

Projet Rakuten

Julien METHIVIER
Geoffray CENGIZALP
Karim TACHE

Sommaire

Sommaire.....	2
Introduction au projet.....	4
Contexte.....	4
Objectifs.....	5
Analyse exploratoire et preprocessing.....	5
Analyse exploratoire pour les données textuelles.....	5
Preprocessing pour les données textuelles.....	5
Analyse exploratoire du dataset d'images.....	14
Analyse générale du dataset.....	14
Analyse de la luminosité et du contraste.....	15
Analyse des doublons.....	16
Preprocessing des images.....	18
Centrage et standardisation des images.....	18
Traitement des doublons et valeurs manquantes.....	18
Rééquilibrage des classes.....	19
Classification du problème.....	20
Choix du modèle et optimisation.....	21
Choix du modèle pour l'analyse d'image.....	21
Benchmark des modèles.....	21
Analyse post-fine-tuning.....	22
Analyse de performance du modèle.....	23
Interprétabilité avec Grad-CAM.....	23
Conclusion.....	26
Choix du modèle pour l'analyse du texte.....	28
Modèles simples - KNN.....	28
Modèles simples - Decision Tree.....	33
Modèles simples - Naive Bayes.....	38
Modèles simples - Regression Logistique.....	43
Modèles simples - Ridge Classifier.....	46
Modèles simples - Linear SVM.....	51
Modèles complexe pré-entraîné - BERT.....	57
Modèles complexe pré-entraîné - PYTORCH.....	60
Combinaison des modèles.....	62
Bilan du projet.....	62
Annexes.....	64
Description des fichiers de code.....	64

Introduction au projet

Contexte

Le sujet de ce projet est une problématique à laquelle la plupart des marketplaces sont confrontées. La classification automatique des produits est extrêmement importante pour réduire les coûts, améliorer la qualité des données catalogue, et optimiser l'expérience utilisateur. Elle permet à une marketplace de proposer des fonctionnalités allant de la recommandation personnalisée à la compréhension de requêtes de recherche complexes.

Du point de vue économique, La classification automatique des produits est extrêmement importante pour réduire les coûts, améliorer la qualité des données catalogue, et optimiser l'expérience utilisateur. Dans un contexte où les marketplaces gèrent des millions de références, une mauvaise catégorisation peut entraîner :

- une perte de visibilité produit (donc une baisse des ventes),
- des erreurs de recommandation,
- ou encore une diminution du taux de conversion en raison de recherches mal filtrées.

En automatisant cette tâche via des algorithmes d'intelligence artificielle, les entreprises peuvent :

- accélérer le temps de mise en ligne des produits,
- réduire les coûts liés à la classification manuelle,
- standardiser l'organisation du catalogue à grande échelle,
- et mieux personnaliser l'offre en fonction des comportements d'achat.

Du point de vue technique, ce projet mobilise plusieurs compétences apprises tout au long de la formation : le traitement de données textuelles et image, l'apprentissage supervisé, l'évaluation de modèles, et l'optimisation des performances algorithmiques.

Il implique la préparation de données textuelles et visuel brutes,

- Côté **texte**, cela a nécessité la mise en place des **pré traitements linguistiques** (nettoyage, tokenisation, filtre des stop words), ainsi que l'utilisation de méthodes de vectorisation (**TF-IDF**) et de resampling (**SMOTE**).
- Côté **image**, cela a également demandé des nombreux traitements pour améliorer la qualité du dataset, ainsi que l'utilisation de méthodes d'augmentation afin d'optimiser la capacité du modèle à bien généraliser

Objectifs

L'objectif général est de réussir à classer automatiquement des produits dans le catalogue produit de Rakuten France en exploitant efficacement les données textuelles et visuelles associées aux produits dans le but d'améliorer la qualité des données, optimiser l'expérience utilisateur et réduire les coûts opérationnels.

Analyse exploratoire et preprocessing

Analyse exploratoire pour les données textuelles

Dans le cadre du projet, il est d'abord apparu que les variables dénomination, description et prdtypecode étaient les plus pertinentes. La variable cible in fine étant prdtypecode.

Le jeu de données présente plusieurs particularités qui ont nécessité une attention particulière en phase de préparation. Tout d'abord, les colonnes "dénomination" et "description" étaient souvent redondantes, avec parfois un simple copier-coller entre les deux. De plus, certaines descriptions étaient rédigées en anglais ou dans d'autres langues, ce qui complexifie le traitement sémantique. Enfin, les textes contenaient de nombreux artefacts tels que des balises HTML, des caractères spéciaux, des accents mal encodés ou d'autres bruits textuels nuisant à l'analyse.

Preprocessing pour les données textuelles

Les données ont nécessité un nettoyage approfondi. Nous constatons tout d'abord que les informations nécessaires à la prédiction de la catégorie sont réparties dans deux colonnes distinctes : "**dénomination**" et "**description**". Afin de simplifier la complexité du prétraitement (tokenisation, vectorisation,...), nous avons pensé à **fusionner ces deux colonnes** pour créer une nouvelle colonne "**Merged**". Cependant avant de passer à cette étape de fusion, nous avons procédé à différents nettoyage de données décrit ci-après.

Tout d'abord, nous avons pris soin de supprimer les doublons en colonnes (parfois le champ dénomination et description contenant les mêmes informations). Nous avons également supprimé les doublons de lignes. Ces suppression de doublons, aussi bien en colonnes qu'en lignes, ont été appliquées à plusieurs reprises, après chaque étape de nettoyage de données.

caractere	
& amp	936
 	451
& lt	100
& 2	93
& gt	82
& blanc	36
& dragons	34
& play	24
& 3	22
& the	21
& easy	17
& ntsc	17
& mavic	17
& bois	16
& modeles	16
& decker	16
& d	14
& jardin	14
& smart	12
& clear	11
& stratton	11
& triphase	11
& plug	11
& zoom	10
& revetement	10
& warner	10
& 1	9
& metal	9
& noir	9
& newton	9

Name: count, dtype: int64

Nous avons commencé par **normaliser le texte** en retirant les **caractères spéciaux** (retrait des accents, cédilles,...), en mettant l'ensemble du texte en **minuscule**, et en **supprimant les espaces inutiles** (double espace, espace en début ou fin de texte, etc.).

Nous avons ensuite retiré l'ensemble des **URLs** présentes dans le texte, ainsi que les **balises HTML** mal retranscrites, probablement dû à une faute de frappe (espace entre le '&' et le caractères, omission du '&'). Nous avons fait attention à ne pas supprimer tous les caractères composées de la chaîne ["&" + "espace" + texte] en sortant un tableau des occurrences comme illustré à gauche.

En affichant les occurrences 30 les plus fréquentes, nous avons identifié les caractères spéciaux suivants :

- "**&**" qui correspond simplement à "&"
- "**&nbs**" qui correspond à un espace qui ne peut pas être cassé par un retour à la ligne,
- "**<**" qui correspond à <,
- "**>**" qui correspond à >.

