Ensemble Learning: Multimodal Product Data Classification for Rakuten

Project Report. Git Repo - https://github.com/rvs36/Ensemble_Learning

VASHISHT Raghav
B00766425
raghav.vashisht@student-cs.fr

KOGIAS Kleomenis
B00763236
kleomenis.kogias@student-cs.fr

WANG Xiangyu
B00759895
xiangyu.wang@student-cs.fr

1 INTRODUCTION

In 2018, E-commerce grew by 18 percent worldwide, however such increase is mainly attributable to Amazon which is responsible of around 80 percent of this growth. One of the main reasons might be linked to an adequate catalog management and more precisely to the use of multimodal deep networks for product categorization. This classification is a key driver of the shopping experience through efficient product search on the catalog or recommendation systems.

About Rakuten

Rakuten, created in 1997 in Japan and at the origin of the marketplace concept, became one of the largest e-commerce platforms worldwide with a community of more than 1.3 billion members. The cataloging of product listings through title and image 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 images and labels from merchants and avoid duplication, especially when selling both new and used products from professional and non-professional merchants, like Rakuten does.

Problem Definition

Given the integer ID and the designation of a product we need to predict each product's type code as defined in the catalog of Rakuten France.
For example, in Rakuten France catalog, a product with a French designation or title Klarstein Presentoir 2 Montres Optique Fibre associated with an image and sometimes with an additional description. This product is categorized under the 1500 product type code. There are other products with different titles, images and with possible descriptions, which are under the same product type code. Given these information on the products, like the example above, the objective is to model a classifier to classify the products into its corresponding product type code using the various Natural Language Processing operations on the product designation.

2 PRIOR WORK

A benchmark model has been built by the data science team at Rakuten. The benchmark algorithm uses two separate models for the images and the text.
For this assignment, ensemble learning methods are used and hence, the a comparison with the benchmark (which is a neurla network) will not be highly appropriate for the scope of this work.

Text Benchmark Model

The text benchmark model is implemented with a CNN (Convolutional Neural Network) of 18-layer depth, containing 6 convolutional layers. The input size consists of 34 neurons as many as the maximum length of the designation field. In case of input shorter than 34 characters, zero padding is used. The above model, is fed only with the designation field of each product, achieving weighted F1-score of 81.1%.

Image Benchmark Model

Secondarily, a model trained only with the images of each product is built. For this purpose, the architecture of ResNet50 Deep CNN is used, pre-trained with the ImageNet dataset. From the 36 million parameters of the model, 24 million remain frozen (27 first layers) while the rest of them are adjusted to the needs of the classification task. The Image Benchmark model achieves a weighted F1-score of 55.34%, besides its more complicated structure, the text-fed model achieves significantly better results.

3 DATASET DESCRIPTION

For this challenge the 99K product list is provided, which is divided into training-set (84,916) and test-set (13,812). For all the products, information is given both in text and image format. The images of all the products is provided. As far as the text is concerned, 3 seperate csv files are given, X_train.csv, Y_train.csv and X_test.csv. The first two files are

used for training process, product features and their labels respectively, while the last one contains the text-features of the test set.
The text features for the products are:

(1) **integer ID:**An integer ID for the product. This ID is used in the Y_train.csv, to bind each product with its corresponding label.
(2) **designation:** The product title followed by a short text summarizing the product
(3) **Description:** A more detailed description of each product
(4) **productid:** An unique ID for the product.
(5) **imageid:**A unique ID for the image associated with the product.

Both for the Rakuten challenge and the scope of this project only the designation field of the text-feautures is taken into consideration.

It is highly important to understand the distribution of the dataset. On the training set there are overall 27 product types taking values from 10 to 2905. The distribution of the product types is depicted on Figure 1, showing that most labels have close to 4000 observations, while the product type of '2583' has more than 10000 products. The different number of observations leads to imbalanced training of the models and that's why weighted F1-score is chosen to evaluate the performance.

