final_report

March 5, 2023

1 Rapport technique

Participants : Olga TOLSTOLUTSKA, Mohamed BACHKAT, Charly LAGRESLE
Mentor : Manu PORTEL
Promotion : DST Bootcamp DEC22
Date du document : 2023/03/06

2 Introduction

2.1 Contexte

Rakuten est un site de e-commerce, regroupant près de 1.3 milliards d'utilisateurs. Afin de faciliter la gestion des suggestions de leurs recherches ou des recommandations qui leurs seront proposées, il est important de classifier les produits. Du fait de la quantité de produits et de leurs catégories associées, ce classement ne peut ni être fait manuellement, ni être régi par des règles. Une approche plus flexible, efficace et reproductible est donc à mettre en place.

L'objectif est de prédire le code de chaque produit, c'est-à-dire sa catégorie, sur la base d'une analyse de données textuelles et d'images.

2.2 Étapes du projet

Le déroulement du projet suit la statégie suivante :

- Analyse exploratoire
  - Prise en main des données
  - Analyse statistique et feature-engineering
  - Choix des métriques
- Conception de modèles de référence en machine learning
- Conception de modèles deep-learning
  - Traitement des images et du texte
  - Fusion de modèles

Le rapport ici présent se compose de parties similaires à ce plan.

3 Analyse exploratoire

L'exploration des données est la première étape de ce projet. Cette dernière permet de se familiarier avec le jeu de données, d'en comprendre la structure, les particularités et d'anticiper les actions à mettre en place pour extraire, transformer et traiter ces données.

3.1 Prise en main des données

Les données fournies pour ce challenge sont les suivantes : * `X_train_update.csv` : données explicatives et textuelles pour l'entraînement * `X_test_update.csv` : données explicatives et textuelles pour les tests * `Y_train_CVw08PX.csv` : variable cible pour l'entraînement * `images/image_train/*.jpg` : images pour l'entraînement * `images/image_test/*.jpg` : images pour les tests

> Nous notons que la varible cible de test n'est pas fournie : les observations de tests ne seront donc pas exploitées.

Le jeu d'entrainement se compose a priori de 84 916 observations : des données textuelles ainsi que des images.

3.1.1 Données d'entrées

Deux types de données d'entrées sont à notre disposition : des textes et des images. Nous analysons chacun de ces types de données séparemment.

Textes Les données `X_train_update.csv` se compose de plusieurs champs :

- `designation` : titre du produit dans différentes langues (0% de NaNs)
- `description` : description détaillée du produit dans différentes langues (35% de NaNs)
- `productid` : identifiant unique du produit (0% NaNs)
- `imageid` identifiant unique de l'image du produit (0% NaNs)

> Les champs `productid` et `imageid` ne servent qu'à construire le nom du fichier image associé au produit. Ils ne seront donc pas étudiés statistiquement parlant.

Près de 35% des observations de la colonnes `description` est manquant : cette colonne sera par la suite fusionnée avec la colonne `designation`, ce qui rendra nul le nombre de NaNs de cette nouvelle colonne.

Images Chaque produit est associé à une image qui le représente. L'image comporte le produit, généralement photographié sur un fond blanc, mais peut également être complété par une mise scène du produit (jardin, décoration) ou par un support (main portant le produit par exemple). Les images sont en couleur, ont une taille de 500x500 pixels et nous sont fournies au format `.jpg`.

3.1.2 Données de sorties

Les cibles sont fournies dans le fichier `Y_train_CVw08PX.csv`. À ce stade, nous nous contentons de noter que 27 catégories différentes seront à déterminer.

3.1.3 Analyse du problème

Type de problème Les observations textuelles et graphiques doivent permettre au modèle d'affecter à chaque observation dont les features sont actuellement (`designation`, `description` , et une image) à une catégorie de produit `prdtypecode`. Cette cible est disponible et se présente sous la forme d'un numéro associé à une catégorie (par exemple `livre`, `tech`, `décoration`, etc.).

Nous avons donc affaire à un problème de catégorisation supervisé. Pour la partie texte, un analyse automatique du langage naturel (*NLP : Natural Langage Processing*) sera à mettre en place. Pour la partie image, l'analyse des pixels et de leur organisation ou *pattern* sera à réaliser.

Particularité du problème La catégorisation se base sur l'apprentissage fait sur les données à notre disposition. Ces données sont de deux types (textuel et graphique). De ce fait : * les prétraitements seront différents * plusieurs modèles seront à concevoir, à entrainer et à faire dialoguer * un modèle pour catégoriser le produit en fonction du texte fourni * un modèle pour catégrosier le produit en fonction de son image associée * une étape de regroupement des prédictions sera à réaliser afin d'obtenir une unique catégorie en fin de process

3.2 Analayse statistique et feature-engineering

Maintenant que la structure des données est connue, l'étape suivante est d'analyser plus en profondeur les données, d'en créer de nouvelle et de préparer le dataset à son entraînement. Ces étapes de feature-engineering sont détaillées dans les prochains paragraphes.

