

# Rakuten

## Projet rakuten

Classification de produits à partir de textes et d'images

Karim Hadjar

Julien Noel du Payrat

Mathis Poignet

*NOV 23 Bootcamp data science*

<b>Introduction.....</b>	<b>4</b>
Objectif du projet.....	4
Contexte.....	4
Fichiers fournis.....	4
Métrique utilisée pour l'évaluation de la performance.....	5
Intérêts du projet.....	5
Equipe.....	6
<b>Architecture.....</b>	<b>7</b>
<b>Exploration des données.....</b>	<b>8</b>
Aperçu initial.....	8
Statistiques d'usage.....	8
Vérification des doublons.....	8
Analyse de la variable cible.....	9
Valeurs manquantes.....	10
HTML dans les données textuelles.....	10
Analyse des langues.....	11
Fréquence des mots sur designation et text.....	13
Fréquence des mots par type de produit.....	14
Les images.....	16
<b>Preprocessing du texte.....</b>	<b>18</b>
Traduction du texte.....	18
Préparation du texte.....	19
<b>Preprocessing des images.....</b>	<b>21</b>
<b>Rééquilibrage des classes.....</b>	<b>22</b>
<b>Modélisation Classification du Texte.....</b>	<b>23</b>
Modèles et Performances pour le texte.....	24
Observations.....	27
Conclusion.....	27
<b>Modélisation images avec LeNet.....</b>	<b>28</b>
1. Impact du preprocessing.....	29
2. Impact des hyper-paramètres.....	30
3. Entraînement complet d'un modèle basé sur les enseignements tirés des sections 1 et 2.....	32
Conclusion.....	32
<b>Modélisation images avec VGG16.....</b>	<b>33</b>
Le meilleur modèle.....	34
Conclusion.....	36
<b>Fusion des modèles de texte et d'image.....</b>	<b>37</b>

1. VotingClassifier.....	37
2. StackingClassifier.....	38
3. VotingClassifier avec modèle texte basé sur CamemBERT.....	39
Conclusion.....	42
<b>Conclusion générale.....</b>	<b>43</b>
Difficultés rencontrés lors du projet.....	43
Bilan.....	43
Suite du projet.....	44
<b>Bibliographie.....</b>	<b>44</b>

# Introduction

## Objectif du projet

L'objectif du projet est de cataloguer des produits selon un code type désignant le produit. La prédiction du type doit se faire à partir de données textuelles (désignation et description du produit) ainsi que de données visuelles (image du produit).

## Contexte

Ce projet s'inscrit dans le challenge Rakuten France Multimodal Product Data Classification, les données et leur description sont disponibles à l'adresse :  
<https://challengedata.ens.fr/challenges/35>

- Les données textuelles : ~60 mb
- Données images : ~2.2 gb
- 99k données avec plus de 1000 classes.

## Fichiers fournis

- X\_train.csv: Contient les variables explicatives destinées à l'entraînement des modèles.
  - index (nb entier): Index du produit.
  - designation (object: string): Designation courte du produit
  - description (object: string, optionnel): Description du produit. Ce champ est optionnel. Tous les produits n'ont pas de description
  - productid (int64): L'id du produit
  - imageid (int64): L'id de l'image du produit
- images.zip: Une fois extrait, un dossier contenant les images des produits. La nomenclature utilisée permet de faire la jonction avec les produits. Chaque fichier d'image se présente sous la forme:  
image\_<imageid>\_product\_<productid>.jpg. Les images sont répartis en deux sous-dossiers:
  - image\_train: Les images correspondants à X\_train.csv
  - image\_test: Les images correspondants à X\_test.csv
- Y\_train.csv: Contient la variable cible à prédire. A savoir le type du produit.

- index (nb entier): Index du produit. Permet de faire la jonction avec X\_train.csv
- prdtypecode (nb entier): Le type du produit
- X\_test.csv: Contient les variables explicatives destinées à l'évaluation des modèles. Sa structure est identique à celle de X\_train.csv. Ce fichier étant fourni dans le contexte du challenge Rakuten, nous n'avons pas obtenu de fichier Y\_test.csv qui nous permettrait de comparer nos performances à celles des participants au challenge. Il est probable qu'on doive se contenter de scinder le dataset d'entraînement en une partie entraînement et une partie test.

Note: Nous avons légèrement modifié les entêtes des fichiers d'origine en y ajoutant le titre de colonne index pour la première colonne

## Métrique utilisée pour l'évaluation de la performance

La métrique weighted-F1 score a été choisie dans le cadre du challenge.

## Intérêts du projet

Étant réalisé dans le cadre de la formation Datascientest, c'est l'opportunité pour nous de découvrir et mettre en applications des techniques de machine learning avancées telles que:

- Computer vision
  - réseaux de neurones convolutifs
- NLP
- Modèles multimodaux
- Deep learning

Bien que ce projet utilise le dataset d'un site de vente, il est assez générique de par la nature de son sujet qui pourrait se résumer ainsi: Attribuer une classe à un objet à partir d'une description textuelle et d'une image. On pourrait imaginer toutes sortes de déclinaisons:

- Commerciales

- Classification automatique de produits mis en vente sur un site de e-commerce (le but original du challenge).
  - Assister un utilisateur dans le choix d'une catégorie lors de la mise en vente de son produit. Quand on connaît le grand nombre de catégories typiquement proposées sur les sites de e-commerce, on peut imaginer la confusion des utilisateurs.
  - On pourrait même imaginer un système plus coercitif qui impose une catégorie en fonction du contenu pour éviter les erreurs de catégorisation.
  - Fournir une recherche intelligente des produits qui puisse automatiquement traduire une description de produit saisie en classe de produits et ainsi produire des résultats pertinents.
- Génériques
  - Un moteur de recherche d'objets à partir de description ou même d'images.
  - Moteur de recherche d'image à partir d'une description et réciproquement (si les modèles individuels de nlp et de computer vision sont suffisamment performants)

## Equipe

- Mathis Poignet
- Karim Hadjar
  - Je possède une expérience dans la création de tableaux de bord et l'utilisation d'outils tels qu'Excel et Power BI. Néanmoins, j'adopte une approche empirique pour l'exploration des données et la sélection des visualisations.
- Julien Noel du Payrat ([GitHub](#) / [LinkedIn](#))
  - J'ai un background de développeur depuis plus de 15 ans, en revanche, je fais mes premières armes en data science. Je n'ai donc pas d'expérience préalable dans des projets similaires à celui-ci.

# Architecture

Nous avons organisé le dépôt git comme ceci:

- **data:** Contient les csv de données fournies. Nous n'avons pas ajouté les images sur git, le dossier étant trop volumineux. Un zip est toutefois accessible publiquement sur google drive: [images.zip](#)
- **notebooks** contient les notebooks à exécuter dans l'ordre listé ici. En effet, la plupart des notebooks produisent des résultats sur lesquels d'autres notebooks s'appuient.
  - **data-exploration:** Exploration des données et dataviz
  - **data\_preprocessing\_traduction\_fr:** La traduction des textes en français
  - **data\_preprocessing\_images:** Le zoom des images
  - **data\_preprocessing\_resampling:** Le rééquilibrage des classes
  - **Data\_preprocessing-text-stopWord-Stemming:** Filtrage des stop words, tokenization et lemmatization des textes
  - **data-modeling-images-1:** La modélisation d'image avec LeNet
  - **data-modeling-images-3:** La modélisation d'image avec VGG16
  - **data-modeling-text-1-TF-IDF:** La modélisation de texte avec TFIDF
  - **data-modeling-text-1bis-TF-IDF:** La suite de la modélisation de texte avec TFIDF
  - **data-modeling-text-2-Cbow:** La modélisation de texte avec Cbow
  - **data-modeling-text-3-Skip Gram:** La modélisation de texte avec Cbow
  - **data-modeling-text-4-RNN-GRU:** La modélisation de texte avec RNN GRU
  - **data-modeling-text-5-Fasttext:** La modélisation de texte avec Fasttext
  - **data-modeling-text-6-CamenBERT:** La modélisation de texte avec Camembert
  - **data-modeling-text-6 retrain-CamenBERT:** La suite de la modélisation de texte avec Camembert pour ré entrainé à partir d'une sauvegarde
  - **data-modeling-fusion:** La fusion des modèles de textes et d'images
- **output:** Contient les résultats des notebooks chacun dans un sous dossier nommé de façon identique au notebook correspondant
- **assets:** Contient quelques ressources images utilisées par les notebooks
- **reports:** Contient le présent rapport au format PDF

# Exploration des données

## Aperçu initial

Nous avons chargé le fichier **X\_train.csv** et effectué une première inspection des données.

index	designation	description	productid	imageid
0	Olivia: Personalisiertes Notizbuch / 150 Seite...	NaN	3804725264	1263597046
1	Journal Des Arts (Le) N° 133 Du 28/09/2001 - L...	NaN	436067568	1008141237
2	Grand Stylet Ergonomique Bleu Gamepad Nintendo...	PILOT STYLE Touch Pen de marque Speedlink est ...	201115110	938777978
3	Peluche Donald - Europe - Disneyland 2000 (Mar...	NaN	50418756	457047496
4	La Guerre Des Tuques	Luc a des id&acute;es de grandeur. Il veut or...	278535884	1077757786
5	Afrique Contemporaine N° 212 Hiver 2004 - Doss...	NaN	5862738	393356830
6	Christof E: Bildungsprozessen Auf Der Spur	NaN	91920807	907794536
7	Conquérant Sept Cahier Couverture Polypro 240 ...	CONQUERANT CLASSIQUE Cahier 240 x 320 mm seyès...	344240059	999581347
8	Puzzle Scooby-Doo Avec Poster 2x35 Pièces	NaN	4239126071	1325918866
9	Tente Pliante V3s5-Pro Pvc Blanc - 3 X 4m50 - ...	Tente pliante V3S5 Pro PVC 500 gr/m <sup>2</sup> - 3 x 4m5...	3793572222	1245644185

## Statistiques d'usage

- Le champ **description** contenait un grand nombre de valeurs nulles (35%), ce qui était attendu pour un champ optionnel.
- Les autres colonnes ne présentaient pas de valeurs nulles.

Nous avons également chargé le fichier **Y\_train.csv** et constaté qu'il n'y avait pas de valeurs manquantes. Les deux fichiers avaient le même nombre d'entrées, ce qui indiquait une correspondance totale entre les deux. Correspondance qui s'est vérifiée lors de la fusion des variables explicatives et la variable cible dans un DataFrame unique.

## Vérification des doublons

Le dataset initial ne présentait pas de doublons de lignes, mais des doublons ont été observés pour les variables explicatives individuellement :

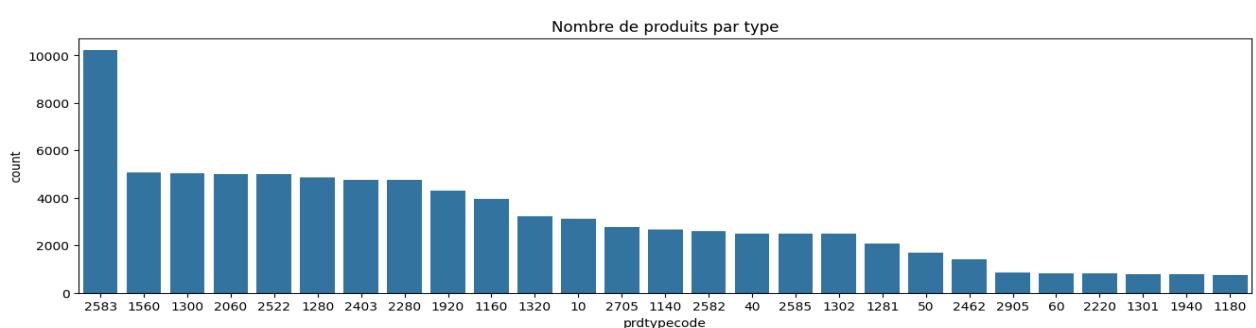
- Doublons de **designation** : 2651
- Doublons de **description** : 37409 (réduit à 7610 après élimination des valeurs manquantes)

1294 produits partageaient des couples **designation**/**description** communs malgré des identifiants de produits distincts. Ainsi, le compte réel des doublons était :

- Doublons de **designation** : 1357
- Doublons de **description** : 6316

La décision de suppression des doublons a nécessité une évaluation en fonction de la variable cible.