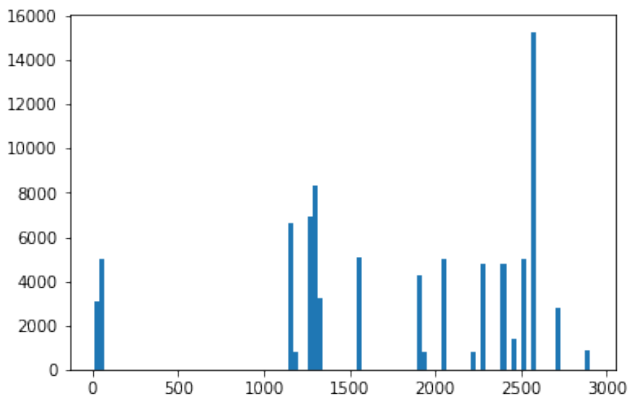

Figure 1: Dataset Distribution on different labels

4 METHODOLOGY

The product designation column is a free flowing text in French, hence to create feature useful for modelling some text processing steps were applied using the pre processing techniques as mentioned later. After that various ensemble learning models were created and their hyper parameters were tuned using the SKlearn random search.

Text Mining

Text mining is the process of automatically extracting "high-quality" information from text. High-quality information is typically derived by transforming the free (unstructured) text in documents and databases into normalized, structured data suitable for analysis or to drive machine learning (ML) algorithms.Text mining identifies facts, relationships and assertions that would otherwise remain buried in the mass of textual big data. Once extracted, this information is converted into a structured form that can be further analyzed, or presented directly using clustered HTML tables, mind maps, charts, etc. Text mining employs a variety of methodologies to process the text, one of the most important of these being Natural Language Processing (NLP).

Text Pre-processing

For this project, some basic text preprossessing methods were applied and then on top of those various combinations of more fancy text preprossessing were tried and then tested on a basic Boosting model (*** Add details of the model****) and then the best combination was chosen. Various basic text preprocessing steps applied are:

- *Lower case the strings* - change all the uppercase alphabets into lowercase.
- *Handling Accented Character*- Diacritics or accents on characters in English have a fairly marginal status. Ideally, cliché and cliche should match, or naive and naïve. This can be done by normalizing tokens to remove diacritics. All French accents were listed and string replace was used to convert accented characters with their canonical form.
- *Tokenization* - It is a task of chopping a character into pieces, called as token, and throwing away the certain characters at the same time, like punctuation, the Spacy package in French was leveraged for the same.
- *Handling Punctuations* - Punctuations are noises while processing text. We are more interesting in the words itself. SpaCy recognises punctuation and is able to split these punctuation tokens from word tokens.
- *Stop word filtering* - Stop words are words which are filtered out before or after processing of natural language data (text). There is no universal stop-word list. Often, stop word lists include short function words, such as "the", "is", "at", "which", "on" etc. in English and "le", "la", "et", "à", "qui" etc. in French. Removing stop-words has been shown to increase the performance of different tasks like search. The stop words detection by spaCy will detect only French stop words.

Various optional preprocessing steps were applied are in order to increase the performance:

- Removing numbers -In the TF-IDF matrix created, there was a plethora of "words" containing only numbers. In very few occasions the numbers provided important information, such as the dimensions of the product, but in most cases did not make sense. For this reason, the numbers were removed from the TF-IDF matrix.
- Removing Special symbols like Phi or Degrees - Some descriptions had special symbols which seemed redundant while creating the TF- IDF matric, hence all such special symbols were also removed.
- Lemmatization - It is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.

To compare the effect of these reprocessing approaches on the prediction performance, we input differently processed training data into the same basic Bagging model based on Decision Trees with estimator number=20, max_features=0.5, and max_samples=0.5. The results are summarised below:.

Pre-processing	Accuracy (%)
Numbers and Special Symbols removed	0.7668
Lemmatization	0.7662
No additional Preprocesssing	0.7645

The additional pre-processing steps did not alter the performance of the Bagging model, that's why the numbers removed approach was chosen, removing this way a substantial number of columns without impacting the final performance.

Ensemble Models Used

In this project the performance of various models is tested

Decision Trees: The fundamental structure of many ensemble learning models is the decision tree. Decision tree is a greedy algorithm that splits the input space into subspaces, aiming to maximise the homogenity of the training data within each subspace. The splitting stops based on certain criteria, such as the depth of the branch, the number of observations on the leaf and the purity of each leaf. Additionally, decision trees are susceptible to overfitting, for this reason the technique of pruning is applied. Pruning can happen either during the training process(pre-pruning) or after the end of the training process(post-pruning). Its main idea is to cut-off certain branches of the tree, in order to reduce the phenomenon of overfitting.