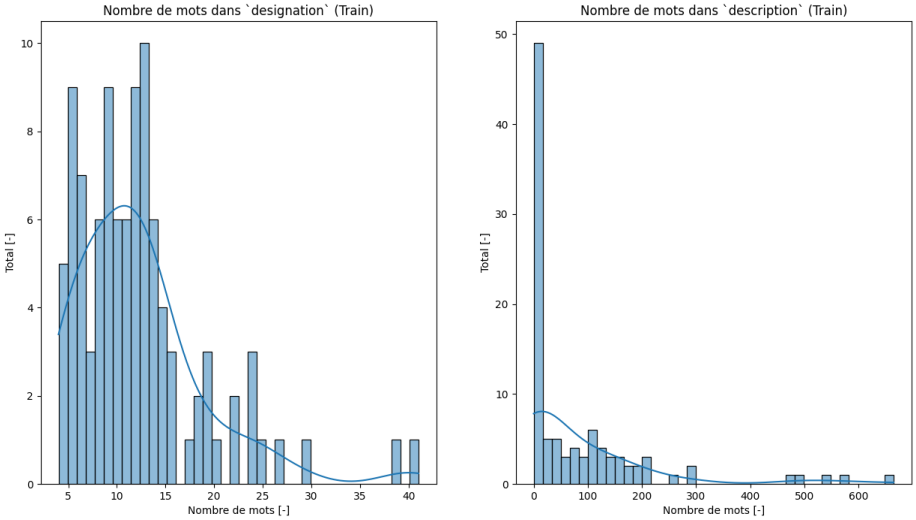
3.2.1 Données d'entrée : textes

L'analyse exploratoire et statistiques de données textuelles nécessite que des données complémentaires soient calculées pour caractériser numériquement nos textes.

Nombre de mots Pour `designation`, les textes sont composés de 4 à 54 mots. La répartition est de type gaussienne : la moyenne se situe à 11.5 mots et l'écart-type est de 6.4 mots. 75% des textes de `designation` ont moins de 14 mots.

Pour `description`, les textes plus longs et comportent entre 0 (certaines descriptions sont vides) et 2 068 mots. En moyenne, la description se compose d'un texte de 78 mots et 75% des produits sont décrits avec moins de 124 mots. La répartition du nombre de mots suit ici la densité de probabilité d'une loi exponentielle.

[88] :



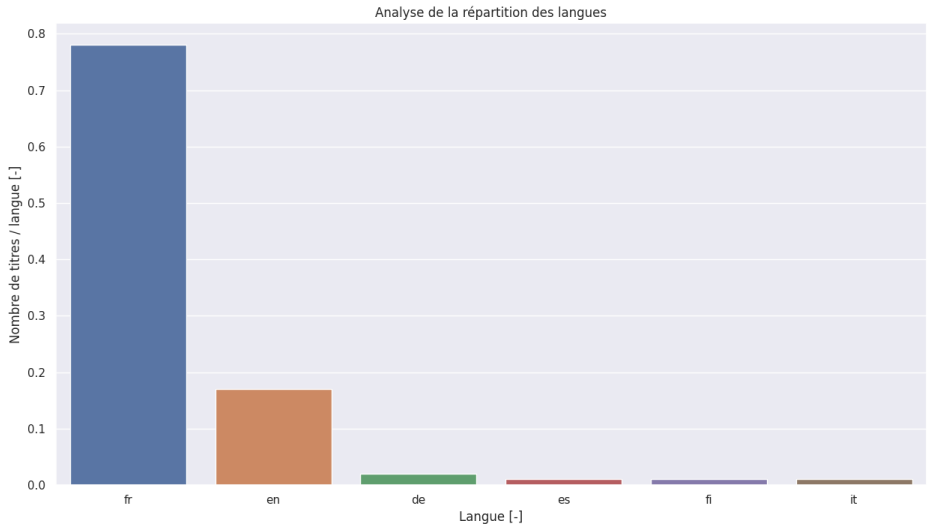
Plusieurs actions seront mises en place suite à cette étude

- Les colonnes `designation` et `description` seront fusionnées en une seule colonne `text`
- La colonne `text` sera limitée à 500 mots.

Langues Intéressons-nous maintenant à la langue. Les colonnes `designation` et `description` contiennent du texte dont la langue peut varier. Le graphique ci-dessous montre la répartition des langues pour la colonne `designation`. La répartition des langues est très hétérogène : * 81% en français * 14% en anglais * 1.5% en allemand * inférieurs à 1.5% : nl, ca, it, ro, pt, etc…

Le choix a été fait de traduire les textes étrangers vers le français.

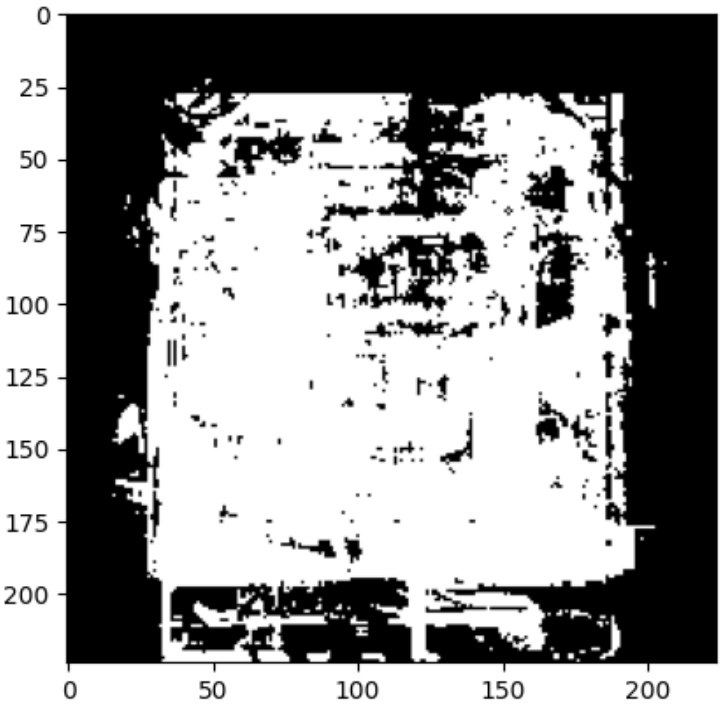
[89] :



3.2.2 Données d'entrée : images

Réduction de dimension Nous avons noté que beaucoup d'images sont photographiées sur un fond blanc. Ce fond n'apporte en soi aucune information et peut mener à un biais dans notre modèle de classification des images.