Pour finir, nous avons également essayé un **correcteur orthographique**. Malheureusement, en plus d'être extrêmement chronophage (ceux à quoi nous pouvions nous attendre étant donné la volumétrie des données), le correcteur s'est avéré très peu efficace en corrigeant énormément de mots qui étaient corrects. Exemples :

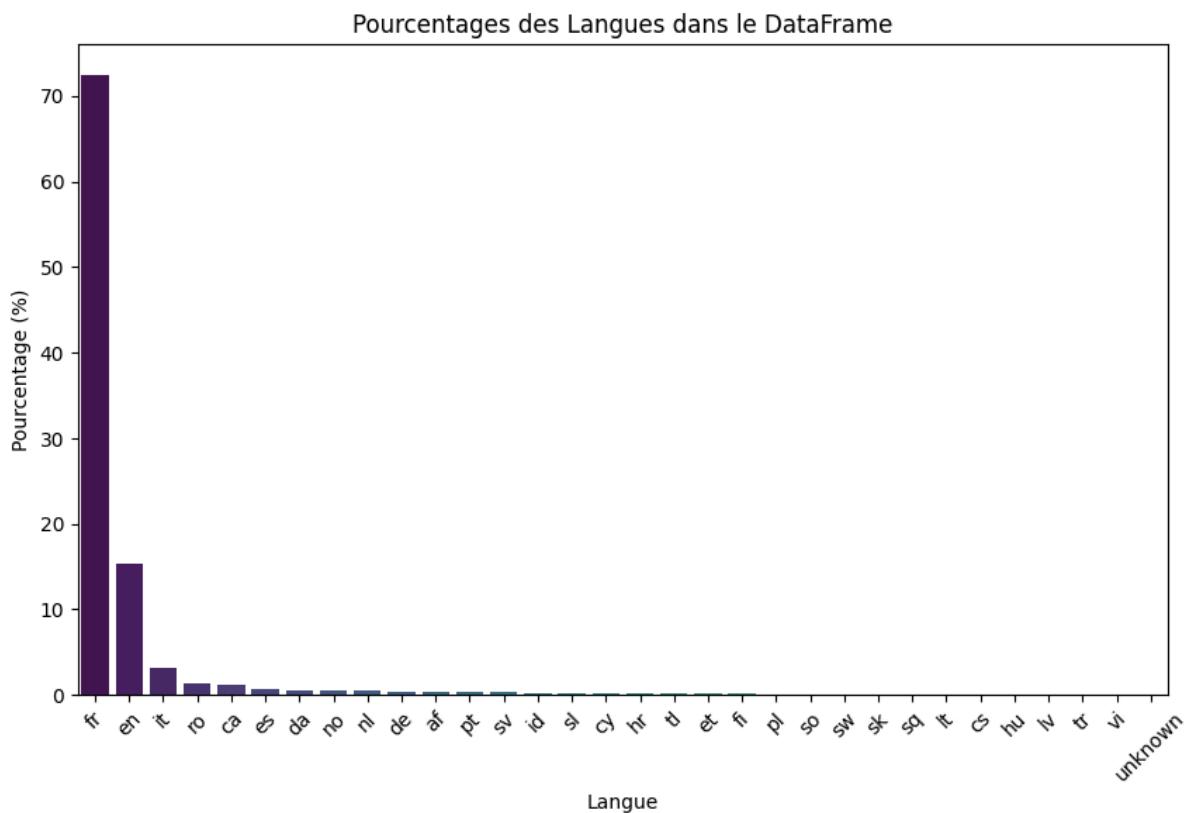
- **donald** (prénom) → **tonal**,
- **cagettes** (vrai mot absent du dictionnaire du correcteur) → **canettes**,
- **ph** (unité scientifique) → **pu**,
- **magnet** (anglicisme) → **magnat**,
- **zumba** (nom propre) → **tomba**
- **tv** (diminutif de télévision) → **tu**
- **vii** (chiffre romain) → **vie**

En fin de compte, nous avons choisi de ne pas utiliser le correcteur orthographique qui semblait apporter plus d'erreurs qu'il en corrigeait.

Pour finir, nous avons **traduit en français** les contenus non francophones de cette colonne en utilisant l'**API DEEPL**. Avant de traduire l'ensemble du texte en français, nous avons commencé par analyser les différentes langues présentes.

La version gratuite de l'API DEEPL n'offrant qu'un quota limité de caractères, nous n'avons pas utilisé la détection de langue proposée par l'API DEEPL. Nous avons utilisé le module "**langdetect**" afin d'économiser notre quota DEEPL.

En analysant les différentes langues contenue dans le dataframe on constate une variété de langages assez importante :



Ce n'est qu'après avoir observé cette répartition de langue (72% français, 15% anglais, 13% autres) que nous avons choisi de traduire l'ensemble de nos données en français. Cette solution nous a paru la plus pertinente car elle permettait de limiter les éventuelles erreurs de traduction.

Pourcentages des langues :

language	%
fr	72.397324
en	15.366631
it	3.195599

ro	1.285241
ca	1.260236
es	0.687629
da	0.590111
no	0.590111
nl	0.561355
de	0.447584
af	0.438832
pt	0.367569
sv	0.336313
id	0.286304
sl	0.281303
cy	0.255048
hr	0.246296
tl	0.227543
et	0.227543
fi	0.191286
pl	0.111271
so	0.103769
sw	0.093768
sk	0.090017
sq	0.088767
lt	0.082515
cs	0.063762
hu	0.042508
lv	0.040008
tr	0.038757
vi	0.003751
unknown	0.001250

DataScientest.com

Agrément organisme de formation 11755665975

09 80 80 79 49

2 place de Barcelone, 75016 Paris

L'utilisation de cette API s'est avérée très efficace mais quelque peu contraignante en raison de la limitation du nombre de caractères pouvant être traduit "gratuitement". Nous avons donc dû faire attention au nombre de caractères à traduire. Cela nous a amené à retravailler notre **pré-nettoyage** et à **tronquer** les descriptions trop longues, souvent répétitives, pour conserver une information concise et pertinente.

A nous trois, nous avions un quota total de **1 500 000 caractères** pour la traduction. Le dataframe contenait **3 876 000 caractères** en langues autres que le français. Nous avons donc tronqué chaque ligne afin de pouvoir rentrer dans le quota de Deepl.

Il a donc fallu travailler et coder en plusieurs étapes pour le texte. Nous avons tout d'abord découpé le dataframe en trois afin de chacun traduire une partie (avec environ 26000 lignes à analyser traduire). Puis le fichier a été réuni à nouveau une fois la colonne merged traduite.

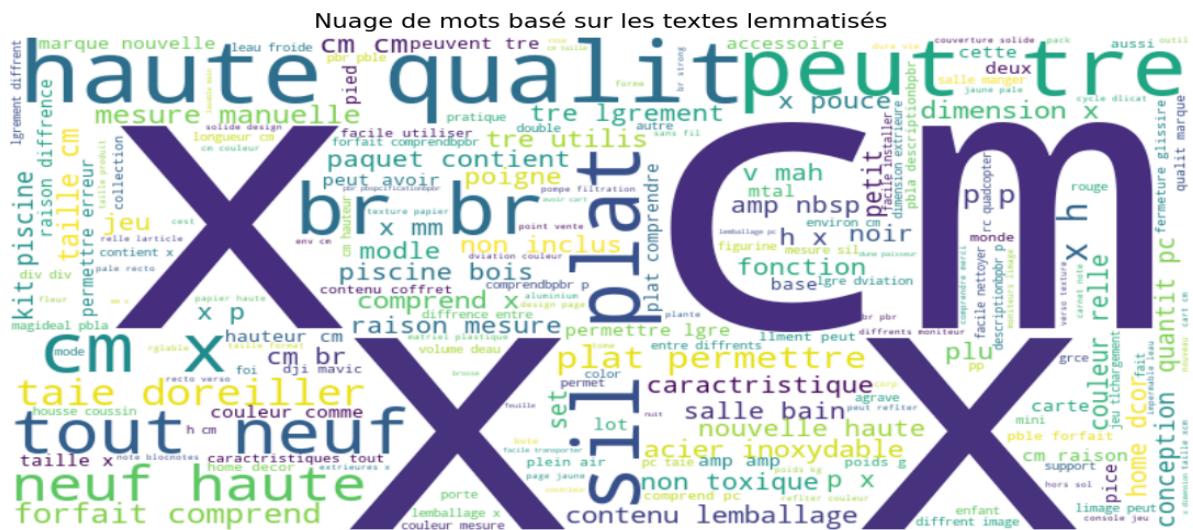
Une fois la traduction faite, nous sommes passés par l'étape de **tokenisation**, suppression des **stopwords**, et **lemmatisation** de nos données afin de réduire la complexité des textes et d'avoir un modèle plus robuste. Nous avons ensuite fait des word clouds afin de visualiser les mots les plus importants au global et pour chaque classe..



