical data. The output could be multiple and trees can be visualized.We import *DecisionTreeClassifier* from *sklearn.tree.*

Random Forest Classifier

It is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Like decision trees, random forest extend to multi-output problems. We import *RandomForestClassifier* from *sklearn.ensemble.*

Bagging Classifier

It is an ensemble meta estimator that fits base classifiers each on a random subsets of the original dataset and then aggregate their individual predictions to form a final prediction. It can reduce the variance by introducing randomization into its construction procedure and then making an ensemble out of it. We import *BaggingClassifier* from *sklearn.ensemble.*

Gradient Tree Boosting

It is a generalization of boosting to arbitrary differentiable loss functions. It supports both binary and multi-class classification. It attempts to solve the minimization problem via steepest descent, which is the negative gradient of the loss function evaluated at the current model. We import *GradientBoostingClassifier* from *sklearn.ensemble.* The algorithm of GradientBoostingClassifier works as below:

$$F(x) = \sum_{m=1}^{M} \gamma_m h_m(x)$$

$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$

$h_m = \arg\min_h \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + h(x_i))$

$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_F L(y_i, F_{m-1}(x_i))$

$\gamma_m = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i)) - \gamma \frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_1)}$

Adaboost Classifier

It fits a sequence of weak learner on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote to produce the final prediction. We import *AdaBoostClassifier* from *sklearn.ensemble.*

Xgboost Classifier

XGBoost model belongs to the wide family of boosting iterative approaches.With boosting models instead of training separately the different models, each model is trained to correct the errors of the previous one.Specifically, XGBoost is a tree-based model based on gradient boosting.The fundamental elements of XGBoost are decision trees and the errors are minimised using a gradient descent-based boosting algorithmWe import *AdaBoostClassifier* from *sklearn.ensemble.*We import *xgboost* directly instead of using scikit-learn.

4 EXPERIMENTAL STRATEGY

Data Preprocessing

Because we are given textual data in designation, which cannot be used directly in models, we have to preprocess these data.

Natural Language Processing – Spacy. We adopt the method of spacy and use the library of 'fr core news sm' to deal with French text. Because it provides very fast and accurate syntactic analysis, and also offers named entity recognition and ready access to word vectors. Firstly, we changed all capitals to lower cases and replaced accents. Secondly, we applied spacy to parse the clean text into words. Thirdly, we removed punctuations and stop words. Lastly, we rejoined the words that left to be sentences and got sentences ready for further processing.

TFIDF. The TF-IDF model is used to scale the amount of feature information by sorting out more representable words and giving them a higher weight. When a word often appears in a specific document, and the frequency of occurrence in other documents is low, which means it is likely to be a word unique to the document and can describe the document well, then the word is given a higher weight; on the contrary the word is given a lower weight.

- TF: Term Frequency, indicates the frequency of each word in the document,which depends on the output of the BoW model.
- IDF: Inverse Document Frequency, represent the popularity of a word. When a word is more common, which means a large number of documents contain the word, the IDF value is lower; otherwise, the IDF value is higher.

$$IDF(X) = log \frac{\#\{documents\}}{\#\{documents\ containing\ wordX\}}$$

- TF-IDF: The TF-IDF value equals to the production of TF and IDF.

Parameters Tuning

Manual Plot – For Loops. We use 'For Loops' to run different parameters with different range of values and plot out the results. By doing this, we could easily find a range of parameters to be put in the GridSearch CV for further work. This step saves us some time by locate the best parameters in a smaller range comparing to directly use GridSearch CV, which is time consuming. We use weighted f1 score as the measurement.

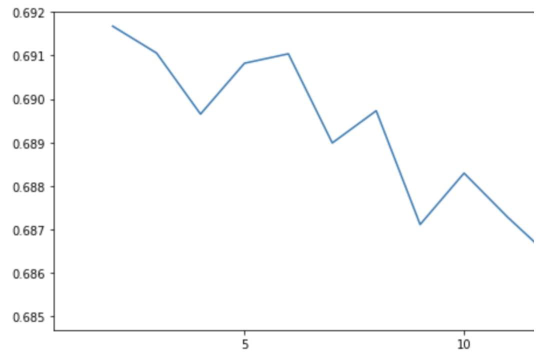


Table 2: Find the Best Parameter for n_estimator
The x-axis: Number of Estimators
The y-axis: Accuracy

GridSearch CV. We use the set of parameter values that generate in the manual plot part as the param grid in GridSearch CV. This method will implement the usual estimator API: when "fitting" it on a dataset, it evaluates all the possible combinations of parameter values and retain the best combination. We import *GridSearchCV* from *sklearn.model_selection.*

RandomizedSearch CV. We also used randomized parameter optimization. This method implements a randomized search over parameters, where each setting is sampled from a distribution over possible parameter values. And it has two advantages over the GridSearch CV:
(1)A budget can be chosen independent of the number of parameters and possible values.
(2)Adding parameters that do not influence the performance does not decrease efficiency.
We applied this method on the Xgboost in our project. We import *RandomizedSearchCV* from *sklearn.model_selection.*

Train_Validation Split. We use *train_test_split* function from *sklearn.model_selection* to randomly split original train data into train and validation data. Then we use train data to fit the model and use the validation data to calculate the accuracy and f1 scores.