[90] :



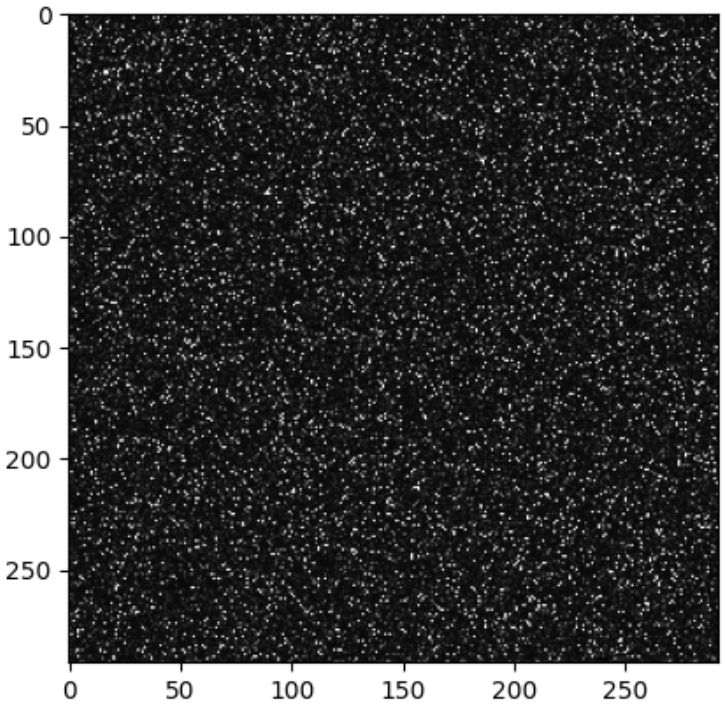
En faisant une analyse de la variance des pixels sur un échantillon d'images, nous notons que les bordures des images présentent une variance très basses.

Les images pourront être rognées de 20% pour supprimer ces marges porteuses de peu d'information avant d'être redimensionnées à la dimension souhaitée.

3.2.3 Données de sorties

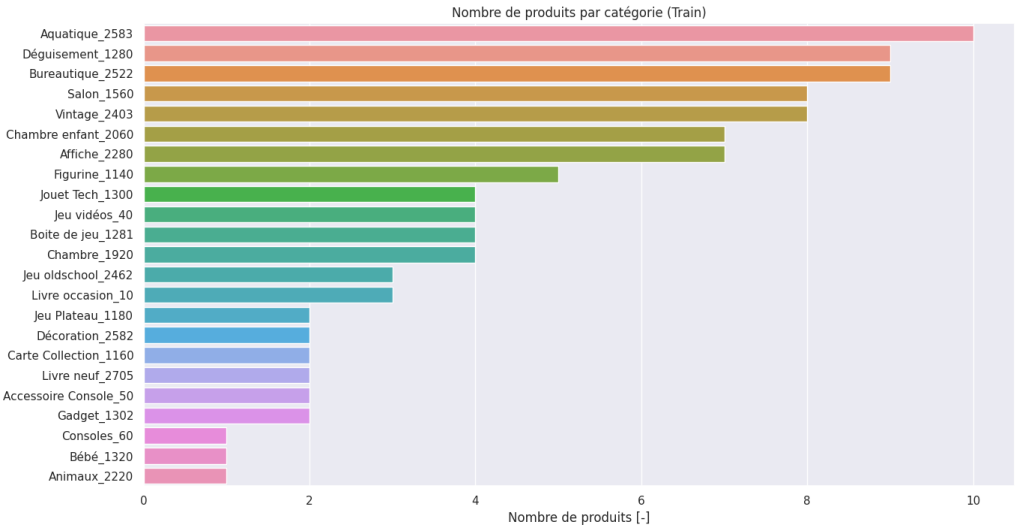
Commme expliqué précédemment, 27 catégories sont présentes dans notre dataset. La figure ci-dessous présente sur 84 916 pixels les catégories en nuance de gris : aucun patern n'émerge visuellement, le dataset d'origine semble être déjà randomisé.