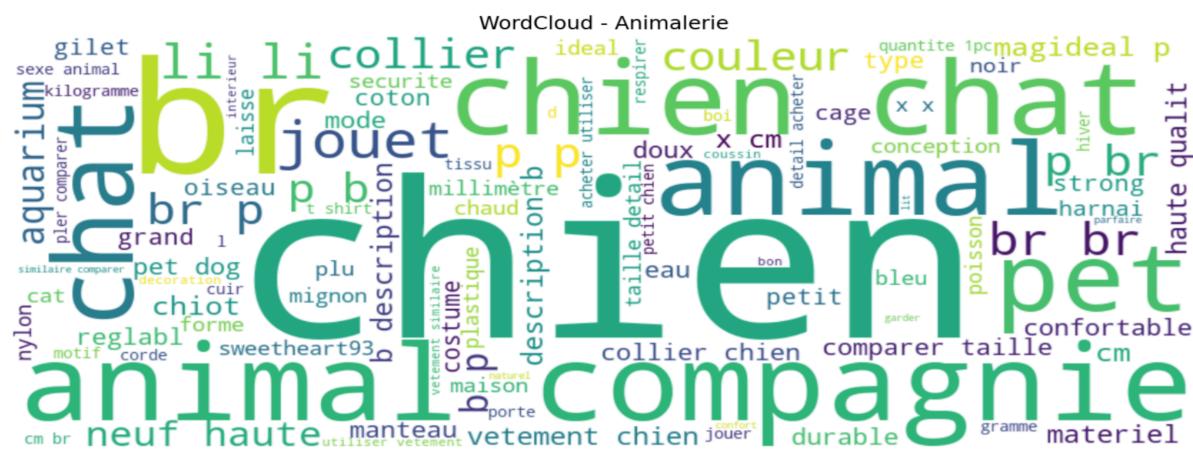
Ce premier wordcloud montre qu'il y a encore un peu de travail pour rendre le texte plus lisible/analyisable. On peut voir aussi en analysant la lemmatisation qu'il y a encore quelques erreurs de texte restant :

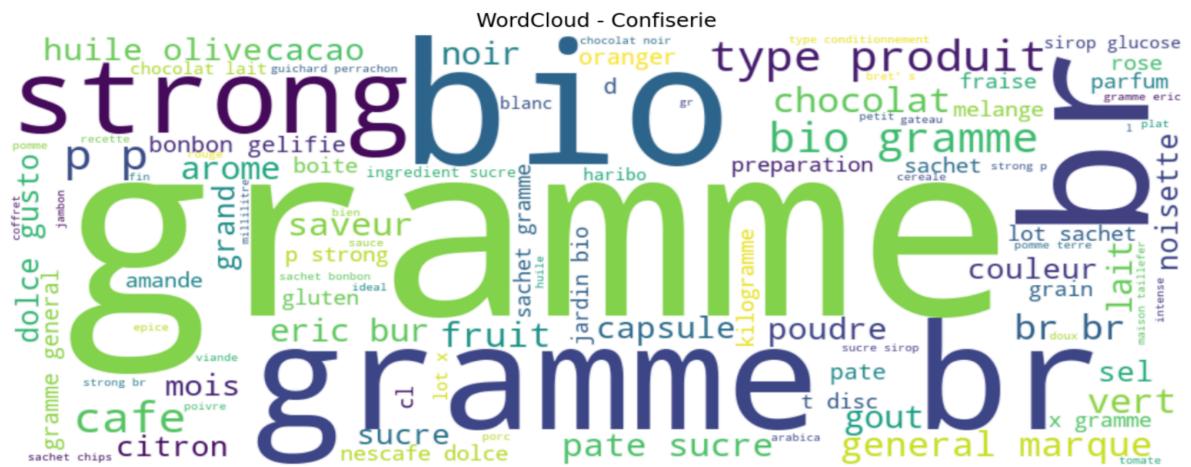
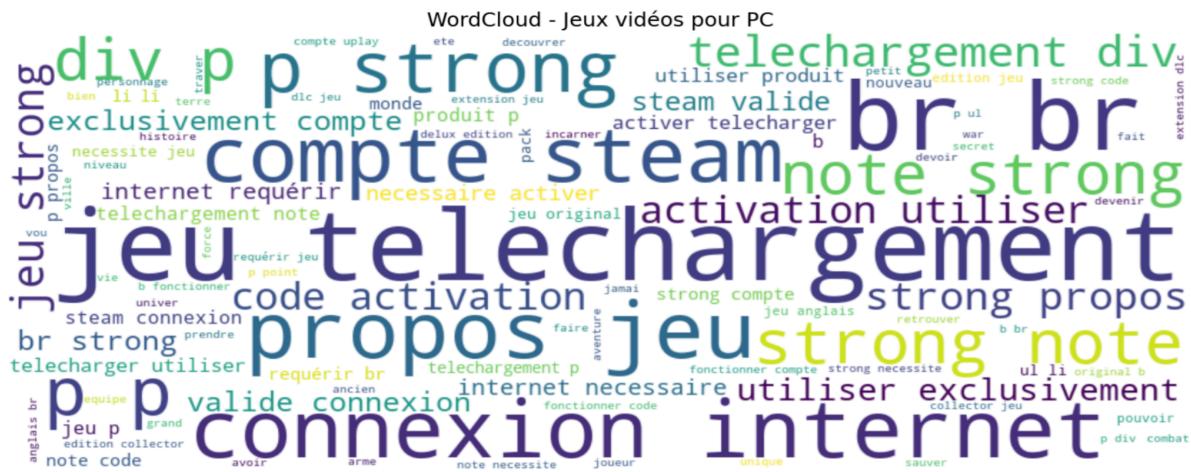
- Nombre de phrases vectorisées : 52775
 - Nombre de tokens uniques (features) : 150257
 - Extraits de tokens : ['aa' 'aaa' 'aaaa' 'aabatteries' 'aabattery' 'aabr' 'aaabrbrcaracteristiques' 'aaafeatures' 'aaafunction' 'aaah']

Après une nouvelle passe de nettoyage, on obtient le wordcloud suivant:

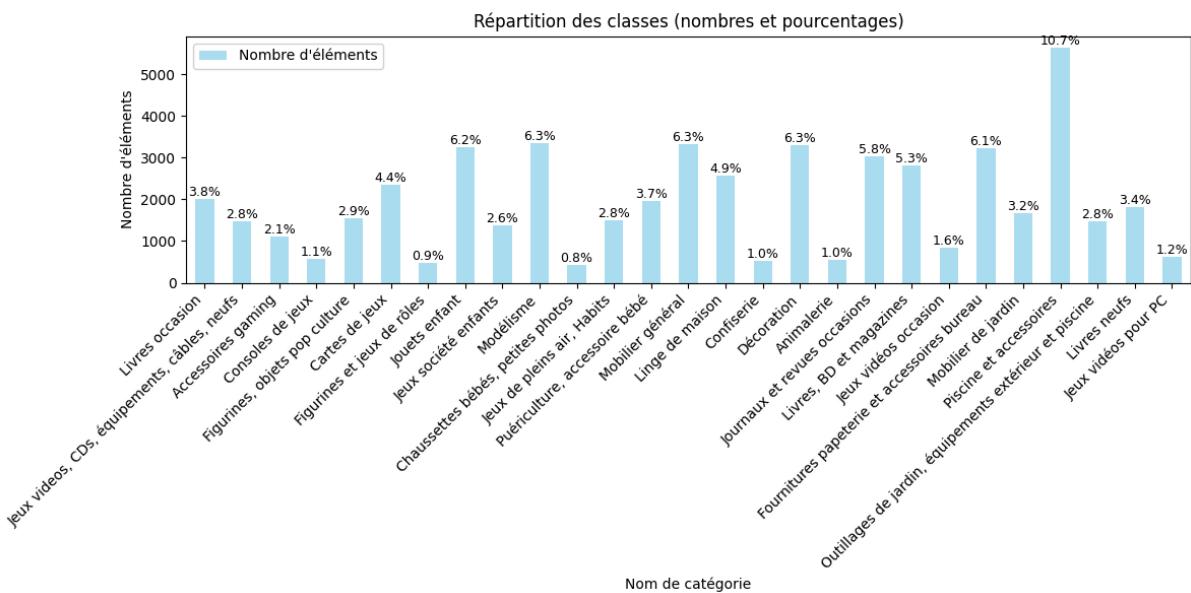


Nous avons également généré les word clouds par classe. Ces derniers nous ont paru assez satisfaisant et représentaient bien les différentes classes :