## Analyse de la variable cible



La variable cible comptait 27 classes possibles. En observant la répartition des produits par classe, nous avons noté des disparités significatives. Certaines classes étaient sous-représentées, tandis que d'autres étaient sur-représentées. Par exemple, la classe 60 était sous-représentée, tandis que la classe 2583 comptait environ deux fois plus d'instances que les classes adjacentes.

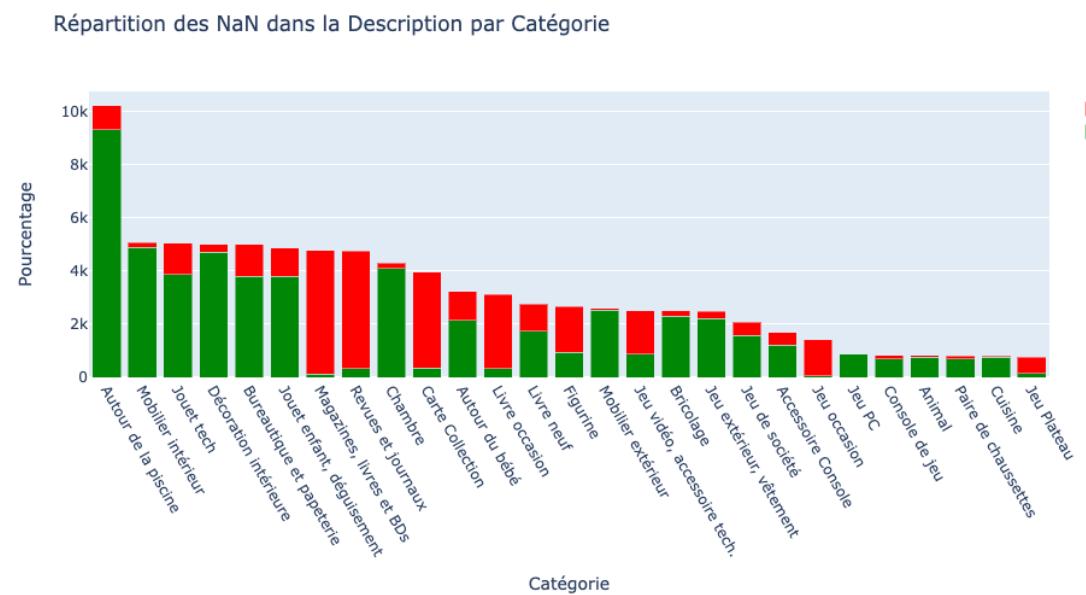
Les doublons dans les données suivaient une distribution similaire à celle des produits par classe. Bien que la suppression des doublons aurait pu être envisagée sans altérer significativement la distribution initiale, cela aurait nécessité une vérification préalable pour s'assurer de l'identité des images associées. Étant donné le faible nombre de doublons (1294), nous avons choisi de les conserver.

Afin de rendre les labels des classes plus explicites, nous avons construit un dictionnaire associant les codes à leurs labels respectifs, que nous avons utilisé par la suite dans les visuels.

## Valeurs manquantes

La proportion de valeurs manquantes dans le champ **description** s'élevait à 35%. Nous avons envisagé plusieurs options pour les traiter:

1. Supprimer les lignes les contenant: Ca aurait représenté une perte de données non négligeable en plus d'aller à l'encontre de la caractéristique optionnelle du champs **description**
2. Les remplacer par des chaînes de caractères vides: Une approche certe simple mais au conséquence incertaines sur les modèles de prédictions
3. Concaténer les variables **designation** et **description** dans une troisième variable qui aurait donc au moins contenu **designation**



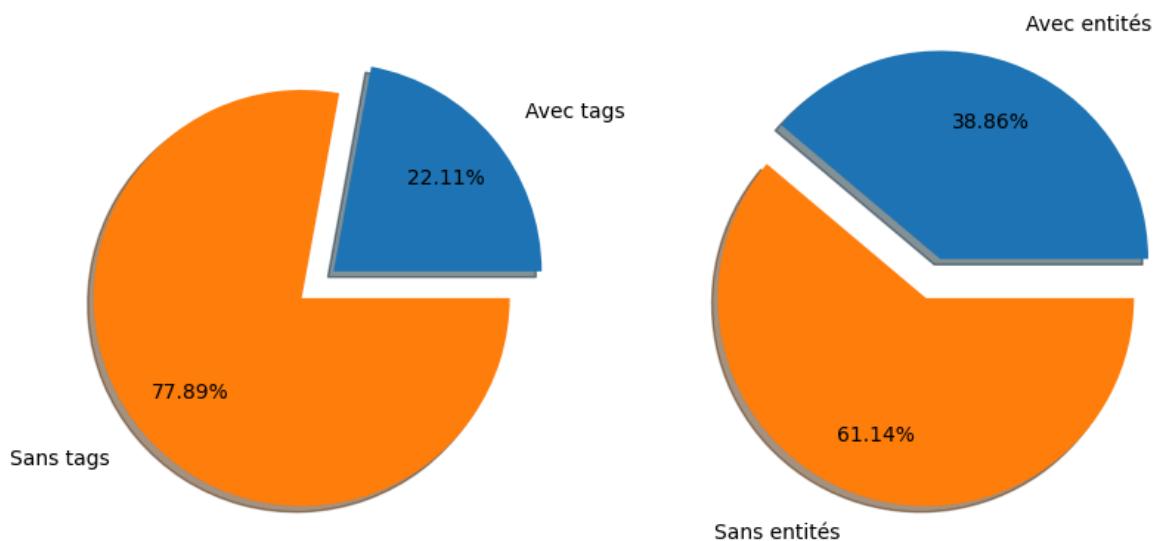
En affichant un graphique en barre, nous avons confirmé que l'option 1 était inenvisageable. Supprimer les lignes avec des valeurs nulles dans **description** serait presque revenu à supprimer certaines catégories. Par conséquent, nous avons préféré opter pour la troisième option.

## HTML dans les données textuelles

En parcourant les données tabulaire, nous avons observé un grand nombre de textes contenant du html soit sous forme de tags (ex: **<b></b>**, **<p></p>**, ...), soit sous forme de caractères encodés (ex: **&#FA;**, **&**, ...).

Notre analyse a révélé une proportion significative de **descriptions** contenant du html au contraire des **designations** pour qui la présence de HTML restait anecdotique.

Proportion de descriptions avec des tags ou des entités HTML (hors NA)



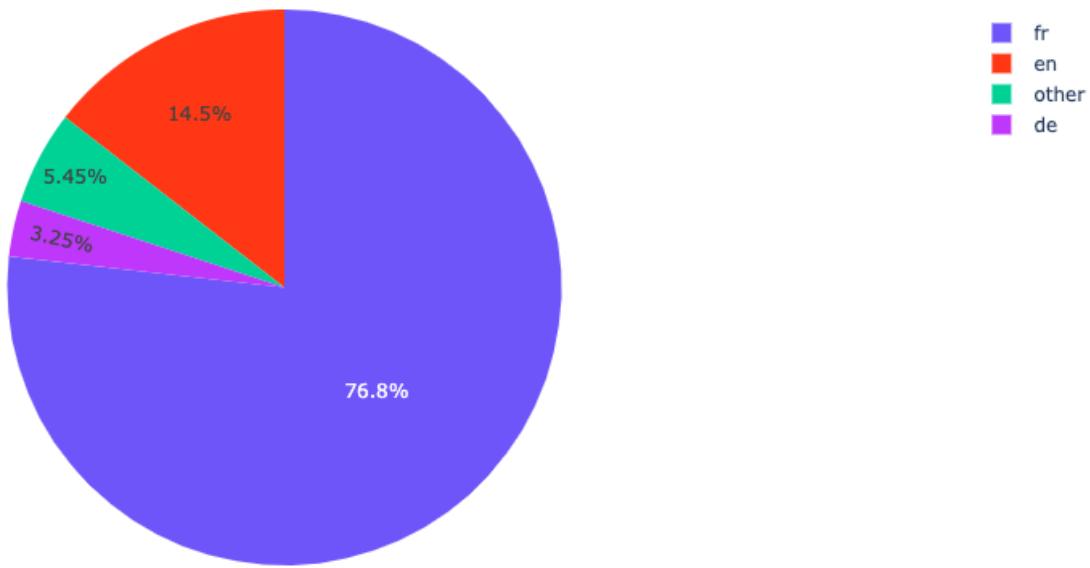
Dans le but de faciliter l'analyse des langues et de la fréquence des mots, nous avons créé une fonction permettant de supprimer le html et remplacer les caractères encodés que nous avons appliqué aux variables **designation** et **description**.

## Analyse des langues

Lors de nos explorations de données, nous avons également remarqué la présence de texte dans plusieurs langues, principalement en français, anglais et allemand.

Pour clarifier la situation, nous avons décidé d'utiliser la librairie **langdetect** pour détecter la langue la plus probable de chaque texte. Nous avons ajouté deux colonnes **lang** et **langprob**, contenant respectivement le code à deux lettres de la langue et la probabilité de détection. Ainsi, nous avons pu confirmer que la fiabilité de la détection était bonne, l'écrasante majorité des probabilités étant très élevées.

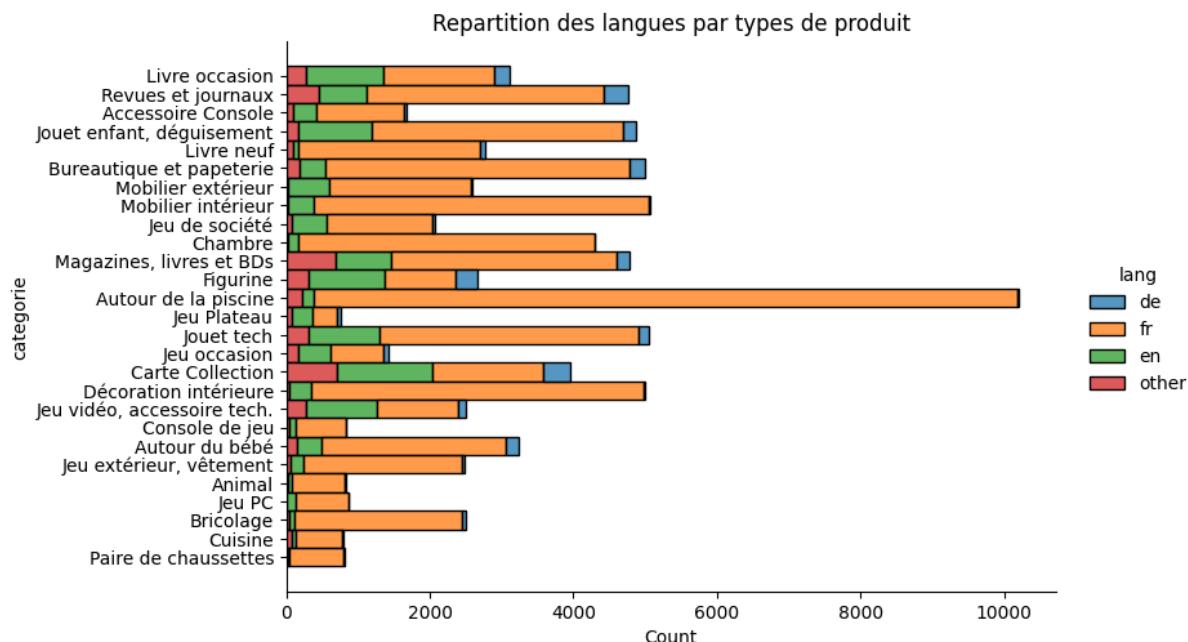
Nous avons ensuite généré un camembert représentant la proportion de chaque langue.