[91]:



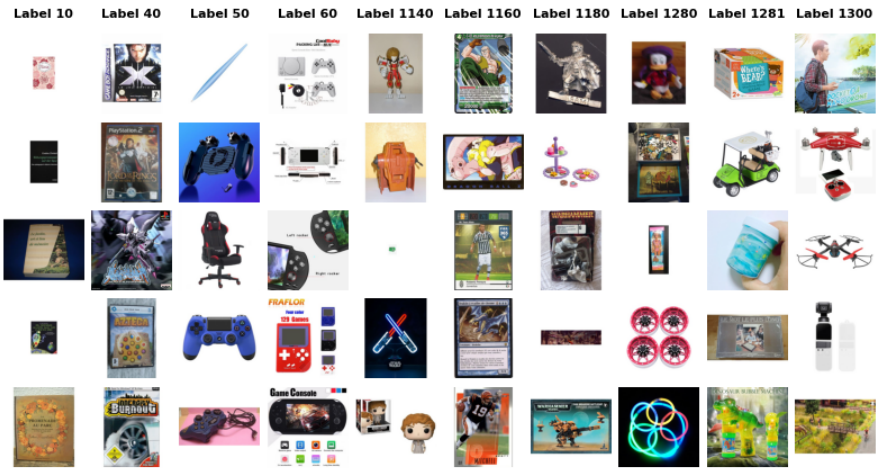
Déséquilibre Le graphique ci-dessous montre le nombre de produits pour chacune des 27 catégories. On remarque facilement que la catégorie “Aquatique” est sur-représentée : elle comporte le double de produits par rapport à la catégorie suivante. On observe également une baisse linéaire du nombre de produits de `Affiche` vers `Accessoire console`. Les catégories de `Cuisine` à `Jeu PC` ne présentent que peu d’individus.

[92]:



Catégories L'analyse visuelle de quelques produits pour chaque catégorie nous permet d'une part de traduire le numéro de catégorie en une variable plus compréhensieble et d'autre part d'étudier la constitution de chacune d'entre elles. Cette analyse a été faite sur toutes les catégories, voici ci-dessous un exemple pour les 10 premières d'entre elles.

[93] :



Les catégories et leurs descriptions sont proposées dans le tableau suivant.

Numéro de catégorie	Description
10	Livre d'occasion
40	Jeu Console
50	Accessoire Console
60	Tech
1140	Figurine
1160	Carte Collection
1180	Jeu Plateau
1280	Déguisement
1281	Boite de jeu
1300	Jouet Tech

Numéro de catégorie	Description
1301	Chaussette
1302	Gadget
1320	Bébé
1560	Salon
1920	Chambre
1940	Cuisine
2060	Chambre enfant
2220	Animaux
2280	Affiche
2403	Revue

Numéro de catégorie	Description
2462	Jeu oldschool
2522	Bureautique
2582	Décoration
2583	Aquatique
2585	Soin et Bricolage
2705	Livre neuf
2905	Jeu PC

Certains catégories ont été délicates à définir, car présentant de très fortes similitudes avec d'autres. Des catégories plus larges pourraient être identifiées. C'est par exemple le cas pour :

- Gaming : jeu PC et jeu Console
- Lecture : livre neuf et livre d'occasion
- Jeu : figurine, jeu oldschool et jeu manuel

Les algorithmes de classification éprouveront probablement les mêmes difficultées à distinguer les produits à l'intérieur de ces catégories plus étendues.

3.3 Choix des métriques

Afin de pouvoir qualifier la performance de nos modèles, le choix de la métrique est important. Les notions importantes avant de considérer le choix de la métrique sont les suivantes :

- connaissance métier : ici, nous catégorisons des produits sur un site de vente et faire une erreur de catégorie n'est pas fatidique, l'erreur peut être corrigée.
- jeu de données déséquilibré : nous n'avons pas d'informations sur l'origine de ce déséquilibre. Une catégorie sur-représentée présente-elle les produits les plus vendus ou les produits les plus souvent mal classés?
- forte tendance à l'overfitting
- une erreur de catégorie n'est pas dramatique

Notre choix se porte sur le *weighted f1-score*, qui considère le déséquilibrage du dataset et fournit un bon compromis entre le *recall* et l'*accuracy*. Si nous avions d'avantage de connaissances des besoins de Rakuten, une nouvelle métrique aurait pu être développée.

Enfin, deux informations complémentaires relatives à la comparaison des modèles: * 27 catégories sont à notre disposition. Cela signifie qu'un score de plus de 3.7% est d'ores et déjà supérieur à un modèle purement aléatoire. * afin de garantir une reproductibilité des résultats, les échantillons d'entrainement et de tests sont pour tous les modèles similaires, avec un *random_state* fixée à 123.

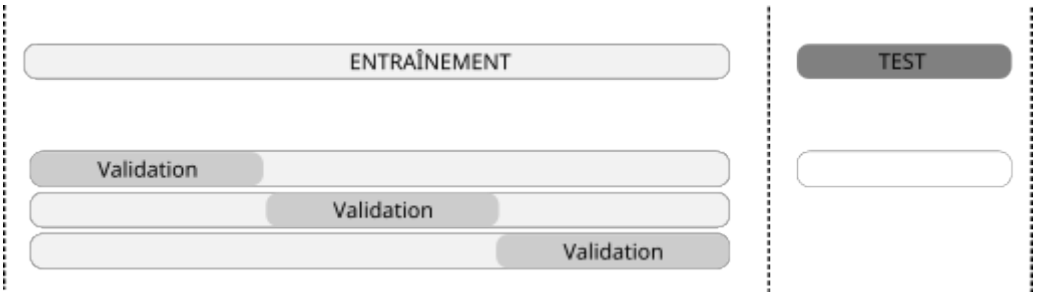
3.3.1 Prévention de l'overfitting

Les modèles implémentés sont sensibles à l'*overfitting*. Afin de minimiser au maximum ce risque, plusieurs actions ont été mises en place : * suivi de l'évolution des métriques d'apprentissage via *Tensorboard* * * ajout des couches `Dropout` dans nos architectures * augmentation nos données * simplification l'architecture des modèles * utilisation de batches de petite taille * application d'une cross-validation à 3 échantillons : * le modèle est fit sur les données *train* * à chaque époque, son évalution est faite sur les données *validation* * une fois l'entraînement terminé, le modèle est ensuite globalement étudié sur les données *test* * appel à des callbacks