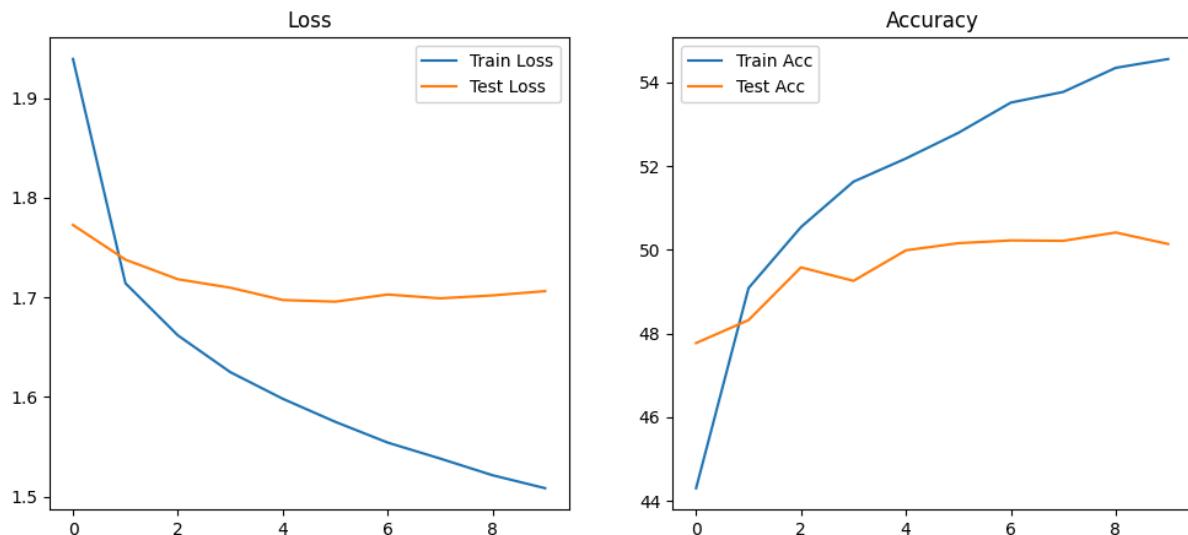




Enfin, nous avons **analysé la distribution des classes** (types de produits) afin de détecter d'éventuels déséquilibres impactant l'apprentissage supervisé. Voici la répartition des classes au sein du dataframe :



Nous pouvons constater une grande diversité dans la répartition des classes, ce qui rend l'analyse compliquée. Il s'agit donc désormais d'équilibrer les classes afin de pouvoir faire une analyse cohérente et pertinente. En effet, on peut constater que le résultat d'un modèle de **classification complexe via Pytorch** (pour une première approche) est assez catastrophique avant équilibrage:



... Rapport de classification (avec noms de catégories):

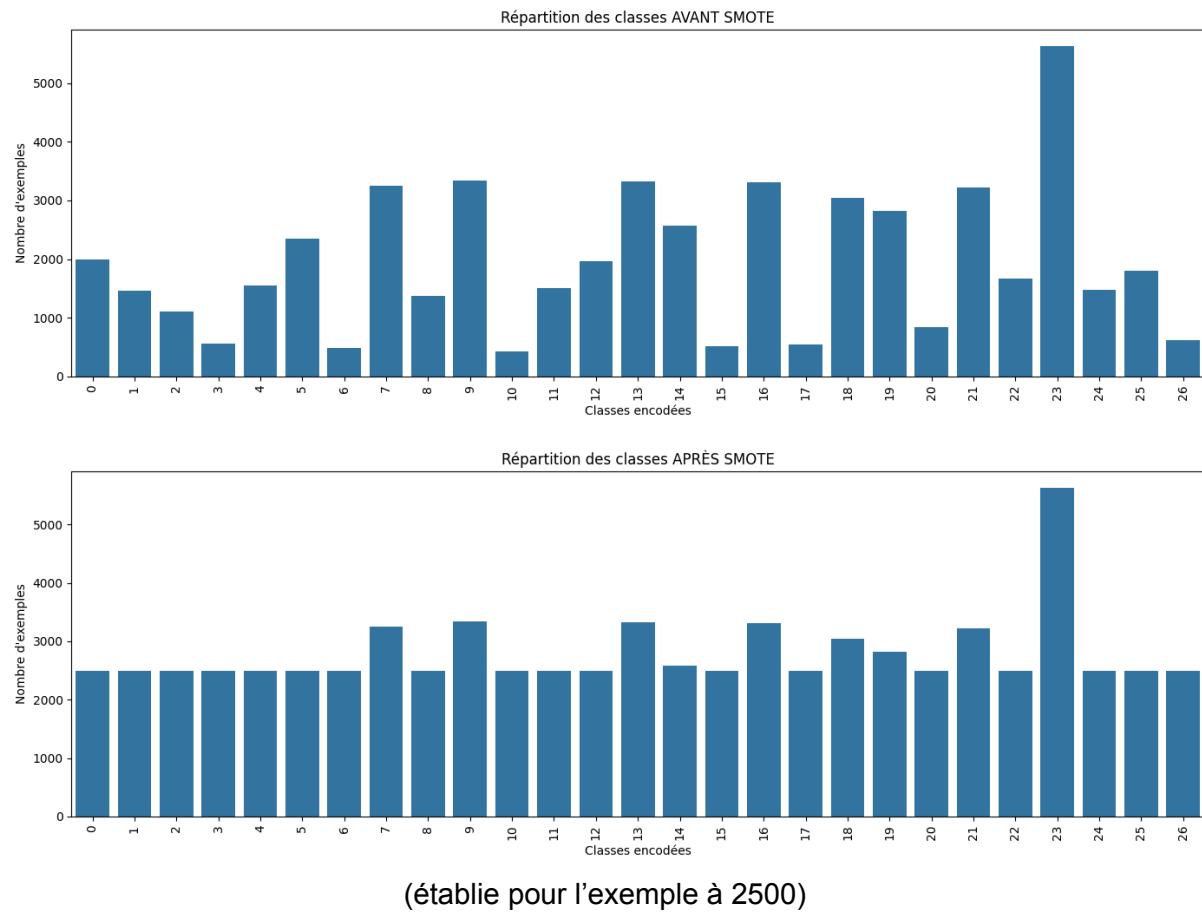
	precision	recall	f1-score	support
Accessoires gaming	0.55	0.24	0.33	225
Animalerie	0.43	0.29	0.34	104
Cartes de jeux	0.78	0.31	0.44	487
Chaussettes bébés, petites photos	0.86	0.47	0.61	79
Confiserie	0.46	0.30	0.36	122
Consoles de jeux	0.85	0.71	0.77	103
Décoration	0.62	0.63	0.62	681
Figurines et jeux de rôles	0.15	0.02	0.03	108
Figurines, objets pop culture	0.36	0.23	0.28	309
Fournitures papeterie et accessoires bureau	0.55	0.59	0.57	636
Jeux de pleins air, Habits	0.46	0.34	0.39	299
Jeux société enfants	0.39	0.32	0.35	282
Jeux vidéos, CDs, équipements, câbles, neufs	0.61	0.23	0.34	295
Jeux vidéos occasion	0.65	0.15	0.24	165
Jeux vidéos pour PC	0.90	0.85	0.88	117
Jouets enfant	0.48	0.33	0.39	669
Journaux et revues occasions	0.19	0.86	0.31	601
Linge de maison	0.88	0.72	0.79	528
Livres neufs	0.55	0.50	0.52	377
Livres occasion	0.30	0.17	0.22	397
Livres, BD et magazines	0.71	0.40	0.52	545
Mobilier de jardin	0.32	0.38	0.35	338
Mobilier général	0.52	0.57	0.54	641
Modélisme	0.77	0.68	0.72	670
Outils de jardin, équipements extérieur et piscine	0.40	0.29	0.34	285