K-fold Cross Validation. The training set is split into k smaller sets. And the model is trained using k-1 of the folds as training data, and the resulting model is validated on the remaining part of the data. The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but does not waste too much data. And we choose K = 5 in our project.

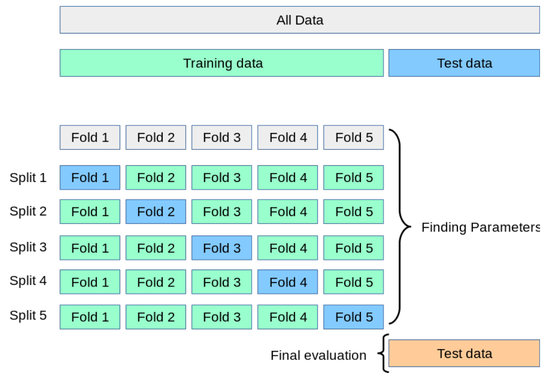


Table 3: How Cross Validation Works

5 COMPARISON AND ANALYSIS OF RESULTS

Decision Tree Classifier

Decision Tree	
F1 Score	Accuracy
0.6917	0.6830

Parameters: min_samples_leaf = 1; min_samples_split = 2; random_state = 0

Analysis: We get the best parameters by applying manual plot. During this process, we find that the prediction accuracy decreases as the min_samples_leaf increases and the same for min_samples_split.So we set the value equals to 1 and 2 respectively.
The Decision Tree Model is a basic classification model that learns simple decision rules inferred from the data features. Decision trees can also be unstable because small variations in the data might result in a completely different tree being generated. Last but not least, decision trees are prone to be overfitting.So the performance can be poor.

Random Forest Classifier

Random Forest	
F1 Score	Accuracy
0.7633	0.7612

Parameters: n_estimators = 28; random_state = 0

Analysis: We get the best parameters by applying manual plot and GridSearchCV. However, when we change the default values of other parameters, especially max_depth, the prediction accuracy decreases greatly to 0.36, so we decided only change n_estimators and left max _depth to be unconstrained.
The RandomForestClassifier is a collection of decision trees whose results are aggregated into one final result, thus limiting overfitting. It is more robust than a single decision tree.

Bagging Classifier

Bagging - Decision Tree	
F1 Score	Accuracy
0.7236	0.7281

Parameters: n_estimators = 10; max_samples=0.8; max_features=0.8

Analysis: We get the best parameters by applying GridSearchCV.
The BaggingClassifier can reduce the variance of a base estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. So the performance will be better than single decision tree.

Bagging - Random Forest	
F1 Score	Accuracy
0.7416	0.7324

Parameters: n_estimators = 28; max_samples=0.8; max_features=0.8

Analysis: RandomForestClassifier and bagging are both ensemble methods and both help reduce the variance, so this combination will perform better than decision tree. But whether it will perform better than just random forest or bagging alone will depends on parameters.

AdaBoosting Classifier

AdaBoosting Classifier	
F1 Score	Accuracy
0.7108	0.7189

Parameters: n_estimators = 28; random_state = 0

Analysis: "The first step simply trains a weak learner on the original data then for each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence." As the algorithm explains, the Adaboosting fully considers the weight of each classifier thus results to higher accuracy if parameters are properly tuned.

Gradient Tree Boosting

Gradient Tree Boosting	
F1 Score	Accuracy
0.7112	0.7253

Parameters: n_estimators = 30; max_leaf_nodes = 4; max_depth = 27; random_state = 0

Analysis: GradientBoosting limits the max_depth thus to minimize overfitting, so it performs better than decision tree. Decision trees in random forests can be built in parallel since they are independent while in boosted tress where the individual trees have to built sequentially, so sometimes random forest performs better than gradientboosting.