Les callbacks sont utiles pour contrôler le déroulement de l'apprentissage. Nous avons mis en place deux méthodes de gestion de l'apprentissage : * `EarlyStopping` : met fin à l'apprentissage si `val_loss` augmente pendant plus de 5 périodes à partir de la 8ème période * `ReduceLROnPlateau` : réduit le taux d'apprentissage si `val_loss` stagne sur un plateau pendant plus de 5 périodes

Finalement, le découpage du dataset pour l'entraînement et la prediction est le suivant :

[94]:



4 Modèles de références

L'analyse exploratoire nous a permis de définir une stratégie pour le traitement du texte et des images. Des modèles de références simples sont mis en place pour catégoriser les produits en fonction des données textuelles et des images à notre disposition. Ces modèles simples se basent sur des techniques de *machine learning* et sur l'ensemble des données fournies.

4.1 Textes

Les modèles de texte nécessitent un travail de preprocessing important. Une fois la méthodologie mise en place, des modèles de classification simple sont entraînés pour définir la catégorie du produit en fonction du texte dont nous disposons.

4.1.1 Étapes de *preprocessing*

Les étapes suivantes sont éxécutées via des pipelines afin de transformer les données textuelles :

- calculer les longueurs des textes pour `designation` et `description`
- calculer le nombre de mots pour `designation` et `description`
- fusionner ces deux colonnes dans `text`
- détecter la langue dans `text`
- préparer la colonne `text` :
  - supprimer la ponctuation
  - supprimer les caractères spéciaux
  - supprimer les valeurs numériques
  - supprimer les majuscules
  - supprimer les balises html
  - supprimer les *stopwords*
  - extraire la racine des mots
  - vectorisation

Extraction de la racine des mots permet de fusionner des mots partageant la même racine et de regrouper leurs poids. Par exemple, les mots `marée`, `marin`, `maritime` sont basés sur le même radical `mar` (ou `mer`).

Suppression des stopwords qui sont des mots courament utilisés dans la langue choisie. En français, les mots `afin`, `elle`, `est`, `sur` en font par exemple partie : ils n'apportent peu ou pas d'informations. Cette étape a été maintenue en place bien que le `vectorizer` `tf-idf` aurait pénalisé ce genre de mots.

Vectorisation du texte via un `Tokenizer`. Ici est calculer l'importante du mot dans un texte. Nous n'utilisons pas le tokenizer en mode `counter` (le mot le plus important est le mot le plus fréquemment présent) mais en mode `tf-idf` pour *Term Frequency - Inverse Document Frequency*. La partie `TF` calcule la fréquence d'apprition d'un mot dans un texte alors que la partie `IDF` corrige ce premier terme en analysant la distribution d'un mot dans l'ensemble des documents. Ainsi, un mot apparaissant fréquemment dans quelques docuemnts seulement aura d'avantage de poids qu'un mot très fréquent dans tous les documents.

À la fin de cette étape de *preprocessing*, nous disposon d'un texte nettoyé. L'étape de vectorisation varie en fonction du modèle utilisé par la suite.

4.1.2 Modèles simples et résultats

Suite à l'application du preprocessing, à la séparation du dataset de texte, plusieurs algorithmes de machine learning ont été entraînés via crossvalidation et testés sur le dataset de test :

- LogisticRegressionClassifier
- RandomForestClassifier
- DecisionTreeClassifier
- KNeighborsClassifier

La comparaison des résultats sur la métrique *weighted f1-score* est proposée ci-dessous:

[95]: `<pandas.io.formats.style.Styler at 0x7fcb965ad780>`

Pour les algorithmes de machine learning simples, sans recherche d'hyperparamètres idéaux, c'est le modèle *LogisticRegressionClassifier* qui obtient le meilleur *weighted f1-score* avec une valeur de 77.1%, juste avant le *RandomForestClassifier* avec son score de 76.0%. Nous notons que de manière générale, certaines classes sont mal catégorisées. En utilisant un crosstab, nous pouvons trouver les produits pour lesquels il y a confusion. Nous analysons ci-dessous les résultats de *LogisticRegressionClassifier* :

- Catégorie `10` (Livre d'occasion) souvent confondue avec `2705` (Livre neuf) et `2403` (Revue)
- Catégorie `40` (Jeu console) souvent confondue avec `10` (Livre occasion) et `2462` (Jeu oldschool)
- Catégorie `1280` (Déguisement) souvent confondue avec `1281` (Boîte de jeu) et `1140` (Figurine)

La confusion apparaît souvent entre des produits du même type. Ce risque avait été évoqué lors de l'analyse exploratoire de nos données cibles : les produits se décrivent dans un vocabulaire similaire.