Accessoires de jardin, équipements extérieur et piscine	0.45	0.25	0.31	285
Piscine et accessoires	0.84	0.78	0.81	1101
Puériculture, accessoire bébé	0.56	0.44	0.49	391
accuracy			0.50	10555
macro avg	0.56	0.44	0.47	10555
weighted avg	0.58	0.50	0.51	10555

Afin de pouvoir procéder à un équilibrage des classes, nous avons d'abord dû **vectoriser** nos données en appliquant une **transformation TF-IDF** (Term Frequency–Inverse Document Frequency) sur les données textuelles. Cette méthode de vectorisation, bien qu'elle ne soit pas une "normalisation" au sens strict mathématique (comme la standardisation par moyenne et écart-type), a pour objectif de **pondérer l'importance des mots** dans les descriptions, en réduisant l'influence des termes trop fréquents ou peu discriminants. Cela permet de transformer le texte en vecteurs exploitables par des algorithmes de machine learning tout en **réduisant le bruit et la redondance**.

Il a fallu également envisager une technique de réduction de dimension, car TF-IDF génère des vecteurs très **hauts en dimension**, souvent **sparsifiés**, ce qui peut augmenter le **risque de surapprentissage** et ralentir l'entraînement des modèles.

Par ailleurs, **SMOTE** (Synthetic Minority Over-sampling Technique) a été utilisé pour **équilibrer les classes** dans un contexte de déséquilibre important. Bien que SMOTE ne soit pas une réduction de dimension en soi, il modifie la structure de l'espace des données en générant des exemples synthétiques dans les zones sous-représentées, ce qui améliore la capacité de généralisation du modèle sur les classes minoritaires.



Analyse exploratoire du dataset d'images

Analyse générale du dataset

Pour commencer, nous avons extrait certaines caractéristiques des images pour les comparer plus facilement et avoir une vision d'ensemble de la qualité du dataset.

Pour cela nous avons ciblé les caractéristiques suivantes:

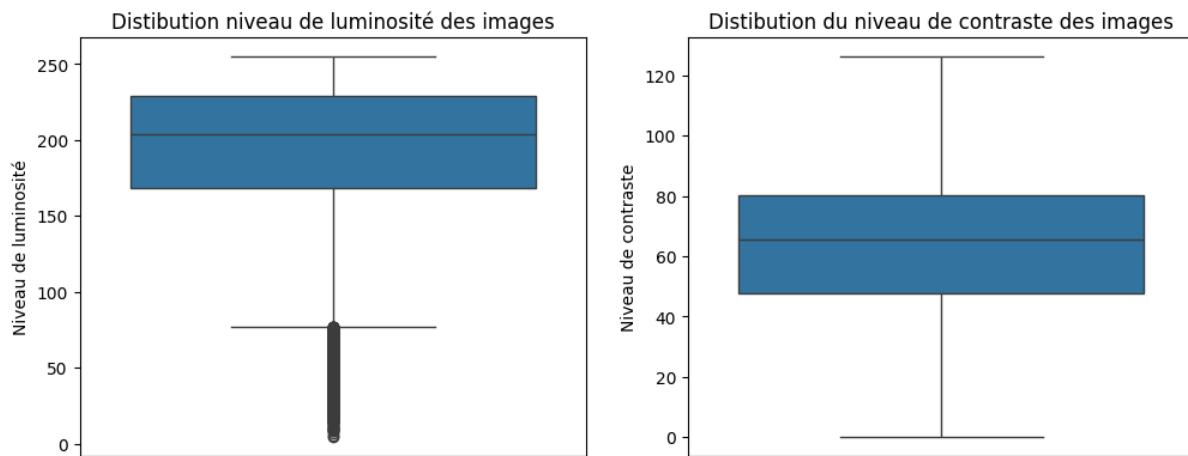
- **Dimensions des images (hauteur / largeur / ratio)**
- **Hash perceptuel de l'image** : utile pour détecter des **doublons** ou des **variantes** d'une même image
- **Luminosité moyenne et niveau de contraste**: intéressant pour évaluer la qualité des images et détecter des problèmes de surexposition ou un manque de contraste qui rendraient les images difficiles à exploiter

Stats	height	width	ratios	file_sizes	mean_luminosity
count	84916.0	84916.0	84916.0	84916.00	84916.000000
mean	500.0	500.0	1.0	26581.79	195.774120
std	0.0	0.0	0.0	13845.76	41.127735
min	500.0	500.0	1.0	2061.000000	4.177464
25%	500.0	500.0	1.0	15712.750000	168.361869
50%	500.0	500.0	1.0	24531.500000	203.464844
75%	500.0	500.0	1.0	35613.000000	229.240259
max	500.0	500.0	1.0	106720.000000	255.000000

Voici les conclusions que nous pouvons tirer de l'analyse des caractéristiques

- Le dataset contient uniquement des images carrées de dimension 500x500 pixels. On peut supposer qu'elles sont formatées avant usage et qu'il en sera de même pour les futures images à analyser
- Les images sont toutes au format JPEG et utilisent le mode RGB
- En terme de taille de fichier le dataset semble homogènes, les fichiers font typiquement en 15Ko et 35Ko et n'excèdent pas les 110Ko
- Sur la base d'une comparaison des hash perceptuels **7590 doublons** ont été identifiés soit environ **9% du dataset**. Nous allons étudier plus en détail ce point car il pourrait s'avérer problématique lors de la phase de modélisation

Analyse de la luminosité et du contraste



Distribution du niveau de luminosité

Les boxplots ci-dessus nous permettent de mettre en évidence certaines informations concernant le niveau de luminosité :

- En majorité les images ont une luminosité comprise entre 170 et 230
- La médiane se situe autour de 200
- Cette distribution indique que les images sont généralement très lumineuses
- Il y a des valeurs aberrantes en dessous de 50, indiquant quelques images très sombres dans le dataset

La fait que la majorité des images soient lumineuses, voir très lumineuses peut s'expliquer par le fait que les images contiennent une grande proportion de blanc. Il semble que le traitement qui normalise la taille des images, rajoute un fond blanc pour arriver à un format carré de 500 pixels de côté. Si l'image d'origine est petite cela aboutit à une image majoritairement blanche ce qui pourrait être problématique pour l'analyse par la suite. Par ailleurs, si on analyse les valeurs aberrantes (ie en dessous de 50) on constate qu'il s'agit souvent d'images de produits présentés sur fond noir ou mis en scène dans un contexte nocturne.

Distribution du niveau de contraste

En ce qui concerne, le niveau de contraste on remarque qu'il se situe principalement entre 45 et 80 avec une médiane autour de 65.