XgBoosting

XgBoosting	
F1 Score	Accuracy
0.776	0.77

Parameters: Gamma = 30; Max_depth = 7; Min_child_weigh = 10; Subsample = 0.6; Colsample_by_ = 0.6; Learning_rate = 0.05; n_jobs = -1

Analysis: As you can observe there are a lot of possible combinations and trying all of them thanks to a naïve Grid Search would be too computationally expensive, given our preprocessed dataset size (even after SVD). On that regard, we preferred to perform first a Randomized Search testing 150 possible combinations in the distributions of the parameters. Thanks to the Randomized Search we can apply a raw tuning of the algorithm to identify which parameters have more weigh on the final output. Afterward we performed

a fine tuning of the hyperparameters, with less combinations, thanks to a Grid Search. The gap in performance with and without tuning is big, starting from a F1 score of 0.66 we obtained a final score of 0.78 on Colab. Notice that we used XgBoost on a Dataset with only 1000 features, meaning that it would probably perform even better on the original dataset (after vectorization). For further improvements on XgBoost we wanted to try parallelized computation using GPU on Google Cloud (using RAPIDS Dask_Xgboost Virtual Machine) but we couldn't succeed on that because Google Cloud free trial doesn't give access to it GPU machines. This is a pain given that we could have ran the algorithm way faster using the full benefits of GPU.
XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements. The XgBoosting improves upon the base GBM framework through systems optimization and algorithmic enhancements. It builds trees parallelly and prunes tree using depth-first approach. It has cache awareness and out-of-core computing. It applies own regularization and has in-built cross validation capability. All these advantages over other models contributes to the better performance.

6 CONCLUSION

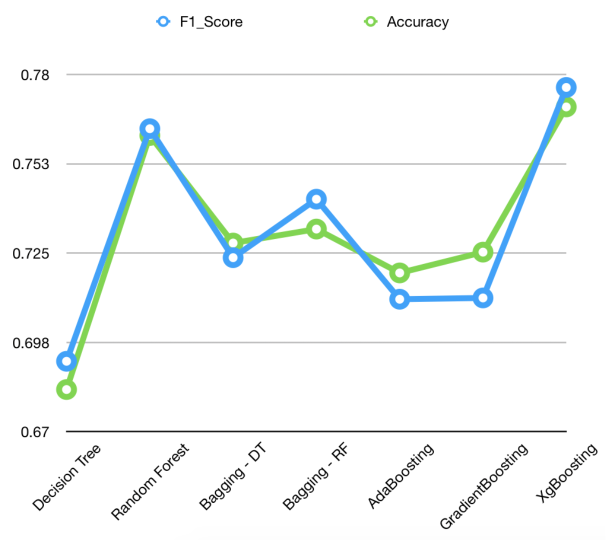
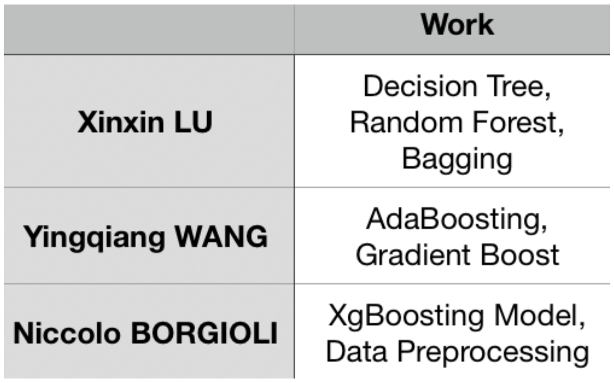


Table 4: Final Results Comparison

As shown in the graph, XgBoosting generates the best performance out of all models, while Decision Tree performs the worst.

During the process of trying different models, we form a better understanding of different kinds of basic classification models and ensemble classification models.

However, we still need further improvements to our models. (1)It is surprising that boosting models such as Gradient Boost and AdaBoost have poorer performance than Bagging and Random Forest. We think it is mainly because we didn't do well in the parameter tuning. The algorithm of Gradient Boost takes a long time to run GridSearch CV. We use the platform of google colab to run our models and sometimes we meet unexpected disruption of running. All these reasons affect the process of parameters tuning and thus cause low accuracy. How to solve these problems still remains a puzzle for us group.
(2)On XgBoost we wanted to try parallelized computation using GPU on Google Cloud (using RAPIDS Dask_Xgboost Virtual Machine) but we couldn't succeed on that because Google Cloud free trial doesn't give access to it GPU machines. This is a pain given that we could have ran the algorithm way faster using the full benefits of GPU.



	Work
Xinxin LU	Decision Tree, Random Forest, Bagging
Yingqiang WANG	AdaBoosting, Gradient Boost
Niccolo BORGIOLI	XgBoosting Model, Data Preprocessing

REFERENCES

[1] Scikit-Learn User Guide - 1.11. Ensemble methods
[2] Scikit-Learn User Guide - 1.10. Decision Trees
[3] Decision Trees and Random Forests; Neil Liberman
[4] Why does Gradient boosting work so well for so many Kaggle problems?; https://www.quora.com/Why-does-Gradient-boosting-work-so-well-for-so-many-Kaggle-problems
[5] Adaboost algorithm; https://easyai.tech/en/ai-definition/adaboost/
[6] XGBoost Algorithm: Long May She Reign!; Vishal Morde
[7] Introduction to Boosted Trees; XGBoost Tutorials