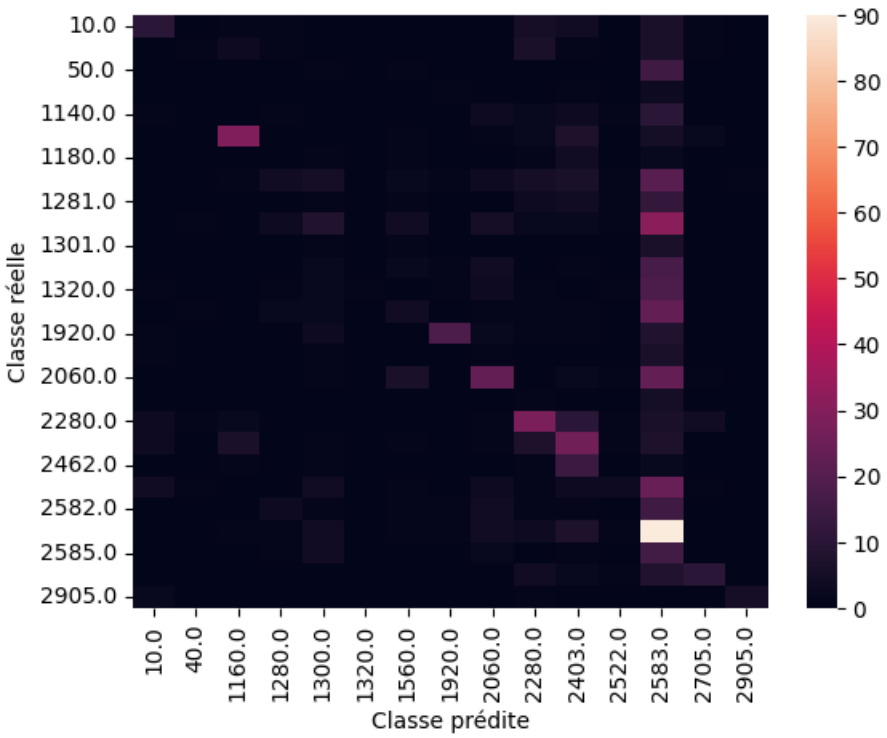
[96]: `<pandas.io.formats.style.Styler at 0x7fcb96575cf0>`

4.2 Images

Feature PCA + Modèle SVC : Dans une deuxième partie de l'étude de la modélisation par ML des images, il est opéré une SVC (avec une réduction de dimensions par PCA) car il est le modèle le plus efficace pour la reconnaissance de formes. Le score obtenu a été de 0.3. Ce score est différent de ceux d'Olga car les images sont plus nombreuses plus de 4000 images et ce qui est décisif c'est le paramètre du SVC, kernel = 'rbf'.Ce score médiocre n'est pas suffisant et un modèle plus avancé a été proposé qui est l'arbre de décision boosté. Olga qui a notamment travaillé sur les modèles de classification de ML a elle aussi obtenu des résultats non satisfaisants. Elle a quant à elle réalisé une cross validation avec des grilles et elle a travaillé sur les métriques. En fait les modèles de classification de Machine learning ne sont pas assez puissants pour notre problèmatique de classification d'images et il faut passer au deep learning.

La matrice de confusion en heatmap est la suivante :

[97]:



5 Modèles de deep-learning

Dans la précédente partie, des techniques simples de modélisation ont été mises en place. Afin d'améliorer notre métrique *weighted f1-score*, des modèles de *deep-learning*, plus complexes et longs à entraîneer sont utilisés. Finalement, ces modèles seront fusionnés pour exploitier au mieux les performances de chacun d'entre eux.

5.1 Textes

Les textes subissent la même étape de *preprocessing* que pour les modèles de *machine learning*.

5.1.1 Modèles testés

Modèle neuronal dense Le modèle Neural_Simple se base sur la tokenisation *tf-idf* de nos textes. Les entrées sur modèle sont donc une matrice représentant le score *tf-idf* de chaque mot de chaque texte, si ce mot est dans le vocabulaire de notre tokenizer.

Layer (type)	Output Shape	Param
text_dense_1 (Dense)	(None, 54)	270054
text_drop_1 (Dropout)	(None, 54)	0
text_output (Dense)	(None, 27)	1485

Seulement 271,539 paramètres sont à entraîner pour ce modèle.

Modèle neuronal embedding Le modèle *Neural_Embedder*, a la particularité de présenter une couche d'embedding qui nous permet de fournir directement un ensemble de mots à ce réseau de neurones. La vectorisation pour ce réseau gère 50 000 mots et seuls les 500 plus fréquents sont conservés.

Layer (type)	Output Shape	Param
text_input (Embedding)	(None, None, 100)	5000000
text_average (GlobalAverage Pooling1D)	(None, 100)	0
text_drop_1 (Dropout)	(None, 100)	0
text_dense_1 (Dense)	(None, 54)	5454
text_drop_2 (Dropout)	(None, 54)	0
text_output (Dense)	(None, 27)	1485

Pour ce modèle, cette couche est conçue via un le tokenizer de keras. Le nombre de paramètres à entraîner est ici bien plus important, avec près de 5 millions de paramètres.

5.1.2 Résultats des modèles textes

Avant de présenter les scores de ces deux modèles, intéressons nous à leurs entraînement afin de vérifier que nous n'avons pas un sur-apprentissage trom important lors de cette phase.

```
HBox(children=(Image(value=b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x01Q\x00\x00\x00\xdc\x
```

Les courbes sont celles de l'apprentissage (traits pleins) et de la validation (traits pointillés) pour le modèle *neural_simple* (en jaune) et le modèle *neural_embedding* (en vert).Il y a un un léger sur-apprentissage, mais les *callbacks* ont permis de réduire le taux d'apprentissage dans le plateau puis de stopper l'entraînement après une certaine période de patience.