Si on considère l'échelle de valeurs suivante pour le contraste

- inférieur 20 = contraste très faible
- entre 20 et 40 = contraste faible
- entre 40 et 80 = contraste moyen
- entre 80 et 100 = contraste élevé
- supérieur 100 = contraste très élevé

On peut en déduire que

- La majorité des images se situe dans la zone de contraste "moyen"
- Peu d'images ont un contraste faible (< 40) ou très élevé (> 100)
- La distribution est relativement équilibrée autour de la médiane

On peut en conclure que le niveau de contraste semble approprié pour la plupart des images, suggérant une bonne qualité générale du dataset

Conclusion

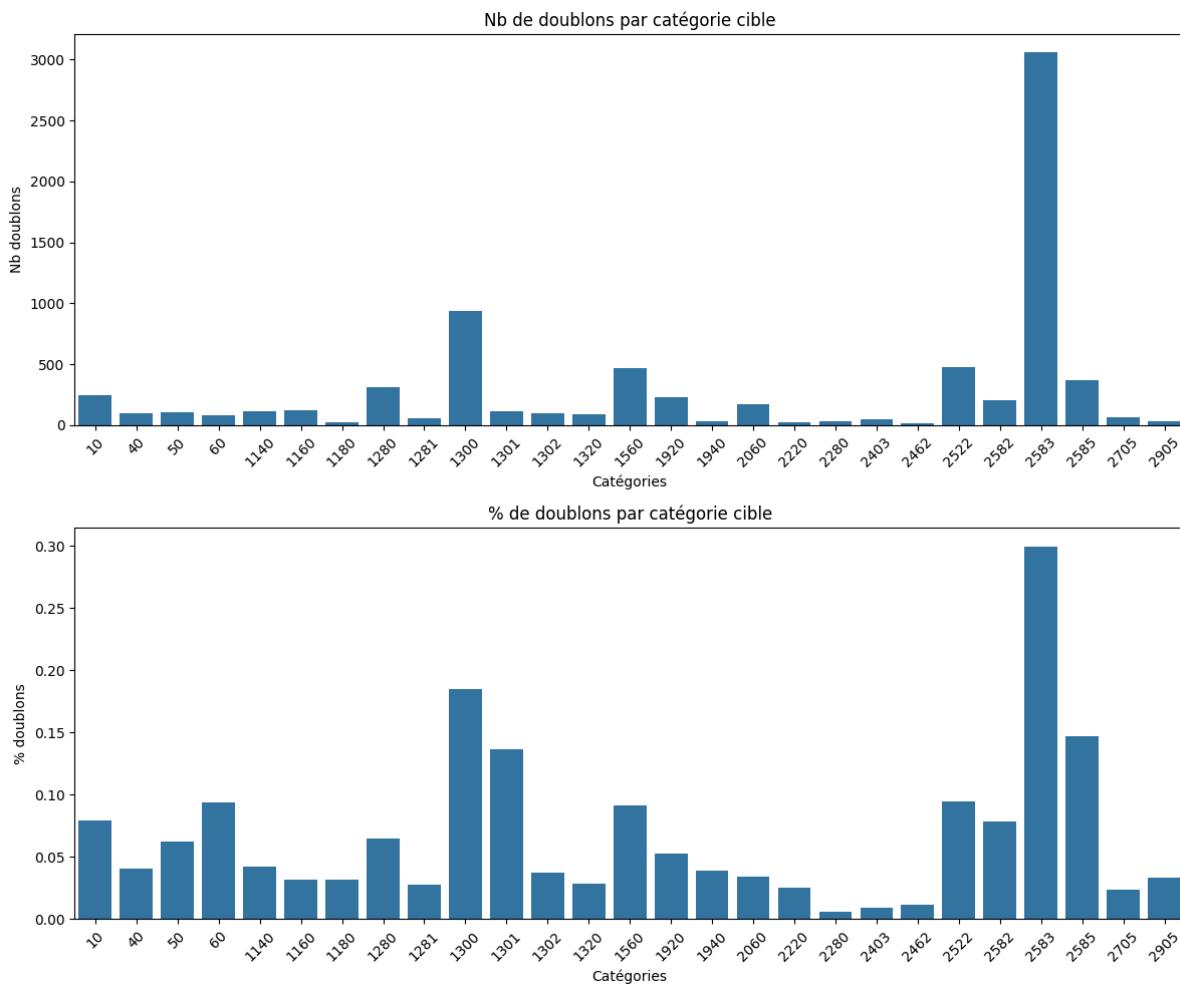
Pour garantir une bonne performance du modèle, il faudra envisager quelques traitements:

- cropper les images pour retirer le remplissage blanc inutile qui risque de polluer l'analyse sur une grande nombre d'images ou écarter ces images (ie celles avec une luminosité très élevée)
- écarter les images avec un contraste trop faible (ie <20) car ce sont souvent des images quasiment monochromes difficile à interpréter
- faire un traitement sur les images pour essayer de maintenir un niveau global du contraste supérieur à 40
- décidée d'une stratégie sur les images avec une luminosité très basse, essayer un traitement ou les retirer du dataset

Analyse des doublons

Identification des doublons via hash perceptuel

Pour identifier les doublons, nous avons utilisé la méthode du hash perceptuel (ou pHash). Cette méthode permet de créer une empreinte numérique d'une image basée sur son contenu visuel. Le hash perceptuel est conçu pour être résistant aux petites différences entre les images (par exemple, redimensionnement, compression, ajustement de couleur) ce qui permet d'identifier comme doublon des images qui sont extrêmement proches sans être totalement identiques.



Volume de doublons par catégorie:

- La catégorie 2583 se démarque très nettement avec environ 3000 doublons
- La catégorie 1300 a également un nombre important de doublons (environ 900)
- La majorité des autres catégories ont moins de 500 doublons

Pourcentage de doublons par catégorie:

- Le plus haut pourcentage de doublons est d'environ 30% (sur la catégorie 2583 qui est aussi celle avec le plus de doublon)
- Trois autres catégories ont une proportion notable autour de 15-18%
- Sinon la plupart des catégories ont moins de 10% de doublons

Conclusion

Les catégories 2583 et 1300 qui comportent les volumes et **pourcentage de doublon les plus importants** vont nécessiter une investigation particulière. Le problème que l'on pourrait rencontrer serait un **risque de surapprentissage** du modèle par rapport à ces images en doublon trop présentes, ce qui pourrait empêcher notre modèle de bien généraliser sur des images inconnues.

A noter, la méthode utilisée pour générer les hash perceptuels, dégrade l'image pour calculer son "empreinte". En particulier, l'image est redimensionnée en 32x32 pixels et convertie en niveau de gris. Les nombreuses images de produit trop petites comme celles identifiées lors de l'analyse de la luminosité ont de forte chance de générer des faux positifs qui gonflent un peu le nombre de doublons. Il faudra donc adresser ce sujet des doublons seulement après avoir fait les traitements de cropping ou de suppressions nécessaire sur les images problématiques.

Preprocessing des images

Centrage et standardisation des images

Notre objectif ici est de préparer les images brutes pour qu'elles soient plus cohérentes, centrées sur l'objet d'intérêt, et standardisées en taille pour améliorer la performance des modèles de classification.

Voici les différents traitements que nous avons appliqué pour atteindre cet objectif:

1. **Suppression du fond blanc** : On binariser l'image et on inverse le seuil afin de faciliter la détection des contours par la suite
2. **Détection des contours** : On récupère les coordonnées du plus grand objet détecté, en supposant que c'est le principal objet d'intérêt
3. **Recadrage autour de l'objet** : On supprime les zones vides/blanches autour de l'objet cela permet de fournir au modèle une image moins "bruitée" et plus "concentrée" en informations
4. **Redimensionnement** : On normalise les dimensions des images ce qui est crucial pour les modèles de deep learning

Cette étape de preprocessing est essentielle pour plusieurs raison :

- **Réduction du bruit** : en supprimant le fond, on empêche le modèle d'apprendre sur des pixels inutiles.
- **Concentration sur l'essentiel** : on force le modèle à se focaliser sur l'objet à reconnaître.
- **Homogénéité des données** : avoir des images de même taille et centrées améliore la convergence du modèle.
- **Gain de performance** : moins de variance inutile entre les images une meilleur précision et un meilleur rappel

Traitement des doublons et valeurs manquantes

Ici, nous avons commencé par extraire les caractéristiques nécessaires pour identifier les doublons et les images que l'on pourrait considérer comme des valeurs manquantes.

Les caractéristiques qui nous ont semblé intéressantes sont les suivantes:

- **Contours** : Le nombre de pixels correspondant à des contours dans l'image
- **Ecart type moyen des canaux de couleur** : utile pour identifier les images monochromes
- **Hash perceptuel de l'image** : utile pour détecter des doublons ou des variantes trop similaires d'une même image
- **Variance moyenne des canaux de couleur** :
- **Ratio de la couleur dominante** : utile pour identifier les images selon une teinte principale
- **Luminosité moyenne** :

Pour les doublons le traitement a été simple nous avons pu les écarter facilement par comparaison du hash perceptuel.