Decision trees, even if they can achieve satisfying results, they are considered weak learners and their combination is proved to achieve improved results. Decision trees can be mainly combined by two techniques, bagging and boosting.

Bagging: The first ensemble learning method is bagging. In bagging many weak (base) learners are trained in parallel, each one of them using a resampled training set. In other words, from the initial training set, observations are randomly drawn with replacement, creating a modified training set for each base learner, repeating or leaving out some of the initial observations(bootstrap). Afterwards, the base learners are trained separately and their outcomes are combined with a voting scheme in the end. The different training sets used for each learner as well as the voting scheme in the final step have as a result a method that reduces the variance of a decision tree.

Gradient Boosting: Gradient Boosting consists of two important notions, boosting and the gradient algorithm. Boosting, is the second main approach to combine weak learners, this time sequentially, with the purpose of improving the errors of the previous weak learner. Overall, gradient boosting consists of a loss function, a weak learner and an additive process. In each iteration a weak learner is added to the already existing model, minimising the overall error by following a gradient descent procedure.

AdaBoost: The actually first important boosting algorithm is Adaboost. In each iteration a weak learner is chosen in order to minimise the classification error, additionally a weight is assigned to it according its relevance. For a classification task, the equation that describes the above behaviour is:

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

where M is the total number of iterations, f_m is the weak learner chosen at step m and θ_m is the importance of each weak learner.
In each iteration, a weak learner is introduced (f_m):

$$F_m(x) = F_{m-1}(x) + \theta_m f_m(x)$$

The purpose is to find the weak learner f_m with importance θ_m that minimise each time the error value, denoted by E

$$E = \sum w_x^m e^{-y(x)\theta_m f_m(x)}$$

The weight in the above function is defined as $w_x^m = e^{-y(x)F_{m-1}(x)}$.
In a basic binary classification problem where $y \in$ {-1, 1}, for each correct classification the product y(x)f(x) is equal to 1, while in case of misclassification it is equal to -1. The loss function is thus transformed to:

$$E = \sum_{correct} w_x^m e^{-\theta_m} + \sum_{false} w_x^m e^{+\theta_m}$$

The estimation of the weak learner is done by minimising the loss function:

$$\frac{\partial L(y, F(x))}{\partial F(x)} = 0$$

After finding the most suitable weak learner (f), its importance is estimated by minimising the equation:

$$\frac{\partial L(y, F(x) + \theta_m f_m(x))}{\partial F(x)} = 0$$

Therefore, the boosting algorithms additively find the best parameters θ, f of the following equation:

$$\{\hat{\theta_m}, \hat{f_m}\} = argmin_{\{\theta_m, f_m\}} \sum L(y_i, F_{m-1}(x) + \theta_m f_m(x))$$

and specifically for the case of Adaboost using the exponential loss function.

Random Forests: It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set by adding some randomness into the training process.Random forests differ in only one way from the general scheme of the bagging: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging".

XGBoost: Similarly to gradient boosting, XGBoost is an ensemble method, boosting decision tree weak learners and using more accurate approximations to find the best tree model. Algorithmic enhancements and hardware optimisation are the key components that make XGBoost the state-of-the-art model amongst the different ensemble learners. The notions of 'depth-first' tree-pruning and parallelisation, which are used on XGBoost, improve the computational performance of the model. Additionally, the cache-aware block structures further enhance the efficiency of the model, making it this way a highly scalable system. Algorithmically, the principle of regularisation is applied, preventing overfitting as well as the weighted-quantile-sketch in order to find the optimum splits. These are some of the most important contributions of XGBoost, making it the best model time-wise and performance-wise.

5 EXPERIMENTAL PART

Hyperparameter tuning

In all the ensembling methods analysed above, the optimisation of their hyperparameters impacts greatly their performance. Two approaches were suggested in order to deal with this issue. The first one, is to grid-search the hyperparameter space using 50% of the TF-IDF matrix. The alternative one, is to use the whole TF-IDF matrix and randomly search through the different combinations of the hyperparameters. These two methods were necessary, because a grid search through the whole dataset would have been exceptionally time-consuming, besides the powerful resources (RAM,CPU, GPU) used.
The method that was chosen, is to use the whole TF-IDF matrix and randomly test its performance in 50 different combinations of its hyperparameters. The models were validated by using 3-fold cross validation. This way an indicative subspace of the parameters is tested while on the same time the accuracy results are representative to the ones that can be achieved on the test-set.