Les *weigthed f1-scores* des modèles sont ceux des données de tests, totalement indépendantes des données d'entraînement et de validation.

[99]: `<pandas.io.formats.style.Styler at 0x7fcb965af370>`

Les *weighted f1-score* obtenus par ces modèles de *deep learning* sont meilleurs que ceux de *machine learning*. Ici, le modèle avec la couche *embedding* est de presque 5 points plus performant que le modèle *LogisticRegressionClassifier*. Les produits propices aux erreurs de classification sont en revanche similaires et ce, pour les mêmes raisons. Les modèles de textes testés se trompent sur les mêmes catégories, il est probable qu'en les combinant, les erreurs restent identiques. La concatenation du modèle de texte avec le prochain modèle d'analyse des images pourrait pallier ce problème.

5.2 Images

5.2.1 Modèle CNN

5.2.2 Modèle VGG16

[100]: `<pandas.io.formats.style.Styler at 0x7fcb965ad4b0>`

[101]: <pandas.io.formats.style.Styler at 0x7fcb965af4f0>

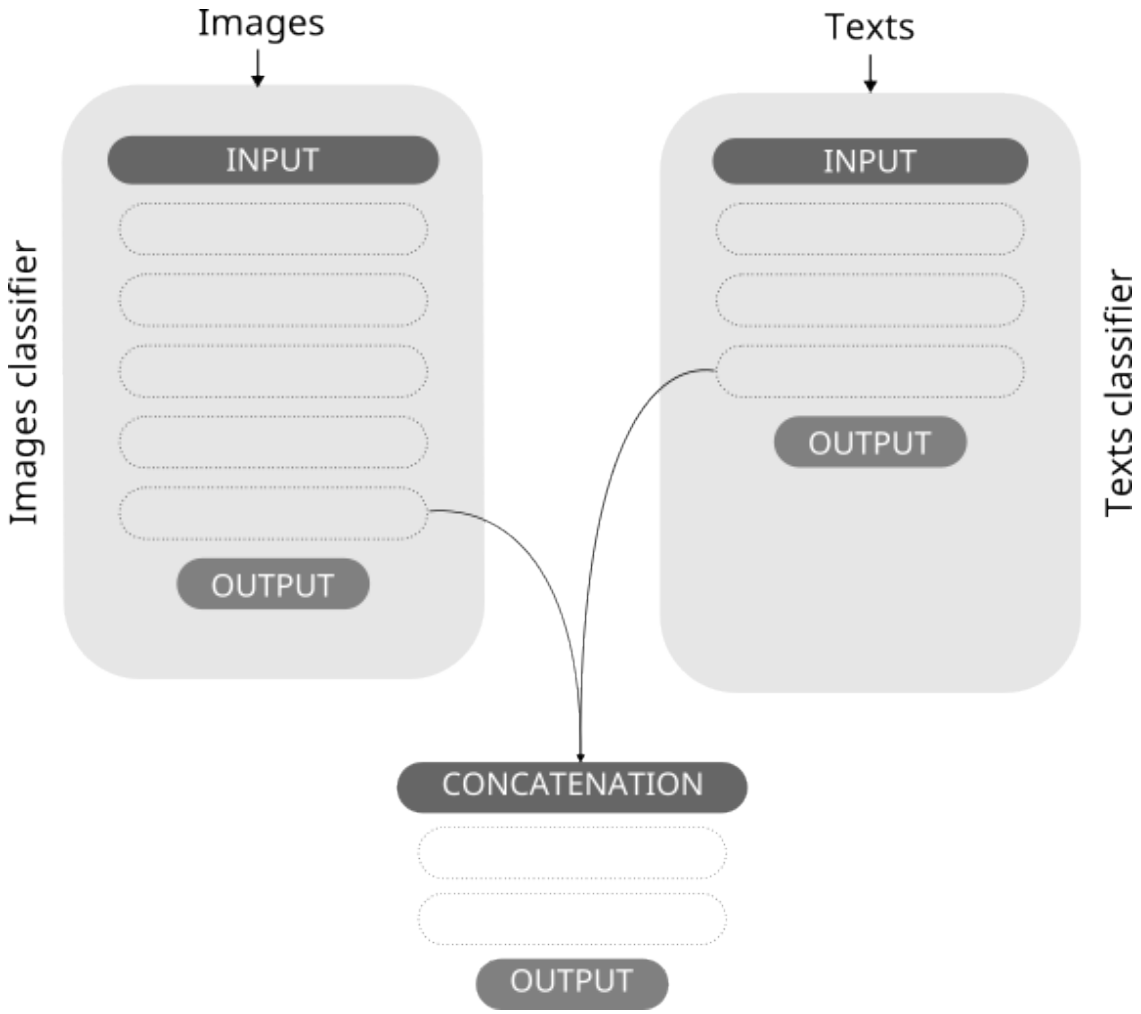
5.3 Fusion des modèles images et textes

Les deux modèles peuvent fonctionner indépendemment. Cependant, il est possible de mutualiser leurs capacités en fusionnant ces deux modèles en un unique modèle de classification.

5.3.1 Fusion par concaténation de couches

La fusion est mise en place par concaténation de couches du modèle de texte et du modèle d'image. Cette concaténation est faite sur les avant-dernières couches de nos modèles : ces couches comportent un maximum d'informations. Les autres couches des modèles pré-entrainés sont quant à elles *freezées*. Des couches denses sont ajoutées suite à la concaténation pour obtenir une classification sur 27 classes. Ci-dessous, un graphique simplifié du fonctionnement de cette concaténation.