En combinant l'analyse des caractéristiques avec une vérification visuelle des images nous avons pu identifier celles à considérer comme des valeurs manquantes.

Par exemple,

- des images monochromes
- des images "placeholder" utilisées par défaut en l'absence d'une image du produit sur certains sites
- des images de produit beaucoup trop lumineuses ou avec trop peu de contour pour être exploitable

Rééquilibrage des classes

Vu le déséquilibre de classes observé pendant l'analyse exploratoire, il est nécessaire d'appliquer des méthodes de rééquilibrage pour éviter les risques d'overfitting.

Nous avons choisi d'utiliser la **médiane comme seuil d'équilibrage**. Cette stratégie simple nous semble assez efficace pour équilibrer notre dataset sans tomber dans les excès du sur/sous-échantillonnage et plus robuste que la moyenne qui serait tirée vers le haut à cause des classes très dominantes. Cette stratégie nous permet également de maintenir une taille de dataset raisonnable ce qui est plus rapide à entraîner.

Pour réduire le risque d'overfitting sur les petites classes nous ajouterons également une **étape d'augmentation** dans notre pipeline d'entraînement afin de **de générer de la diversité** afin d'aider notre modèle à mieux généraliser.

Classification du problème

Le défi qui nous est proposé porte sur le thème de la classification taxonomique à grande échelle, dont l'objectif est de prédire la catégorie de chaque produit dans l'arborescence taxonomique. La particularité de ce projet est qu'il s'agit d'une problématique de classification bimodale car nous devons travailler sur deux types différents de données en entrée, du texte et des images.

Pour comparer nos modèles nous avons choisi d'utiliser comme métrique principale le **macro-F1 score**. Le macro-F1 score est particulièrement pertinent dans notre cas de classification, surtout en raison du **déséquilibres de classes** (ce qui est quasi systématique dans les catalogues e-commerce).

En effet, lorsqu'on utilise le macro-F1 score, on calcule le F1-score pour chaque classe indépendamment, puis on fait la moyenne de ces scores, **en accordant un poids égal à chaque classe**, quelle que soit sa fréquence. Cela permet de mieux évaluer les performances globales du modèle, en particulier dans les contextes déséquilibrés où certaines classes sont sous-représentées.

autres métriques utilisées en complément:

- **accuracy** : proportion de prédictions dont la catégorie prédite correspond exactement à la vérité terrain
- **weighted-F1 score** : pour avoir une idée de la performance globale sur le jeu de données tel qu'il est distribué
- **F1 score par classe** : pour identifier les classes sur lesquelles le modèle échoue ou performe

Choix du modèle et optimisation

Choix du modèle pour l'analyse d'image

Benchmark des modèles

Nous avons commencé notre étude par un benchmark de plusieurs modèles TensorFlow/Keras que nous avons entraînés en utilisant la même structure de pipeline.

Le pipeline comprenait les étapes suivante :

- Définition d'une couche d'entrées
- Augmentation des données
- Prétraitement spécifique au modèle
- Passage dans le modèle de base pré-entraîné
- Couche d'aplatissement (Global Average Pooling)
- Couche de normalisation
- Couches entièrement connectées (denses)
- Couche de classification (activation softmax)

L'étape d'**augmentation des données** applique des transformations aléatoires à l'image pendant l'entraînement

- **Flip horizontal** : pour généraliser l'orientation
- **Rotation** : pour apprendre l'invariance aux angles
- **Zoom** : pour rendre le modèle robuste aux tailles variables
- **Contraste** : pour s'adapter à diverses conditions lumineuses

L'objectif étant d'améliorer la robustesse et la généralisation du modèle avec un jeu de données plus varié artificiellement.

L'étape de **prétraitement spécifique au modèle** permet de préparer les données dans un format compatible avec ce que le modèle de base attend (ex : normalisation pour ResNet, centrage pour EfficientNet, etc.)

Voici les résultats obtenus avec des modèles complètement gelés.

Modèle	Test Accuracy	Test Loss	Macro F1 Score	Params	Training Time (s)
ResNet50	0,5207	1,6162	0,5201	23643035	4751
ResNet101	0,5172	1,6164	0,5145	42713499	4865
EfficientNetB0	0,5085	1,6448	0,5047	4084158	3262
EfficientNetB3	0,5057	1,6907	0,5006	10825034	5587
VGG16	0,444	1,932	0,4383	14728539	5898
VGG19	0,4352	1,9563	0,427	20038235	6453

Le benchmark met en évidence la supériorité de **ResNet50**, qui offre les meilleures performances en termes de précision et de macro-F1 score, bien qu'au prix d'un nombre de paramètres élevé. Toutefois, **EfficientNetB0** se démarque par son **efficacité**, atteignant des résultats comparables avec **beaucoup moins de paramètres** et un temps d'entraînement **significativement plus court**. Les modèles VGG, quant à eux, sont **largement dépassés** dans ce contexte.

Nous avons ensuite essayé d'optimiser le résultat en adoptant une approche de **transfert learning** afin de profiter de l'entraînement initial des modèles et de l'apprentissage déjà acquis tout en essayant de l'entraîner à reconnaître plus spécifiquement les types d'objets que nous cherchons à classifier.

Pour ce faire, nous avons dégelé le dernier bloc de chaque modèle, soit environ 15% des couches (bloc5 pour VGG16, conv5 pour ResNet50 et block7a pour EfficientNetB0). Voici les résultats obtenus:

Modèle	Test Accuracy	Test Loss	Macro F1 Score	Params	Training Time (s)
EfficientNetB0	0.6297	1.4942	0.6284	4420542	2709
ResNet50	0.5854	1.578	0.5885	24158363	3475
VGG16	0.4558	1.8966	0.4428	14885979	9692

Analyse post-fine-tuning

Modèle	Gain F1 score	Gain Params	Gain Training Time
EfficientNetB0	+0.1237	4.0M → 4.4M	↓ 3262 → 2709
ResNet50	+0.0684	23.6M → 24.1M	↓ 4751 → 3475
VGG16	+0.0045	≈14.7M	↑ 5898 → 9692

EfficientNetB0

- **Gagne largement en performance** (+12 pts de F1 score), dépassant tous les autres modèles
- **Plus rapide à entraîner** que ResNet50 ou VGG16.
- Toujours **ultra-léger (4.4M params)**, donc très rentable.
- **Meilleur compromis général** : précision, score F1, temps et taille.

ResNet50

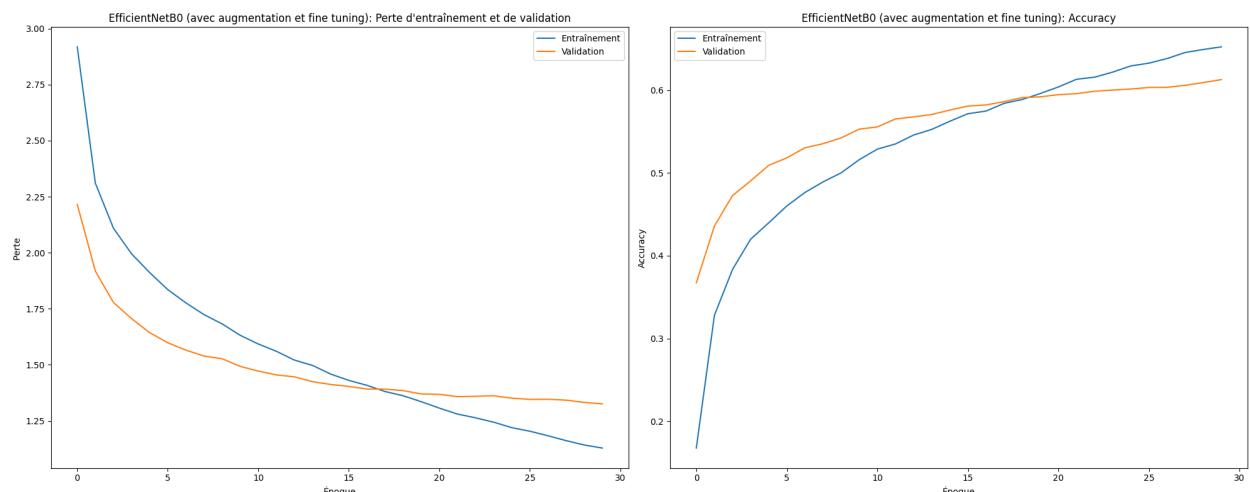
- Il s'améliore aussi, mais **moins que EfficientNetB0**.
- Reste performant, mais **plus coûteux** en paramètres.
- **Temps d'entraînement réduit** post-fine-tuning, ce qui est un bon point.