La plupart des observations étaient en français, anglais ou allemand, représentant plus de 85% du contenu total.

Nous avons également tenu à analyser la proportion de langues par produit ce qui nous a permis de constater que le français constituait la majorité relative dans chaque type de produit. Les modèles NLP étant en général orienté vers une langue particulières, nous avons alors envisagé nos options:

1. Supprimer les données dans une langue autre que le français: Cela aurait impliqué une perte de donnée de 25% et aurait pu déséquilibrer encore davantage les classes déjà minoritaires
2. Traduire toutes les langues vers l'anglais qui est le mieux supporté par les modèles NLP: Ca aurait représenté un travail de traduction massif qui aurait dépassé les capacité gratuites allouées par les API de traductions
3. Traduire toutes les langues vers le français: Ca réduirait la proportion de données à traduire à 25% tout en préservant au mieux la qualité originale des textes. C'est l'option que nous avons adoptée.



## Fréquence des mots sur designation et text

Pour approfondir notre analyse des données, nous avons extrait la fréquence des mots des variables textuelles **designation** et **text** (la concaténation de **designation** et **description**).

Nous avons ajouté deux nouvelles colonnes, **mots\_designation** et **mots\_text**, contenant la liste des mots extraits de chaque colonne respective. Cette approche nous a permis d'observer les différences de fréquence des mots entre les deux colonnes.

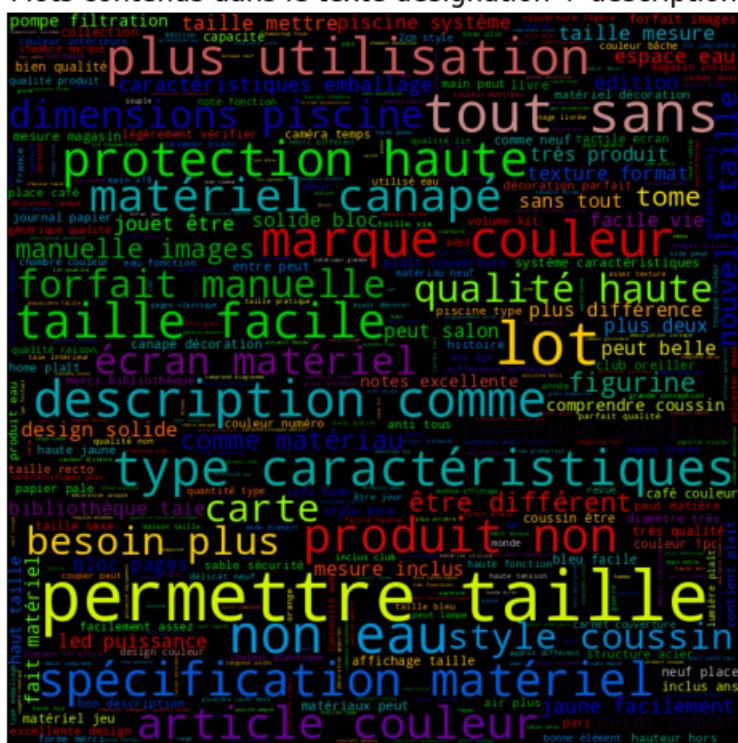
L'extraction des mots s'est effectuée avec succès, et nous avons pu observer un premier nuage de mots représentant la fréquence des mots dans la colonne **designation**. Nous avons remarqué une nette prédominance des mots français, avec quelques mots anglais en arrière-plan.

En examinant ensuite la fréquence globale des mots dans la colonne **text**, nous avons confirmé la prédominance du français, les mots anglais reculant encore plus vers l'arrière-plan. Nous avons également observé un changement dans les mots les plus identifiables, indiquant que le champ **description** apportait une information complémentaire non redondante par rapport à **designation**.

#### Mots contenus dans la designation

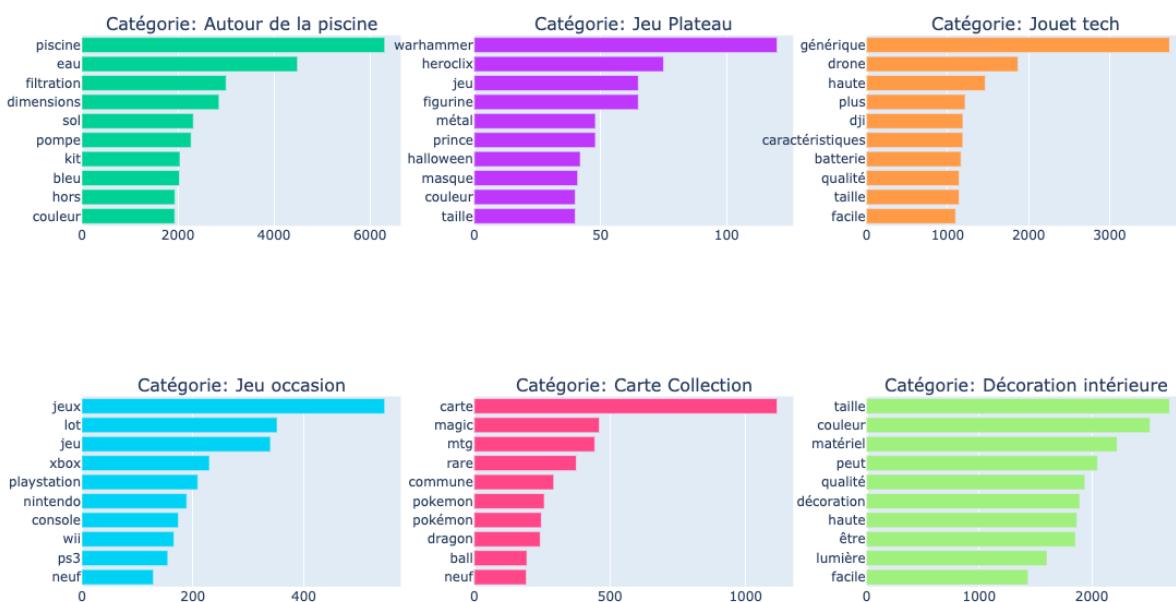


Mots contenus dans le texte designation + description



## Fréquence des mots par type de produit

Nous avons affiché une série de graphiques en barres, un par catégorie, représentant les occurrences des dix mots les plus fréquents dans chaque catégorie respective. Par soucis de concision, nous ne les avons pas tous affichés ici mais le lecteur curieux pourra les consulter intégralement dans le notebook data-exploration.



Nous avons observé dans la plupart des catégories des mots reflétant leur thème, mais aussi certains mots génériques tels que "peut", "plus", "être", probablement en raison d'un filtrage insuffisant des stop words.

D'autres termes typiques d'un vocabulaire produit, tels que "dimension", "taille", "longueur", ainsi que des mots classiques du marketing de vente comme "haute", "qualité" et "facile", ont également été repérés. On s'est interrogé sur l'éventualité de filtrer ce genre de mots qui ne semblaient pas apporter d'information sur la nature des produits.

Nous avons également remarqué des catégories particulières, comme "livre neuf", dont les mots ne reflétaient pas toujours la nature du produit, certains mots ressemblant davantage à des titres de livres. Curieusement, la catégorie "livre occasion" ne présentait pas cette caractéristique, avec un vocable correspondant davantage au monde de l'édition.

Enfin, nous avons affiché des nuages de mots par catégorie, mettant en évidence certains mots plus en arrière-plan qui n'apparaissaient pas dans le top 10. Cette

représentation a souligné l'importance d'un processus de sélection de features efficace pour identifier les mots les plus explicatifs. Voici, un exemple:



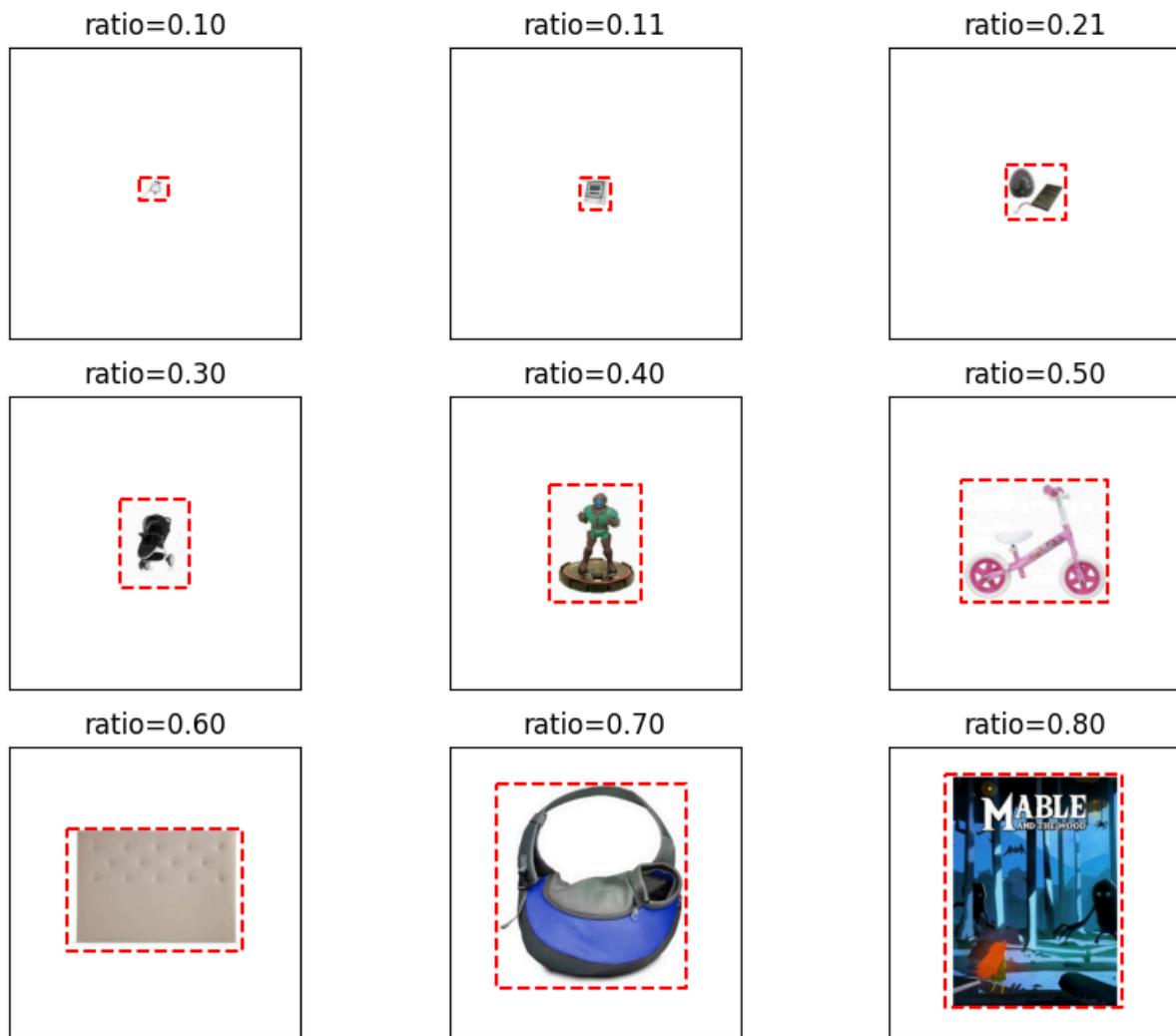
## Les images

Pour commencer, nous avons extrait les images du fichier image.zip, en vérifiant leur présence et leur taille. Nous avons constaté que toutes les images étaient présentes et qu'elles avaient bien une taille de 500x500 pixels.