The hyperparameters tested for each ensemble model are presented in the table below. The abbreviations of DT, RF and GB stand for Decision Tree, Random Forest and Gradient Boosting respectively.

Model	Hyperparameters tested
DT	criterion, max_depth, splitter
Bagging	estimator, n_estimators, bootstrap
RF	n_estimators, max_depth, min_samples_split, min_samples_leaf, bootstrap
AdaBoost	n_estimators, learning_rate, base_estimator
GB	n_estimators, min_samples_leaf, max_leaf_nodes, max_depth
XGBoost	n_estimators, max_depth learning_rate, gamma, colsample_bytree

Computation power

Given the size of the data set with approx. 85,000 rows of training data and 13,000 rows of testing computation was a big challenge. Hence, the services from Google Cloud were leveraged for the purpose. Google Cloud allows us to access machines with gigantic computing powers.This model was trained on a machine with 16 vCPUs, 60 GBRAM and NVIDIA Tesla T4 x 1 GPU.
Despite using such powerful machines, it was extreemly slow to train the gradient boosting model while XGboost was trained relatively quicker, exhibiting the ability of XGboost to handle big data sets thanks to featuers like -

- Approximate greedy algorithm
- Parallel learning
- Weighted quantile sketch
- Cache Aware access
- Blocks for out of core computations

6 RESULTS

Below are presented the best weighted F1-scores achieved for the various ensemble models. These results correspond to the weighted F1-score of the test-set, after submitting our predictions to Rakuten challenge.

Model	weighted F1-score (%)
Decision Trees	72.21
Bagging	79.37
Random Forests	78.02
AdaBoost	57.67
Gradient Boosting	70.11
XGBoost	80.03

From the previous table, it can be inferred that most of the ensemble models achieve weighted F1-score between 70 to 80%. Adaboost has the lowest performance by scoring almost 58%, while XGBoost has the best performance, slightly above 80%. XGBoost's best performance is achieved by setting its hyperparameters to 1000 estimators, maximum depth of 6, learning rate of 0.3 and gamma value of 0.3.
The models analysed above can be divided into 3 categories, a decision tree, which is a base learner, the ensemble models that are built based on the Bagging method, that is Bagging and Random Forests and lastly the models that are built based on the boosting method, that is Adaboost, Gradient Boosting and XGBoost.
In the 99k product dataset, we can observe that the boosting models perform the worst as both Adaboost and Gradient Boosting have results less than 71%, on the other hand the bagging algorithms score close to 79%. That means that for this classification problem models that aim to reduce the complexity and tackle the issue of overfitting suit best, as this is the main functionality of bagging algorithms. On the other hand, algorithms that reduce the bias and increase the variance, increasing the complexity, like the boosting algorithms under-perform on this classification task.
XGBoost can be considered a special case because of its extremely advanced characteristics. The advantages of regularisation, handling missing values, cross validation at each iteration and effective tree-pruning distinguish itself. Even with these optimisations, XGBoost scores no more than 1% better than the Bagging algorithm.

Evaluation Criteria

The evaluation metric used is the weighted-F1 Score. F1 score is a metric that takes into consideration both recall and precision and takes a range of values between 0 and 1. Scikit-Learn package provides an F1 score implementation and the weight is added by setting the average setting to 'weighted'. 'Weighted' calculates metrics for each label, and find their average weighted by support (the number of true instances for each label). This account for label imbalance;

7 PREDICTIONS USING THE IMAGES

As an extension of the predicting the class of the products using the text description, we tried to use the images to to make simialr predictions.

Model Selection

VGGNet-16 is a convolution neural net (CNN) architecture which was used to win ILSVR(Imagenet) competition in 2014. It is considered to be one of the excellent vision model architecture till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameter it focuses on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC(fully connected layers) followed by a softmax for output. As shown in the image.