[102]:



En appliquant cette méthode à nos modèles `text_neural_embedding` et `image_vgg16`, près de 15 millions de paramètres ne sont plus à entraîner. La concaténation a bien lieu au niveau des couches

précédant les couches denses de classication.

Layer (type)	Output Shape	Param #	Connected to
vgg16_input	[(None, None, None,3)]	0	[]
vgg16_image_CNN_Lenet_1	(None, None, None,512)	14714688	['vgg16_input[0][0]']
global_average_pooling2d_image_CNN_Lenet_1	(None, 512)	0	['vgg16_image_CNN_Lenet_1[0][0]']
flatten_image_CNN_Lenet_1	(None, 512)	0	['global_average_pooling2d_image_C
dropout_image_CNN_Lenet_1	(None, 512)	0	['flatten_image_CNN_Lenet_1[0][0]'
text_dense_1_input	[(None, None)]	0	[]
dense_image_CNN_Lenet_1	(None, 32)	16416	['dropout_image_CNN_Lenet_1[0][0
text_dense_1_text_neural_simple_0	(None, 54)	270054	['text_dense_1_input[0][0]']
dropout_1_image_CNN_Lenet_1	(None, 32)	0	['dense_image_CNN_Lenet_1[0][0]']
text_drop_1_text_neural_simple_0	(None, 54)	0	['text_dense_1_text_neural_simple_
dense_1_image_CNN_Lenet_1	(None, 54)	1782	['dropout_1_image_CNN_Lenet_1[0
fusion_concat	(None, 108)	0	['text_drop_1_text_neural_simple_
fusion_dense1	(None, 128)	13952	['fusion_concat[0][0]']
fusion_drop1	(None, 128)	0	['fusion_dense1[0][0]']
fusion_output	(None, 27)	3483	['fusion_drop1[0][0]']

Total params: 15,020,375
Trainable params: 19,217
Non-trainable params: 15,001,158

Suite à l'entrainement des avant-dernières et dernieres couches des modèles de textes et d'image ainsi que les couches denses du modèle concaténé, on remarque que le modèle est rapidement en overfitting.

```
HBox(children=(Image(value=b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x01M\x00\x00\x00\xda\x
```

Suite à la concaténation des modèles d'images et de textes, les résultats du modèle sont encourageants avec un *weighted f1-score* supérieur de 0.8%, soit 82.7%.

[104]: `<pandas.io.formats.style.Styler at 0x7fcb96425150>`

Plusieurs observations sont à faire à ce stade :

- Le modèle concaténé n'est pas impacté par les performances réduites du modèle d'image.
  - Toutes les catégories dépassent le score de 54%
  - Une catégorie sur trois dépasse le score de 90%
  - Un catégorie atteint un score de 99%
- Le modèle concaténé s'aide du modèle d'image pour catégoriser les produits où le modèle de texte sous-performait :
  - La catégorie 1080 (Jeu Plateau) gagne 25 points
  - La catégorie 2705 (Livre neuf) gagne 23 points

[105]: `<pandas.io.formats.style.Styler at 0x7fcb965aebf0>`

6 Conclusion

Le projet, parti d'une phase d'exploration, de la création de modèles de classification simple, aboutit à un modèle complexe issu de la fusion de modèles de *deep learning*. Le *weighted f1-score* atteint pour ce dernier modèle est de 82.9%. Pour rappel, un modèle de classification aléatoire atteindrait seulement un score de 3.7%.

L'état de l'art évolue rapidement et de nouvelles techniques pourraient être mises en place.

Le modèle de texte par exemple pourrait être doté d'une couche d'embedding pré-entrainée, par exemple celle issue de CamemBERT : son entraînement a été effectué sur 138GB de données en français. Sur un tel modèle, l'entraînement des 110 millions de paramètres serait extrêmement coûteux en termes de temps et d'infrastructure. Cette couche remplacerait alors notre couche seulement entrainée sur nos données, et liée à une couche dense de classification pourrait améliorer les scores de notre modèle de texte.

Le projet est un projet mêlant de l'analyse de texte et du traitement d'images : des notions poussées de deep-learning sont nécessaires à la compréhension et l'implémentation de telles techniques. De nombreux challenges sont apparus tout au long de ce projet :

- L'accès à des ressources de calcul de type GPU ou TPU nous a été quasi impossible, notamment via Google Collab. Tous les calculs ont été réalisés sans GPU.
- L'accès aux 84916 images, stockées dans un Google Drive et nécessaires à l'entraînement du modèle d'images, était érattique : de nombreuses coupures de ce lien entre Google Drive et Goolg Collab ont entraîné ici aussi une grande perte de temps et une grande frustration.
- Des notions relatives aux générateurs, au *deep learning* sont rapidement nécessaires pour la réalisation de ce projet. Il a été délicat d'assimiler rapidement ces méthodes pour les mettre en pratique.
- Les implications et les attentes de chaque membre de l'équipe par rapport à ce projet sont variées. La répartition du travail a été de ce fait déséquilibrée.

Finalement, ce projet a été très intéressant, car complexe et faisant appel à des notions avancées mêlant le traitement de textes et le traitement d'image. L'exploration de données, le travail de groupe, les différentes implémentations et sprints ont fait de ce projet un projet d'entreprise.