Au passage, nous avons remarqué beaucoup d'images qui semblait être une petite image entourée de beaucoup de blanc. Nous avons donc calculé la bounding box de

l'image centrale pour chaque image à l'aide d'une fonction maison. Nous avons également vérifié que chaque image était non vide ce qui était le cas à l'exception de quatre d'entre elles.

Nous avons ensuite vérifié visuellement quelques exemples d'images avec leur bounding box.



Environ 20% des images avaient un ratio inférieur à 0.8, ce qui suggère qu'elles pourraient bénéficier d'un zoom.

Pour faciliter le preprocessing des images dans un futur notebook, nous avons copié les colonnes pertinentes dans un nouveau DataFrame `df_img`, que nous avons enregistré dans un fichier CSV.

# Preprocessing du texte

## Traduction du texte

La traduction du texte à partir du DataFrame a été réalisée en utilisant la bibliothèque googletrans.

### Initialisation du Traducteur

La première étape consiste à initialiser l'objet Translator de la bibliothèque googletrans. Cet objet sert de point d'entrée pour effectuer les traductions.

### Fonction de Traduction

La fonction `translate_text` est conçue pour traduire la colonne `text` d'un DataFrame vers le français. Elle prend en compte la langue d'origine de chaque texte (indiquée dans la colonne `lang`) et effectue la traduction uniquement si la langue d'origine n'est pas le français. Cette approche est justifiée par le fait que plus de 80% du jeu de données est déjà en français, ce qui minimise le nombre de traductions nécessaires et, par conséquent, le temps de traitement global. Il aurait sûrement été judicieux d'isoler les textes à traduire avant de les envoyer dans la fonction.

### Sécurisation de la Fonction

Plusieurs mesures de sécurisation ont été implémentées dans la fonction pour assurer sa robustesse :

1. Gestion des Refus de Google Translate : La fonction intègre un mécanisme de reprise après erreur pour gérer les éventuels refus ou erreurs de l'API Google Translate. En cas de ReadTimeout, la fonction tente de redémarrer la traduction à partir du dernier index sauvegardé. Ce mécanisme n'est pas parfait et des arrêts subsistent.

2. Sauvegarde Intermédiaire : Les traductions sont sauvegardées progressivement dans un fichier JSON spécifié par save\_path. Cette approche permet de ne pas perdre les traductions déjà effectuées en cas d'interruption du processus.
3. Gestion de la Longueur du Texte : Google Translate impose une limite sur la longueur du texte pouvant être traduit en une seule fois. La fonction vérifie la longueur du texte et le découpe en morceaux de 4000 caractères si nécessaire, traduisant chaque morceau séparément avant de les recombiner.

### **Choix de Traduire en Français**

Le choix de traduire tous les textes en français, plutôt qu'en anglais, repose sur deux considérations principales :

- Majorité de Textes en Français : Étant donné que plus de 75% du jeu de données est en français, traduire les textes restants en français simplifie le traitement ultérieur en unifiant la langue de l'ensemble des données.
- Optimisation du Temps de Traitement : La traduction en français réduit le volume de travail pour l'API de traduction et diminue le temps de traitement global. Bien que le temps de traitement dépasse 15 heures pour l'ensemble du jeu de données, cette approche reste plus efficace que de traduire dans une langue moins représentée dans le jeu de données.

## **Préparation du texte**

Les données textuelles ont été préparées en utilisant plusieurs techniques avant la modélisation :

**Nettoyage** : Suppression des caractères inutiles, des balises HTML, des espaces supplémentaires, etc.

**Tokenisation** : Division du texte en mots ou phrases (tokens) pour faciliter leur traitement.

**Suppression des mots vides** : Les mots très fréquents (comme "le", "et", "à") qui n'apportent pas de valeur significative pour l'analyse sont retirés.

**Normalisation** : Application de la lemmatisation ou du stemming pour réduire les mots à une forme de base.

**Vectorisation** : Transformation du texte en vecteurs numériques que le modèle peut traiter. Ceci peut être réalisé par des techniques comme Bag of Words, TF-IDF, word embeddings (Word2Vec, GloVe), ou des embeddings contextuels (BERT, GPT).

Voici les préparations testées lors du projet:

- **Texte non traduit avec Filtrage des stop words, WordNetLemmatizer, et TF-IDF :** Cette approche a permis de réduire le bruit dans les données textuelles et de concentrer l'analyse sur les termes significatifs.
- **Texte traduit avec CBOW (Continuous Bag of Words)** : Les textes ont été vectorisés en utilisant l'approche CBOW de Word2Vec, permettant de capturer le contexte des mots dans les vecteurs.
- **Skip Gram** : Une autre technique de Word2Vec, Skip Gram, a été utilisée pour prédire le contexte à partir des mots, offrant une alternative à CBOW.
- **Texte traduit tokenisé avec padding** : Cette méthode a préparé les données pour les modèles de réseaux de neurones récurrents (RNN) en normalisant la longueur des séquences de texte.
- **Texte traduit tokenisé avec pour BERT**: Transformation du texte en vecteurs numériques que le modèle BERT peut traiter (padding, limité à 512 tokens)

## Preprocessing des images

Pour le preprocessing des images, nous avons commencé par importer les données CSV prétraitées issues du notebook de data-exploration et les informations sur les images exportées depuis ce même notebook.

Ensuite, nous avons fusionné les informations sur les images dans le DataFrame principal en nous basant sur leurs index respectifs. Cette opération s'est déroulée sans problème, et aucune donnée manquante n'a été constatée.

Nous avons ensuite déplacé chaque image dans un sous-dossier propre à sa classe d'appartenance, et ajouté une colonne **imagefile** au Dataframe contenant leur chemin relatif.

Après cela, nous avons procédé au zoom des images qui en avaient besoin. Nous avons défini une fonction pour effectuer le zoom, puis nous avons vérifié visuellement que le zoom s'était bien déroulé en affichant des exemples d'images avant et après le zoom dont voici un échantillon:



Enfin, nous avons sauvegardé le DataFrame avec la colonne **imagefile** et exporté les images zoomées dans une nouvelle archive **images\_pre.zip**.

## Rééquilibrage des classes

Pour le rééquilibrage des données, nous avons commencé par importer les données prétraitées issues du notebook de preprocessing des images.

Ensuite, nous avons séparé les variables explicatives (X) et la variable cible (y). Nous avons conservé quelques variables supplémentaires dans X pour nous aider lors de l'analyse des résultats des modèles.

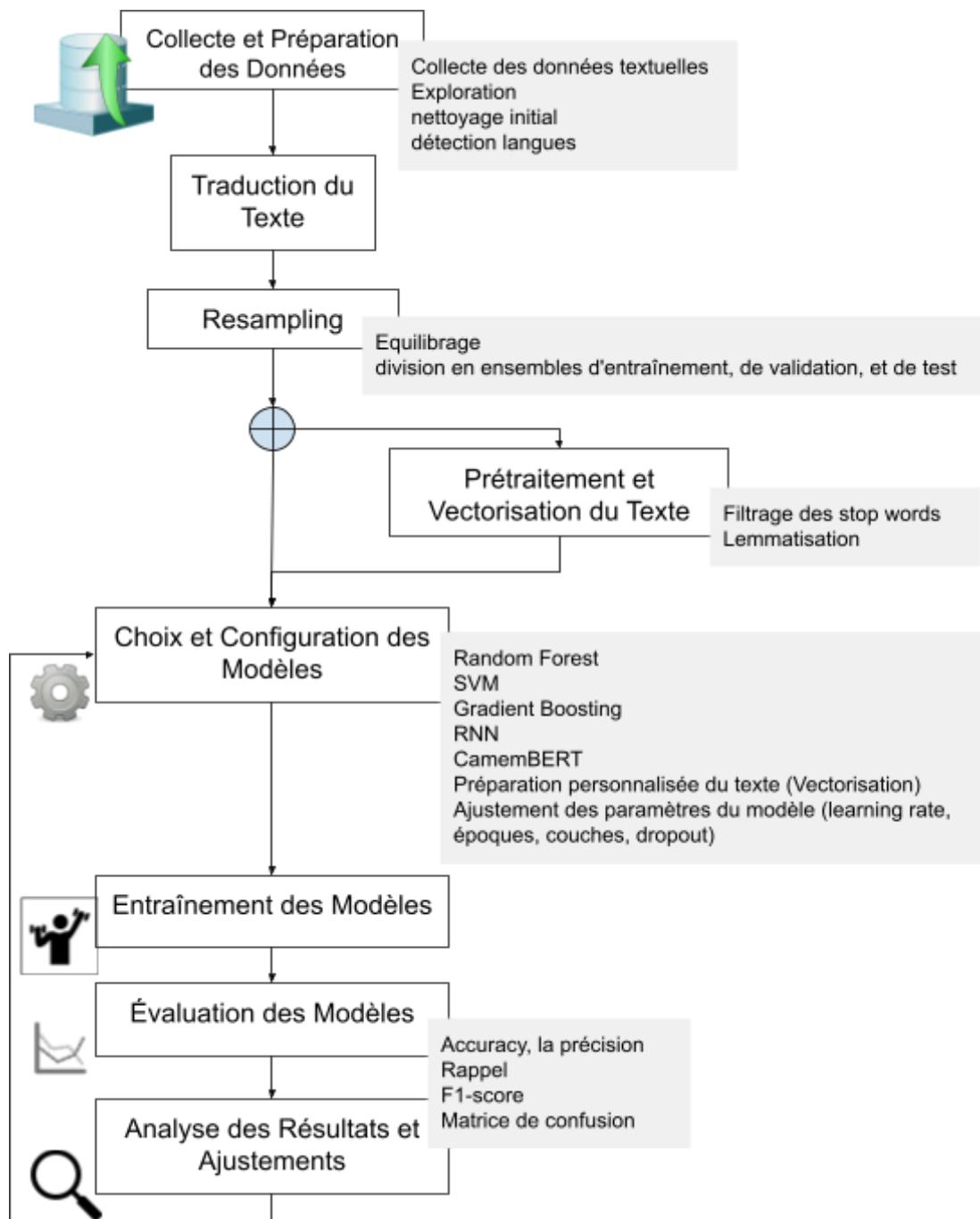
Nous avons scindé les données en trois ensembles : un ensemble d'entraînement (80% des données), un ensemble de validation (10% des données), et un ensemble de test (10% des données). Nous avons veillé à respecter la répartition initiale des classes dans chaque ensemble.

Pour rééquilibrer les classes de l'ensemble d'entraînement, nous avons réduit le nombre d'observations dans la classe majoritaire et augmenté le nombre d'occurrences des classes minoritaires jusqu'à ce qu'elles atteignent toutes environ 4000 observations, représentant la population des classes moyennement peuplées du dataset.

Enfin, nous avons exporté les trois ensembles de données dans le dossier d'output du notebook, afin qu'ils soient accessibles par les notebooks de modélisation.

# Modélisation Classification du Texte

Cette partie détaille les différentes approches de préparation de texte et de modélisation pour la classification de textes. L'objectif est d'évaluer l'efficacité de diverses techniques de traitement de texte et de modèles d'apprentissage automatique, en tenant compte de l'équilibrage des données et des paramètres spécifiques à chaque modèle.