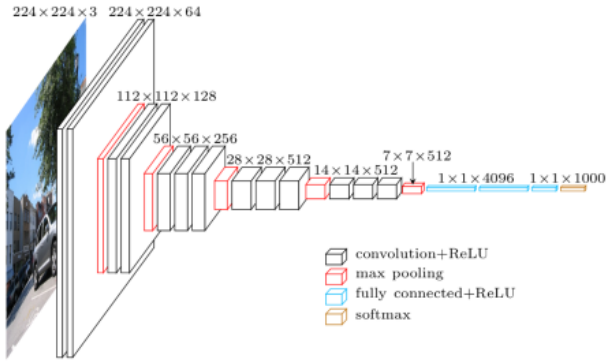

Figure 2: VGG Architecture

This work uses the same architecture, and hence the neural network used has a total of 134,362,969 trainable parameters. The model summary can be seen in the image.

Model hyper-parameters

- The model has *steps_per_epoch* equal to 100 i.e. the total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch.
- For the early stopping the patience is 20 epochs, with monitor on validation accuracy.
- Total number of steps (batches of samples) to validate before stopping i.e. the *validation_steps* is set to 100
- And the total number of epochs is 50.

Early stopping

Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows to specify an certain number of training epochs and stop training once the model performance stops improving on a hold out validation dataset.
For the early stopping a monitor is set on validation accuracy and hence the model was saved before the model started to overfit. As seen in the image, the validation loss increases after the early stopping point and the validation accuracy very subtly starts to decay.

8 VGGNET-16 RESULTS

VGG16, with the parameters defined above, obtained a weighted F1-score of 51%.

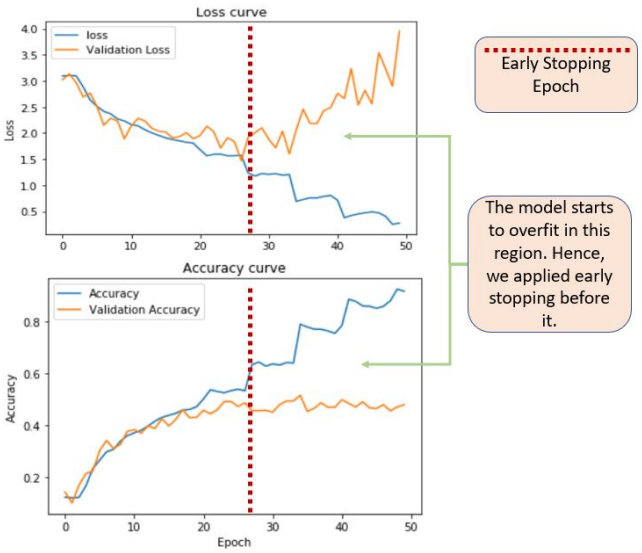

Figure 3: Early stopping in training the VGG model

9 CONCLUSION

To sum up the development of this project is divided into three phases. The first one, is pre-processing the designation field of each product. Towards this end, the techniques of lower-casing the text, handling the accented characters, tokenization , stop word and punctuation handling as well as number removing is applied. Afterwards, the pre-processed text is fed into different ensemble learning methods, in order to compare their performance. The best performance is achieved by using XGBoost, having a weighted F1-score of 80%. The last and optional phase of this project, is to take advantage of the images provided. For this purpose, VGGNet-16 is selected achieving F1-score of 51%. In future, is proposed to combine the features deriving from the designation field and the images aiming to improve the current results.

Member	Contribution
Raghav Vashisht	Random forest, Gradient Boosting and Image prediction
KOGIAS Kleomenis	XGboost. DecisionTrees and Text Pre processing
WANG Xiangyu	Bagging, AdaBoost and Text Pre processing

10 REFERENCES

(1) Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition
(2) Ho TK (1998). "The Random Subspace Method for Constructing Decision Forests
(3) Ho, Tin Kam (2002). "A Data Complexity Analysis of Comparative Advantages of Decision Forest Constructors" (PDF). Pattern Analysis and Applications. 5 (2): 102–112.
(4) Friedman, J. H. (February 1999). "Greedy Function Approximation: A Gradient Boosting Machine"
(5) Mason, L.; Baxter, J.; Bartlett, P. L.; Frean, Marcus (1999). "Boosting Algorithms as Gradient Descent"
(6) https://homes.cs.washington.edu/ tqchen/pdf/BoostedTree.pdf
(7) https://xgboost.readthedocs.io/en/latest/index.html
(8) https://arxiv.org/pdf/1603.02754.pdf