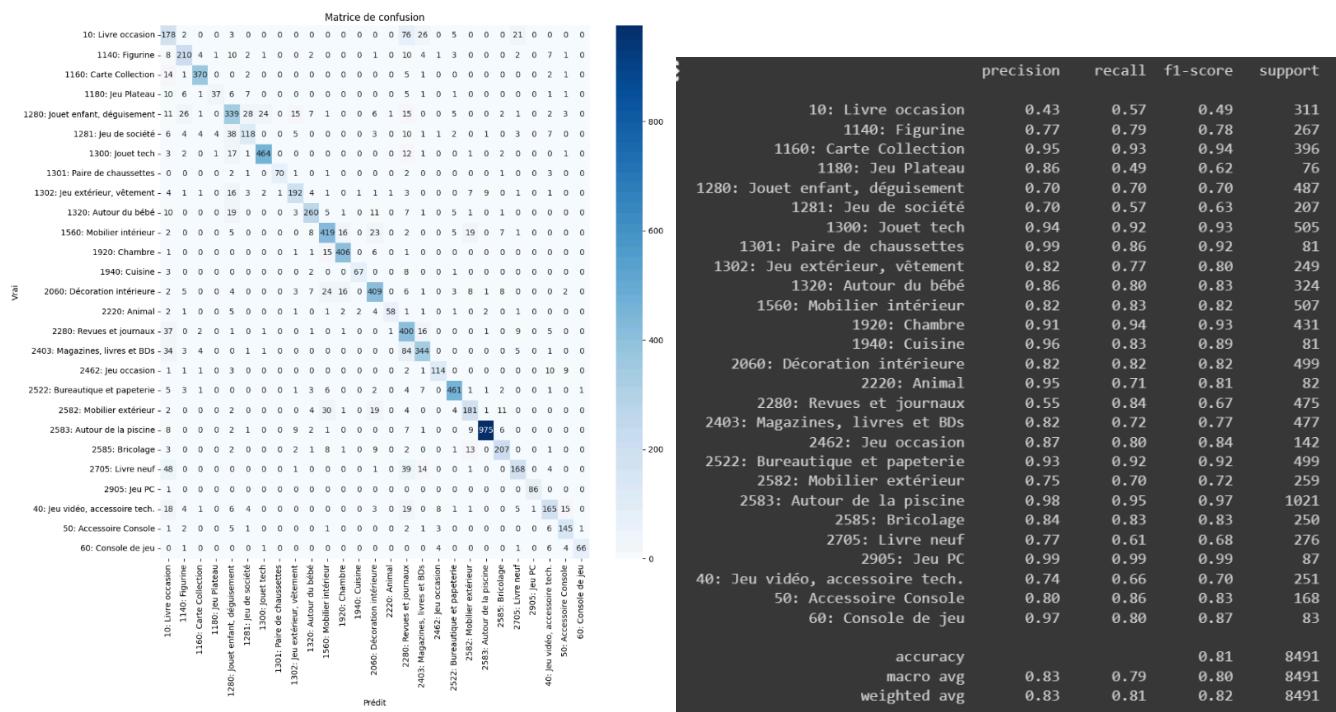
## Modèles et Performances pour le texte

#	Préparation du Texte	Paramètres	Equilibrage	Modèle	Paramètres du Modèle	Accuracy	Précision	Rappel	F1-Score
1	Texte non traduit Filtrage stop Words WordNetLemmatizer TF-IDF		Non	Random Forest	random_state=122	0,77	0,78	0,74	0,78
2	Texte traduit CBOW	vector_size = 100 window = 5	oui df_train.csv	Random Forest	random_state=123	0,68	0,69	0,68	0,68
3	Texte traduit CBOW	vector_size = 100 window = 5 min_count = 1 sg = 0	oui df_train.csv	Gradient Boosting	n_estimators = 100 learning_rate = <b>1,0</b> max_depth = 1 random_state = 0	0,3	0,31	0,3	0,3
4	Texte traduit CBOW	vector_size = 100 window = 5	oui df_train.csv	SVM		0,7	0,67	0,7	0,68
5	Skip Gram	vector_size = 100	oui	Random	random_state=123	0,74	0,75	0,71	0,74
6	Skip Gram	vector_size = 100	oui	Gradient	n_estimators = 100	0,3	0,31	0,3	0,3
7	Skip Gram	vector_size = 100	oui	SVM		0,74	0,72	0,75	0,75
8	Texte traduit tokenisé + padding	padding = 'post' maxlen = maxlen	oui df_train.csv	RNN avec GRU	optimizer='adam' loss='categorical_crossentropy' metrics=['accuracy']	0,7	0,68	0,67	0,7
9	Filtrage stop Words WordNetLemmatizer TF-IDF		oui df_train.csv	Random Forest	random_state=122	0,775	0,79	0,76	0,78
10	Filtrage stop Words WordNetLemmatizer TF-IDF		oui df_train.csv	SVM		0,81	0,83	0,81	0,82
11	Filtrage stop Words WordNetLemmatizer TF-IDF		oui df_train.csv	Gradient Boosting	n_estimators = 100 learning_rate = <b>0,1</b> max_depth = 1 random_state = 0	0,68	0,74	0,68	0,68
12	Filtrage stop Words WordNetLemmatizer tokenisé + padding	padding = 'post' maxlen = maxlen	oui df_train.csv	RNN avec GRU	optimizer='adam' loss='categorical_crossentropy' batch_size=32 epochs=10 validation_data=(X_te	0,74	0,75	0,74	0,74
13	Filtrage stop Words WordNetLemmatizer		oui df_train.csv	fasttext	lr = 0.1 # Taux d'appren dim = 100 # Taille de ws = 5 # Taille de la epoch = 25 # Nombre d' minCount = 1 # Nombre d' wordNgrams = 2 # Nombre d' loss = 'softmax' # Fonction de perte bucket = 2000000 # Taille de la fenêtre	0,7	0,72	0,7	0,7
14			oui df_train.csv	fasttext	lr = 0.1 # Taux d'appren dim = 100 # Taille de	0,54	0,64	0,54	0,54
15			oui df_train.csv	fasttext	lr = <b>0,01</b> # Taux d'appren dim = 100 # Taille de	0,47			
16			oui df_train.csv	fasttext	lr = 0.01 # Taux d'appren dim = 100 # Taille de	0,72	0,73	0,72	0,72

#	Préparation du Texte	Paramètres	Equilibrage	Modèle	Paramètres du Modèle	Accuracy	Precision	Rappel	F1-Score	Matrice de Confusion	Remarques
17	Filtrage stop Words WordNetLemmatizer	oui df_train.csv		fasttext	lr = 0.01 # Taux d'app dim = 100 # Taille de	0,78	0,78	0,78	0,78		<a href="#">17</a>
18	Filtrage stop Words WordNetLemmatizer	oui df_train.csv		fasttext	lr = 0.01 # Taux d'app dim = 100 # Taille de	0,8	0,8	0,8	0,8		<a href="#">18</a> 16min
19		oui df_train.csv		fasttext	lr = 0.01 # Taux d'app dim = <b>150</b> # Taille de	0,798	0,8	0,8	0,8		<a href="#">19</a>
20		oui df_train.csv		fasttext	lr = 0.01 # Taux d'app dim = <b>150</b> # Taille de	0,75	0,75	0,76	0,75		<a href="#">20</a>
21	Filtrage stop Words WordNetLemmatizer	oui df_train.csv		fasttext	lr = <b>0.02</b> # Taux d'app dim = 150 # Taille de ws = 5 # Taille de la epoch = 400 # Nombre minCount = 1 # Nombre wordNgrams = 2 # Util loss = 'softmax' # Fonction bucket = 2000000 #	0,797	0,8	0,8	0,8		

22	tokeniser / encoder camemBERT		oui df_train.csv df_val.csv	CamenBERT	num_train_epochs = per_device_train_batch_size = 16 per_device_eval_batch_size = 16 warmup_steps = 500 weight_decay = 0.01, logging_steps = 10, evaluation_strategy = "epoch", save_strategy = "epoch", load_best_model_at_end = True	0,88	0,88	0,88	0,88		3h V100 df_test en entrée du modèle au lieu de df_val. Les modèles tel que BERT, sont limités à des vecteurs de 512 tokens. Le texte fait parfois plus de 4000 caractères. Est ce limitant?
23	tokeniser / encoder camemBERT		oui df_train.csv df_val.csv	CamenBERT	model 22 + 1 epoch	0,88	0,89	0,88	0,88		1h V100
24	tokeniser / encoder camemBERT		oui df_train.csv df_val.csv	CamenBERT	model 22 + 3 epoch per_device_train_batch_size = 16 per_device_eval_batch_size = 16 weight_decay = 0.02	0,87	0,8	0,87	0,87		2h45 V100 de 8 à 16 :taille de lot plus grande peut réduire la variabilité des mises à jour de poids et potentiellement lisser certaines caractéristiques nuancées des données de 0,01 à 0,02 : renforce la régularisation, mais cela pourrait aussi avoir empêché le modèle d'ajuster finement ses poids pour capturer des patterns plus complexes dans les données
25	tokeniser / encoder camemBERT		oui df_train.csv df_val.csv	CamenBERT	num_train_epochs = per_device_train_batch_size = 16 per_device_eval_batch_size = 16 warmup_steps = 500 weight_decay = 0.01, logging_steps = 10, evaluation_strategy = "epoch", save_strategy = "epoch", load_best_model_at_end = True	0,86	0,87	0,87	0,86		
26	tokeniser / encoder camemBERT		oui df_train.csv df_val.csv	CamenBERT	model 25 + 3 epoch	0,885	0,89	0,89	0,89		
27	tokeniser / encoder camemBERT		oui df_train.csv df_val.csv		model 25 + 3 epoch	0,89	0,89	0,89	0,89		
28	tokeniser / encoder camemBERT		oui df_train.csv df_val.csv		model 25 + 3 epoch learning rate = 1e-5	0,8874	0,89	0,89	0,89		learning rate par default = 5e-5

- **Random Forest et SVM (Support Vector Machine)** ont été appliqués sur des données préparées avec différentes techniques. Le modèle SVM sur des données préparées avec TF-IDF et filtrage des stop words a montré la meilleure performance avec une accuracy de 0,81 et un F1-Score de 0,82.



- **Gradient Boosting** a été testé avec différentes configurations, montrant une grande sensibilité aux paramètres. Une configuration a été trouvée avec un learning rate de 0,1, atteignant une accuracy de 0,68.
- **RNN avec GRU** a été utilisé sur des textes tokenisés et paddés, atteignant une accuracy de 0,74, démontrant l'efficacité des modèles séquentiels sur les données textuelles.
- **FastText** a été exploré avec divers paramètres, montrant une amélioration significative de la performance avec des ajustements fins. Le meilleur modèle FastText a atteint une accuracy de 0,8 avec un learning rate de 0,01, une dimension de vecteur de 150, et 400 époques.
- **CamemBERT** a été exploré avec le minimum de prétraitement du texte (incluant le nettoyage balises et traduction). Le résultat d'un entraînement sur 3 époques seulement nous a permis d'atteindre une accuracy de 0,88 dès le premier essai, montrant toute la puissance des modèles transformers. Le principal problème des modèles BERT est la limitation à 512 tokens pouvant mener à une perte d'information et un contexte réduit.

## Observations

- **Équilibrage des Données :** L'équilibrage des données a joué un rôle crucial dans l'amélioration des performances des modèles, en particulier pour les classes sous-représentées.
- **Importance des Paramètres :** Les variations dans les performances en fonction des paramètres choisis (par exemple, le learning rate pour Gradient Boosting et fasttext, la taille des vecteurs de mots) illustrent l'importance d'un ajustement minutieux des hyperparamètres.
- **Efficacité des techniques de préparation du texte :** Les techniques de préparation du texte ont eu un impact notable sur la performance des modèles, avec une tendance générale montrant une meilleure performance pour les données prétraitées.
- **Efficacité des Modèles Transformers :** Les essais avec CamemBERT (essais 22 à 28) montrent des performances supérieures en termes d'accuracy, précision, rappel et F1-score comparés aux autres modèles testés, soulignant l'efficacité des modèles transformers pour la classification de texte.
- **Importance du Prétraitement :** Les résultats varient significativement en fonction du type de prétraitement appliqué, indiquant l'importance de cette étape. Par exemple, l'utilisation de WordNetLemmatizer et TF-IDF semble produire de bons résultats avec les modèles traditionnels (Random Forest, SVM, Gradient Boosting).

## Conclusion

En résumé, les essais montrent l'efficacité des modèles transformers pour la tâche de classification de texte et soulignent l'importance d'un bon prétraitement des données. L'ajustement fin des hyperparamètres et l'exploration de techniques de prétraitement supplémentaires semblent être des voies prometteuses pour améliorer les performances. La prise en compte des spécificités du texte et l'expérimentation avec différentes architectures et paramètres peuvent conduire à des améliorations significatives des résultats.

## Modélisation images avec LeNet

L'objectif du notebook était de concevoir un premier modèle de classification d'images. On a utilisé une architecture **LeNet** pour commencer. C'est un CNN simple qui nous a permis de nous faire une première idée de la contribution des différents facteurs (données, hyper-paramètres) à la performance de notre modèle.

Nos ressources de calcul (et de temps) étant limitées, on a mené les comparaisons de la section 1 (surtout) et 2 sur un sous-ensemble des données, un nombre d'époques limité et certains hyperparamètres restreints.

On a adopté la stratégie suivante:

1. On a d'abord tenté de mesurer l'impact des différentes étapes de prétraitement sur nos modèles. Toutes choses égales par ailleurs, on a testé le même modèle avec, dans l'ordre:
  1. Le dataset sans rééquilibrage de classes et avec les images non zoomées
  2. Le dataset sans rééquilibrage de classes et avec les images zoomées
  3. Le dataset avec rééquilibrage de classes et avec les images zoomées
2. Enfin, après avoir trouvé le dataset le plus approprié, on a testé l'impact de différents hyper-paramètres sur la performance:
  1. Un learning rate plus élevé avec une stratégie de *learning rate decay*
  2. Un batch\_size plus gros
  3. Une taille d'image plus grande
3. On a sélectionné le dataset et les hyper-paramètres les plus prometteurs des sections précédentes et entraîné le modèle de façon plus poussée sur un dataset complet, en restaurant les hyper-paramètres qu'on avait restreints par souci d'économie de temps lors de nos comparaisons.

*Note: Nous avons pris un soin particulier à développer une librairie de fonction qui nous permettrait d'être plus résilient aux interruptions qui ont lieu régulièrement lors de la phase d'entraînement de modèle sur google colab. Ce système nous a permis de ne re-exécuter que le strict nécessaire lors de la relance d'un notebook: Les modèles et leur historique d'entraînement étaient rechargés, leurs entraînements complétés sur le nombre d'époques manquantes si nécessaire. Finalement, les résultats de leur évaluation sur l'ensemble de test étaient affichés.*

*Nous avons également développé un ensemble de fonctions qui nous ont permis d'enregistrer un rapport contenant les informations pertinentes sur nos différentes stratégies de modélisation ainsi que leurs résultats au fur et à mesure de nos tests.*

## 1. Impact du preprocessing

On peut voir dans le tableau qui suit, un comparatif des résultats des trois tests menés dans le cadre de la stratégie 1.

Les IDs de modèles suivent une convention à trois chiffres:

1. Le numéro du notebook (ici 1)
2. Le premier niveau de stratégie (ici 1)
3. Le deuxième niveau de stratégie (ici 1, 2 ou 3)

La signification des en tête de ligne est la suivante:

- **model\_name:** Le nom qu'on a donné au modèle
- **dataset\_resampled:** Si on utilise ou non l'ensemble de donnée avec rééquilibrage de classes
- **image\_dataset\_zoomed:** Si on utilise les images zoomées ou non
- **train\_size:** La taille du jeu de données d'entraînement (en nombre d'observations) lorsqu'on opère sur une portion du jeu de données seulement (ce qui est le cas ici).
- **image\_size:** A quelle taille les images sont redimensionnées avant d'être utilisées pour l'entraînement du modèle.
- **batch\_size:** Quelle taille de batch est utilisée.
- **test\_f1\_score:** Le f1 score obtenu sur l'ensemble de test après l'entraînement.
- **test\_accuracy:** L'accuracy obtenue sur l'ensemble de test après l'entraînement.
- **loss:** La valeur finale de la fonction de perte sur l'ensemble d'entraînement.
- **val\_loss:** La valeur finale de la fonction de perte sur l'ensemble de validation.

<b>model_id</b>	<b>111</b>	<b>112</b>	<b>113</b>
<b>model_name</b>	Modèle 1.1.1	Modèle 1.1.2	Modèle 1.1.3
<b>dataset_resampled</b>	False	False	True
<b>image_dataset_zoomed</b>	False	True	True
<b>train_size</b>	21229.0	21229.0	21229.0
<b>image_size</b>	128.0	128.0	128.0
<b>batch_size</b>	64.0	64.0	64.0
<b>test_f1_score</b>	0.23761	0.242837	0.207138
<b>test_accuracy</b>	0.290391	0.295337	0.208314
<b>loss</b>	2.482575	2.500585	2.707499
<b>val_loss</b>	2.452756	2.452618	2.748649

Nos tests nous ont laissé penser que l'utilisation d'images zoomées semblerait judicieux.

Le choix du dataset ré-équilibré a été plus difficile. D'un côté, le f1-score global penchait en faveur du non mais une observation plus détaillée des rapports de classification et matrices de confusion nous a indiqué qu'il semblait plus prometteur. En effet, les modèles 111 et 112 avaient des prédictions manquantes pour certaines classes ce qui aurait compromis la fiabilité de nos futurs tests.

Nous avons donc décidé d'utiliser un dataset avec images zoomées + rééquilibrage pour la suite.

## 2. Impact des hyper-paramètres

Nous avons obtenu les résultats suivants de notre seconde stratégie.

<b>model_id</b>	<b>121</b>	<b>122</b>	<b>123</b>
<b>model_name</b>	Modèle 1.2.1	Modèle 1.2.2	Modèle 1.2.3
<b>dataset_resampled</b>	True	True	True
<b>image_dataset_zoomed</b>	True	True	True
<b>train_size</b>	21229.0	21229.0	21229.0
<b>image_size</b>	128.0	128.0	256.0
<b>batch_size</b>	64.0	128.0	64.0
<b>test_f1_score</b>	0.221112	0.20597	0.240261
<b>test_accuracy</b>	0.23846	0.211258	0.240108
<b>loss</b>	2.718754	2.756888	2.771065
<b>val_loss</b>	2.672804	2.736719	2.760436

Avec le modèle **121**, suite à l'implémentation d'un *decay learning rate*, nous avons constaté une amélioration de la vitesse d'apprentissage (la courbe de loss s'aplatissait moins) et des performances (+0.02 sur le f1-score).

L'augmentation de **batch\_size** testée avec le modèle **122** a eu pour conséquence un léger aplatissement de la loss sans impact sur le f1-score de validation ainsi qu'une diminution inattendue du nombre de classes ayant des prédictions manquantes

L'augmentation de la taille des images nous a permis de gagner 0.04 de f1-score ainsi et de ne plus avoir de classes ayant des prédictions manquantes.

Suite à cette analyse, les hyper-paramètres qui nous ont semblé les plus prometteur étaient:

- Utiliser un *decay learning rate*
- Utiliser un **batch\_size** de 128 au moins
- Utiliser au moins une **image\_size** de 256

### 3. Entraînement complet d'un modèle basé sur les enseignements tirés des sections 1 et 2

On a sélectionné le dataset et les hyper-paramètres les plus prometteurs des sections précédentes et entraîné le modèle de façon plus poussée sur un dataset complet, en restaurant les hyper-paramètres qu'on avait restreints par souci d'économie de temps lors de nos comparaisons.

On a d'abord lancé un premier test sur 10 époque avec 30% des données, qui nous a permis de paramétriser notre *learning rate decay*.

On a ensuite lancé le test final sur 20 époques et avec la totalité des données.

Nous avons obtenu un f1-score de 0.31 sur les données de test.

## Conclusion

model_id	111	112	113	121	122	123	131	132
model_name	Modèle 1.1.1	Modèle 1.1.2	Modèle 1.1.3	Modèle 1.2.1	Modèle 1.2.2	Modèle 1.2.3	Modèle 1.3.1	Modèle 1.3.2
dataset_resampled	False	False	True	True	True	True	True	True
image_dataset_zoomed	False	True						
train_size	21229.0	21229.0	21229.0	21229.0	21229.0	21229.0	32869.0	109566.0
image_size	128.0	128.0	128.0	128.0	128.0	256.0	256.0	256.0
batch_size	64.0	64.0	64.0	64.0	128.0	64.0	128.0	128.0
test_f1_score	0.23761	0.242837	0.207138	0.221112	0.20597	0.240261	0.270583	0.307977
test_accuracy	0.290391	0.295337	0.208314	0.23846	0.211258	0.240108	0.280146	0.306877
loss	2.482575	2.500585	2.707499	2.718754	2.756888	2.771065	2.603446	2.3765
val_loss	2.452756	2.452618	2.748649	2.672804	2.736719	2.760436	2.512697	2.376215

Tableau récapitulatif des modèles testés dans le notebook

Ce notebook nous a permis de tester l'impact de différents preprocessing (ré-équilibrage des classes, zoom des images) sur la performance d'un modèle CNN classique comme **LeNet**. On a aussi pu tester et mieux comprendre l'influence de différents hyper-paramètres comme le learning rate, la taille des batch et la taille des images.

Les performances obtenues avec le dernier modèle étaient encore loin des 0.5-0.6 de f1-score qu'on aurait aimé atteindre. Il est probable qu'entrainer encore le modèle aurait amélioré les performances même si on ignore jusqu'à quel point. Notre temps

et nos ressources de calcul étant limité, il était difficile de pousser plus loin l'expérience avec ce modèle **LeNet**.

## Modélisation images avec VGG16

Ce notebook a fait suite au notebook 1 qui nous a permis de sélectionner les datasets les plus appropriés à notre tâche de classification et aussi de mieux comprendre l'impact des divers hyper-paramètres sur une architecture classique comme LeNet.

N'ayant pu atteindre un niveau de performance adéquat dans un temps raisonnable, nous avons cette fois utilisé sur un modèle de transfert learning en utilisant une architecture **VGG16** avec les poids d'**Imagenet**, un dataset d'images génériques.

Nous avons procédé en cinq étapes:

1. Effectuer un entraînement de test sur une portion limitée du dataset pour évaluer le choix de nos hyper-paramètres initiaux.
2. En se basant sur l'évaluation du premier test, lancer un entraînement sur les données complètes
3. Ré-entraîner les 4 dernières couches de **VGG16** pour tenter d'ajuster au plus près les poids de la partie extraction de features du meilleur modèle de 1 et 2
4. Sélectionner le modèle le plus prometteur puis refaire des entraînements en tentant de résoudre les problèmes d'overfitting observés
5. Ré-entraîner les 4 dernières couches de **VGG16** pour tenter d'ajuster au plus près les poids de la partie extraction de features du meilleur modèle de 4

Nous avons obtenu les meilleurs résultats avec le modèle de l'étape 3 (id **331**) qui a atteint un f1-score de 0.60, comme le montre le tableau récapitulatif suivant:

model_id	311	321	331	341	342	343	351	352
model_name	Modèle 3.1	Modèle 3.2	Modèle 3.3	Modèle 3.4.1	Modèle 3.4.2	Modèle 3.4.3	Modèle 3.5.1	Modèle 3.5.2
dataset_resampled	True	True	True	True	True	True	True	True
image_dataset_zoomed	True	True	True	True	True	True	True	True
train_size	21913.0	109566.0	109566.0	109566.0	109566.0	109566.0	109566.0	109566.0
image_size	224.0	224.0	224.0	224.0	224.0	224.0	224.0	224.0
batch_size	64.0	64.0	64.0	32.0	32.0	32.0	32.0	32.0
accuracy	0.558103	0.669275	0.742927	0.421559	0.486169	0.525681	0.664643	0.551354
f1_score	0.552243	0.665658	0.739655	0.415287	0.481566	0.521459	0.660953	0.545998
val_accuracy	0.516927	0.552675	0.597656	0.472641	0.50625	0.523821	0.573113	0.541274
val_f1_score	0.52043	0.561671	0.601403	0.483479	0.51619	0.529464	0.582362	0.548519
test_f1_score	0.514401	0.556558	0.6015	0.492511	0.512809	0.530819	0.583075	0.553074
test_accuracy	0.515191	0.552991	0.598681	0.480217	0.506123	0.528144	0.57772	0.549223
loss	1.433475	1.067026	0.806426	1.990221	1.739799	1.601717	1.09201	1.505246
val_loss	1.707712	1.61175	1.47987	1.791615	1.648645	1.634477	1.492979	1.522122

## Le meilleur modèle

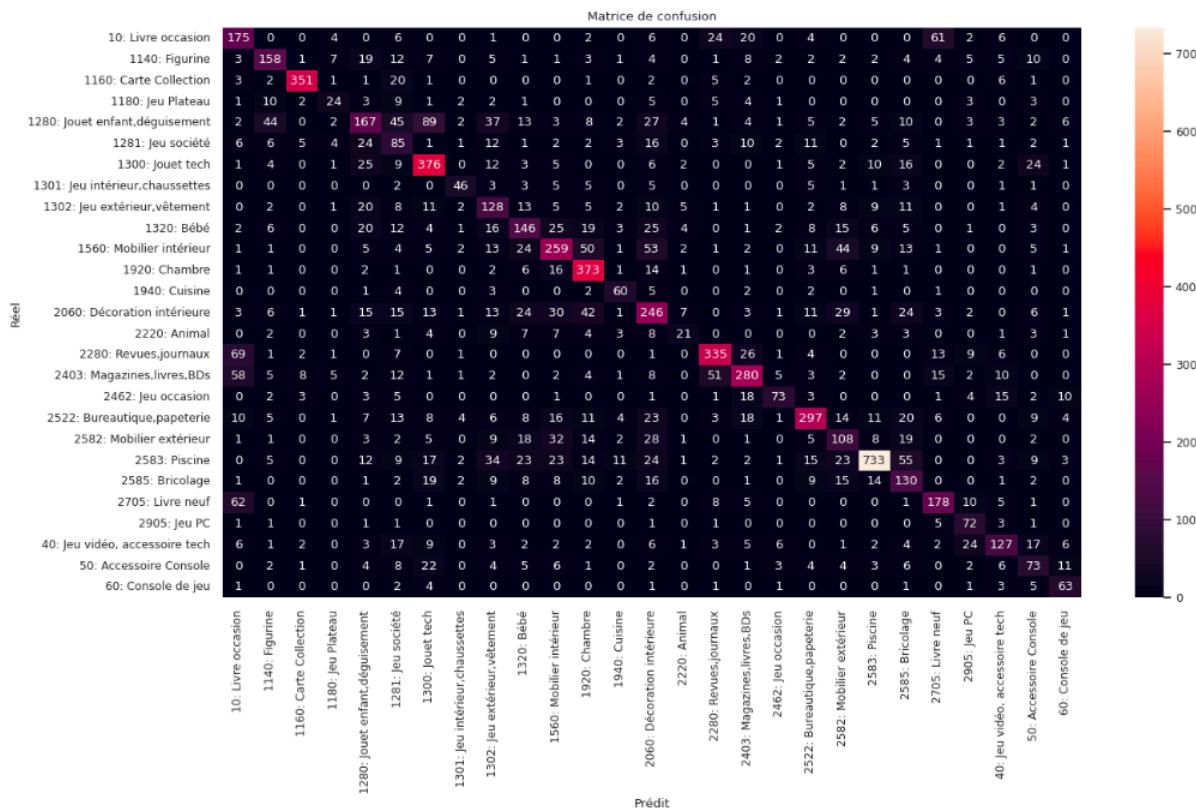
Le graphique suivants montrent l'évolution des métriques de notre modèle **331** durant l'entraînement:

- L'évolution de l'accuracy et du f1-score en fonction des époques.
- L'évolution des indicateurs de perte loss et val\_loss.



Voici également reproduit ici son rapport de classification et sa matrice de confusion:

	precision	recall	f1-score	support
10	0.43	0.56	0.49	311
1140	0.60	0.59	0.59	267
1160	0.93	0.89	0.91	396
1180	0.46	0.32	0.37	76
1280	0.49	0.34	0.40	487
1281	0.27	0.41	0.33	207
1300	0.63	0.74	0.68	505
1301	0.69	0.57	0.62	81
1302	0.40	0.51	0.45	249
1320	0.48	0.45	0.46	324
1560	0.58	0.51	0.54	507
1920	0.65	0.87	0.74	431
1940	0.61	0.74	0.67	81
2060	0.45	0.49	0.47	499
2220	0.43	0.26	0.32	82
2280	0.75	0.70	0.73	476
2403	0.67	0.59	0.63	477
2462	0.72	0.51	0.60	142
2522	0.73	0.60	0.65	499
2582	0.39	0.42	0.40	259
2583	0.89	0.72	0.80	1021
2585	0.39	0.52	0.45	250
2705	0.61	0.64	0.63	276
2905	0.51	0.83	0.63	87
40	0.62	0.51	0.56	251
50	0.39	0.43	0.41	168
60	0.58	0.76	0.66	83
accuracy			0.60	8492
macro avg	0.57	0.57	0.56	8492
weighted avg	0.62	0.60	0.60	8492



Nous avons tenté de rester synthétiques dans ce rapport mais le lecteur curieux pourra consulter tous les détails dans le notebook intitulé **data-modeling-images-3**.

## Conclusion

Ce notebook nous a permis d'explorer les avantages du transfert learning qui permet d'avoir rapidement un extracteur de features performant pour des images, accélérant ainsi drastiquement l'apprentissage d'un classifieur.

Il est possible qu'on puisse avoir des performances correctes en investissant le temps et les ressources de calcul nécessaires à l'entraînement d'un CNN depuis le début. Cela représente toutefois un temps et un investissement qu'il paraît difficile de justifier dans un contexte d'entreprise, à moins que celle-ci ne soit vraiment tournée vers la recherche.

Quant à la question du meilleur modèle qu'on a retenu, après des discussions avec notre mentor qui nous ont permis d'éclaircir la notion d'overfitting, nous avons résolu de retenir le modèle 331, c'est-à-dire celui qui atteint le f1-score maximum du notebook. En effet, la crainte d'un l'overfitting le concernant n'était finalement pas justifiée. Malgré une séparation des courbes de validation et d'entraînement, le

modèle continuait de progresser en capacité de généralisation, même si la vitesse de progression restait inférieure à celle de la progression de l'apprentissage sur l'ensemble d'entraînement.

## Fusion des modèles de texte et d'image

L'objectif du notebook **data-modeling-fusion** était de tester des modèles d'ensemble qui permettent de fusionner les prédictions des meilleurs modèles que nous avions pu trouver au cours de nos modélisations sur les textes et les images.

Nous avons expérimenté deux grands types de modèles d'ensemble en trois étapes:

1. Un modèle classique de fusion qui reprend les principes de la classe **sklearn.ensemble.VotingClassifier** à laquelle on aurait passé le paramètre **voting="soft"**.
2. Une émulation de **sklearn.ensemble.StackingClassifier** utilisant **LogisticRegression** comme classifieur final.
3. Nous avons réutilisé le modèle de fusion de 1 avec un nouveau modèle texte basé sur **CamemBERT**

### 1. VotingClassifier

Comme annoncé dans l'introduction, pour ce premier test de fusion de nos modèles texte et image, nous avons utilisé une approche probabiliste répliquant l'algorithme employé par la classe **sklearn.ensemble.VotingClassifier** à laquelle on aurait passé le paramètre **voting = "soft"**.

**VotingClassifier** n'étant pas directement compatible avec les modèles keras (bien qu'il existe des systèmes d'adaptateurs). Il nous a paru plus simple d'effectuer nous-même la moyenne des probabilités des prédictions en sortie des deux modèles.

Nous avons sélectionné les modèles de texte et d'image suivants :

- Le modèle d'image **331** ayant obtenu un f1-score de 0.60 (voir le notebook **data-modeling-images-3**)
- Le modèle de texte **1.10svm** avec un f1-score de 0.81 (voir le notebook **data-modeling-text-1bis**)

Après avoir effectué quelques préparations d'usage: Création d' **ImageDataGenerator** pour les images et transformation du texte en matrice TFIDF pour le texte, nous avons chargé nos deux modèles et appelé leurs méthodes **predict** et **predict\_proba** pour obtenir les probabilités de prédictions sur l'ensemble de données de test.

Ces prédictions ont ensuite été passées à notre fonction maison chargée de calculer la moyenne des probabilités de prédictions en affectant un poids identique de 0.5 à chacun des modèles.

On a obtenu un f1-score de 0.8252, légèrement supérieur à celui obtenu avec le modèle texte, qui était de 0.81. Cela nous a semblé un peu en dessous de nos attentes, mais pour un modèle basé sur la moyenne des probabilités de prédictions, il était finalement logique que les résultats s'alignent sur le meilleur des modèles lorsque l'autre modèle a un f1-score légèrement supérieur à 0.50 (0.60).

Nous avons ensuite réitéré le même test mais en modifiant les poids assignés aux différents modèles lors du calcul de la moyenne:

- Pour le modèle de texte, nous avons mis un poids de 0.6
- Pour le modèle d'image, nous en avons mis un de 0.4

Ces poids ont été déterminés en testant toutes les combinaisons possibles variant par tranche de 10%: 0.9/0.1, 0.8/0.2, ..., 0.1/0.9.

C'est la combinaison 0.6/0.4 qui a montré les meilleures performances, le f1-score étant des 0.8328, soit légèrement supérieur à celui de 0.8252 obtenu lors du test précédent.

## 2. StackingClassifier

Dans la seconde section du notebook, nous avons testé un modèle d'ensemble différent. Reprenant la logique de la classe **StackingClassifier** de sklearn, nous avons entraîné un modèle **LogisticRegression** à partir des probabilités de prédictions de nos modèles texte et image.

Plus précisément, nous avons suivi ces étapes pour l'entraînement:

1. Obtenir les prédictions de notre modèle d'image sur l'ensemble d'entraînement.

2. Obtenir les prédictions de notre modèle de texte sur l'ensemble d'entraînement.
3. Concaténer ces prédictions.
4. Entraîner **LogisticRegression** à partir des prédictions concaténées en 3

Nous avons ensuite suivi des étapes similaires pour obtenir les prédictions de **LogisticRegression** sur les données de test en remplaçant l'ensemble d'entraînement par celui de test et en appelant **LogisticRegression.predict** au lieu de **LogisticRegression.fit** à l'étape 4.

Nous avons obtenu un f1-score de 0.8300 avec ce système, soit une performance intermédiaire se situant entre celle des tests 1.1 et 1.2.

### 3. VotingClassifier avec modèle texte basé sur CamemBERT

Ce nouveau test a réitéré le test de la section 1 mais en utilisant cette fois un modèle texte différent:

- Le modèle de texte modele\_camembert\_028 avec un f1-score de 0.887 (voir le notebook data-modeling-text-6 retrain-CamenBERT)

Nous avons obtenu le meilleur f1-score du notebook avec 0.8907.

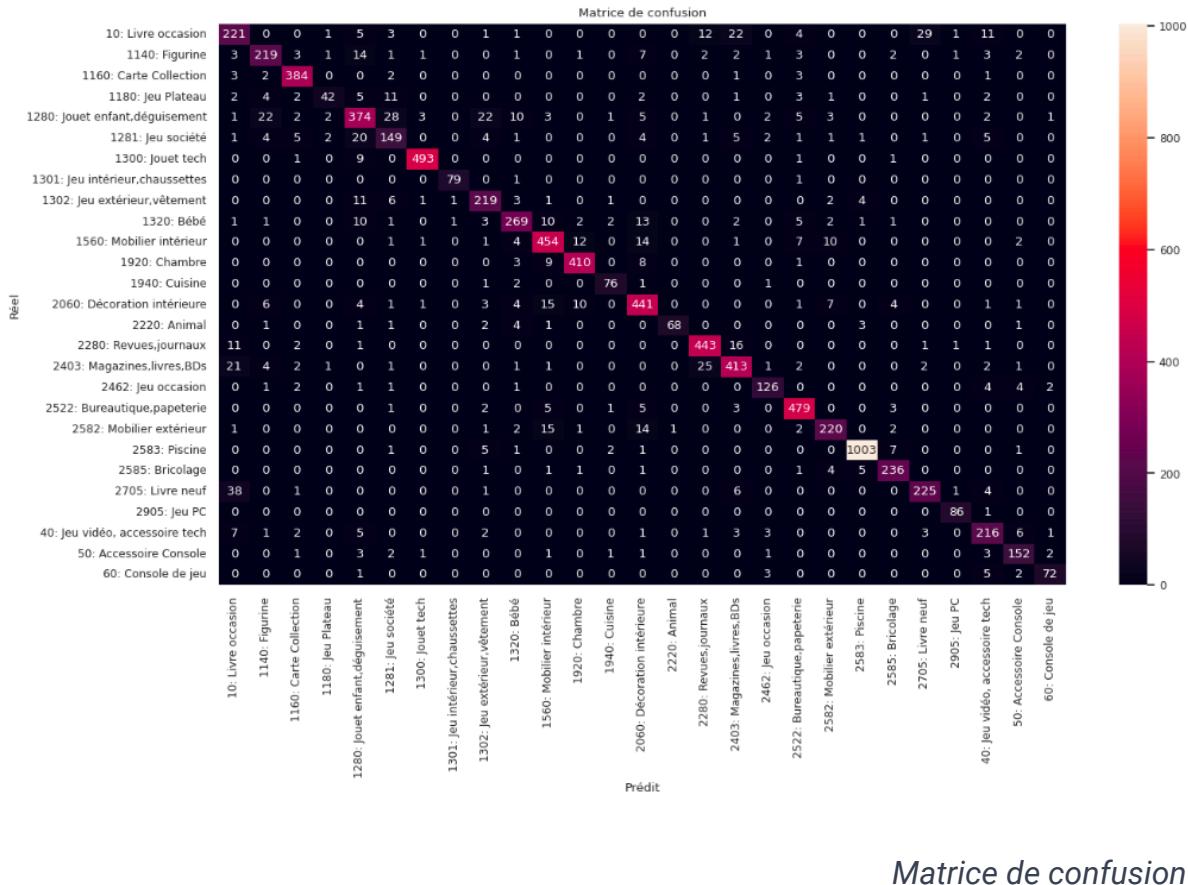
Signalons qu'on a testé toutes les combinaisons de poids et que c'est celle-ci qui a obtenu les meilleures performances.

On constate que le modèle image ne contribue que marginalement à la performance puisque le f1-score du modèle texte seul est, rappelons le 0.887.

	precision	recall	f1-score	support
10	0.71	0.71	0.71	311
1140	0.83	0.82	0.82	267
1160	0.94	0.97	0.96	396
1180	0.86	0.55	0.67	76
1280	0.81	0.77	0.79	487
1281	0.71	0.72	0.71	207
1300	0.98	0.98	0.98	505
1301	0.98	0.98	0.98	81
1302	0.82	0.88	0.85	249
1320	0.87	0.83	0.85	324
1560	0.88	0.90	0.89	507
1920	0.94	0.95	0.94	431
1940	0.90	0.94	0.92	81
2060	0.85	0.88	0.87	499
2220	0.99	0.83	0.90	82
2280	0.91	0.93	0.92	476
2403	0.87	0.87	0.87	477
2462	0.90	0.89	0.89	142
2522	0.92	0.96	0.94	499
2582	0.88	0.85	0.86	259
2583	0.99	0.98	0.98	1021
2585	0.92	0.94	0.93	250
2705	0.86	0.82	0.84	276
2905	0.96	0.99	0.97	87
40	0.83	0.86	0.84	251
50	0.88	0.90	0.89	168
60	0.92	0.87	0.89	83
accuracy			0.89	8492
macro avg	0.89	0.87	0.88	8492
weighted avg	0.89	0.89	0.89	8492

*Rapport de classification*

Dans le rapport de classification, on obtient logiquement les f1-score les plus élevés qu'on ait eu jusqu' alors. Plusieurs classes atteignent 0.99. Le f1-score le plus bas est de 0.67.



Sur la matrice de confusion, les vrais positifs n'ont jamais été aussi élevés. On remarque encore quelques erreurs de prédictions mais elles sont à la fois moins nombreuses et moins élevées individuellement.

## Conclusion

model_id	411	412	421	431
model_name	Modèle 4.1.1	Modèle 4.1.2	Modèle 4.2.1	Modèle 4.3.1
fusion_model	VotingClassifier(voting='soft') emulation	VotingClassifier(voting='soft') emulation	StackingClassifier(final_emulator=LogisticRegr...	VotingClassifier(voting='soft') emulation
image_model_weight	0.5	0.4	0.5	0.5
text_model_weight	0.5	0.6	0.5	0.5
image_model_id	331.0	331.0	331.0	331
text_model_id	1.10svm	1.10svm	1.10svm	modele_camembert_028
dataset_resampled	True	True	True	True
image_dataset_zoomed	True	True	True	True
test_size	8491.0	8491.0	8491.0	8492.0
test_f1_score	0.825214	0.832794	0.830024	0.890718
test_accuracy	0.827229	0.834766	0.830055	0.891309

Tableau récapitulatif des modèles d'ensemble testés

Ce tableau recense les différents modèles testés au cours du notebook. Il reprend dans l'ordre de haut en bas les informations suivantes: id du modèle, type de modèle de fusion utilisé, poids accordés aux modèles individuels, id des modèles individuels, ensemble de données utilisés, taille du jeu de test, f1-score et accuracy obtenus par l'évaluation du modèle sur le jeu de test.

Nous avons finalement obtenu des performances proches de celles que nous visions (0.9 de F1-score). Le dernier modèle de texte **CamemBERT** a vraiment permis un bond en avant par rapport aux performances obtenues avec des modèles de texte plus classiques tels que ceux utilisant **TF-IDF** en conjonction avec **SVM**.

Il est probable qu'un peu de temps supplémentaire nous aurait permis d'atteindre et peut-être de dépasser le seuil de 0.9 de F1-score. Toutefois, à ce stade, l'échéance du projet étant imminente, nous devrons nous contenter de ces performances qui demeurent tout à fait correctes pour un premier projet de classification utilisant le deep learning.

Nous aurions également souhaité pouvoir expérimenter des CNN plus modernes. Nous sommes conscients d'être restés plutôt classiques avec **VGG** et **LeNet**. Nous avons privilégié une approche assez conservatrice car au stade où nous avions fait ces choix d'architectures, avec notre niveau de connaissance du moment, ils nous ont semblé plus faciles d'accès. Cela nous a toutefois permis d'approfondir notre connaissance des "classiques" de la computer vision (de faire nos gammes, en quelque sorte) et d'obtenir des bases qui seront sûrement utiles pour la suite de nos parcours en data science.

# Conclusion générale

## Difficultés rencontrés lors du projet

On a d'abord pensé pouvoir faire l'économie de certaines étapes de preprocessing comme le zoom des images ou le rééquilibrage des classes pour vite se ravisier devant le besoin de comprendre d'où venait les problèmes de performance des premiers modèles. Il nous a fallu éliminer les facteurs de mauvaise performance un à un pour tenter d'en comprendre l'origine, les données étant les premières visées. Cela nous a permis de comprendre l'importance de bien respecter l'ordre des étapes d'un projet de machine learning.

Dès les premières modélisations à base de CNN, nous avons vite réalisé à quel point les avertissements concernant la nécessité d'un hardware suffisamment puissant pour entraîner des modèles de deep learning étaient fondés. Il nous est alors apparu que la formule gratuite de google colab ne serait pas suffisante pour mener à bien ce projet, tant l'accès à un GPU sans limites de disponibilité était vital.

Malgré le passage à une formule payante sur google colab qui nous a donné accès à un hardware plus puissant, le temps d'entraînement en deep learning restait lent. On a rapidement compris que pour mener les tests qu'on voulait, on devrait opérer sur une portion des données seulement en espérant que les enseignements qu'on en tirerait seraient généralisables à l'ensemble des données.

La multiplication des tests dans les notebooks nous également confronté à la nécessité de rendre les notebooks résilients aux interruptions de travail volontaires ou non afin qu'ils puissent reprendre le traitement là où il s'était arrêté sans devoir tout réexécuter.

Enfin, nous avons compris l'importance d'avoir un système de log pour s'y retrouver dans la multiplicité des nos tests de modélisation.

## Bilan

L'utilisation de modèles pré-entraînés et le transfert learning peuvent réduire considérablement le besoin en calcul en exploitant des caractéristiques apprises sur de grands ensembles de données, réduisant ainsi le temps nécessaire pour l'entraînement sur un nouveau jeu de données.

## Suite du projet

Pour surmonter les limitations des modèles transformers traditionnels, l'exploration de modèles tels que BigBird, conçus pour traiter efficacement de longs documents, représente une voie prometteuse. Ces modèles offrent une alternative pour gérer des textes excédant largement la limite de 512 tokens, potentiellement améliorant la compréhension contextuelle et la performance globale sur des tâches de classification de textes complexes.

Côté image, nous aurions aimé avoir le temps d'expérimenter des architectures CNN différentes telles que EfficientNet, qui nous aurait sans doute permis d'augmenter la taille des images fournies au réseau de neurones, ou encore Inception pour sa capacité à faire le focus sur les portions significatives des images.

## Bibliographie

[ILLUSTRATION DU TRANSFORMER - Loïck BOURDOIS \(lbourdois.github.io\)](#)

[Word2vec : NLP & Word Embedding - DataScientest](#)

[Comprendre le fonctionnement d'un LSTM et d'un GRU en schémas - Pensée Artificielle \(penseeartificielle.fr\)](#)

[Hands-on CamemBERT: Une Introduction au Modèle CamemBERT | CamemBERT \(camembert-model.fr\)](#)

[Blog - CamemBERT - final.ipynb - Colaboratory \(google.com\)](#)

[INTRODUCTION TO DATA FUSION \(medium.com\)](#)

[Ensemble: Scikit-learn and Keras \(medium.com\)](#)

[What is Hard and Soft Voting in Machine Learning? \(medium.com\)](#)