VGG16

- A très peu bénéficié du fine-tuning.
- Les performances restent faibles.
- Temps d'entraînement **le plus long**, pour un gain minime.

Après fine-tuning, **EfficientNetB0 devient le modèle le plus performant** avec une macro-F1 Score de **0,6284**, tout en conservant une faible complexité et un temps d'entraînement réduit. Il surpassé **ResNet50**, pourtant initialement en tête, qui reste compétitif mais plus coûteux en ressources. Le modèle **VGG16**, quant à lui, reste en retrait malgré une durée d'entraînement bien plus longue.

Par ailleurs, l'analyse des courbes d'entraînement nous montre que le modèle semble **généraliser correctement**. Le gap entre entraînement et validation est faible, ce qui suggère **peu d'overfitting**.



Analyse de performance du modèle

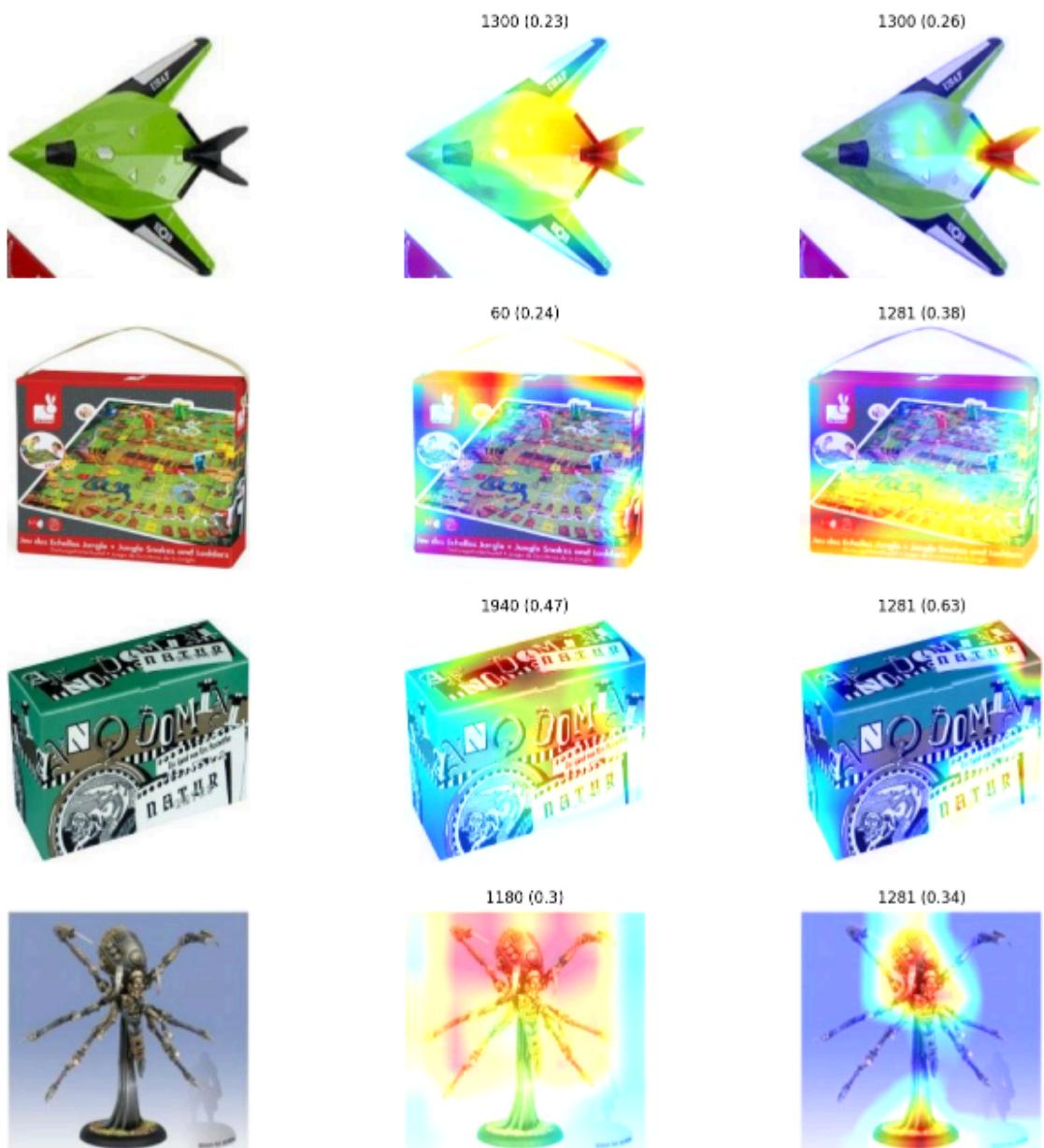
Interprétabilité avec Grad-CAM

Nous avons utilisé la méthode Grad-CAM (Gradient-weighted Class Activation Mapping) pour analyser certains résultats surprenants afin d'expliquer l'influence du fine tuning sur les performances de notre modèle.

Si globalement le fine tuning du modèle à fait grandement progresser la performance du modèle certaines catégories reste moins bien reconnue.
 Voici une exemple d'analyse faite sur le cas de cas de la classe **1281 "Jeux société enfants"** qui bien qu'ayant profité du fine tuning (F1 score +120%) reste l'une des moins bien reconnue.

Dans les exemples suivant, nous affichons de gauche à droite

1. l'image d'origine
2. le Grad-CAM obtenu en sortie du **modèle de base** (avec la classe prédite et le niveau de confiance)
3. le Grad-CAM obtenu en sortie du **modèle avec fine tuning** (avec la classe prédite et le niveau de confiance)



Analyse Grad-CAM pour le modèle de base

Boîte de jeu jungle Janod:

- **Prédiction incorrecte :** ✗ 60 "Consoles de jeux" (0.24) - faible confiance
- **Attention sur l'emballage :** Grad-CAM réparti sur toute la boîte
- **Confusion catégorielle :** Le modèle confond avec une console, probablement à cause de l'aspect "boîte rectangulaire" et des couleurs vives

Boîte de jeu :

- **Prédiction incorrecte :** ✗ 1940 (0.47) - confiance modérée
- **Attention sur le texte :** Focus principal sur les éléments textuels de l'emballage
- Le modèle semble se concentrer sur les caractéristiques graphiques de la boîte (dessins, texte etc...)

Figurine squelette :

- **Prédiction incorrecte :** ✗ 1180 "Figurines et jeux de rôles" (0.3) - faible confiance
- **Attention diffuse :** Activations réparties sur toute la figurine
- Le modèle peine à identifier les caractéristiques discriminantes

Analyse Grad-CAM pour le modèle avec fine tuning

Boîte de jeu jungle Janod:

- **Prédiction correcte :** ✓ 1281 "Jeux société enfants" (0.38) - amélioration de +58%
- **Attention plus focalisée :** le Grad-CAM est concentré sur les éléments clés (illustration, logos)
- **Classification correcte** mais confiance encore modérée

Boîte de jeu :

- **Prédiction correcte :** ✓ 1281 "Jeux société enfants" (0.63) - mais confiance modérée
- **Attention plus focalisée :** Concentration sur des zones spécifiques de l'emballage
- Le fine-tuning a permis d'identifier correctement la classe

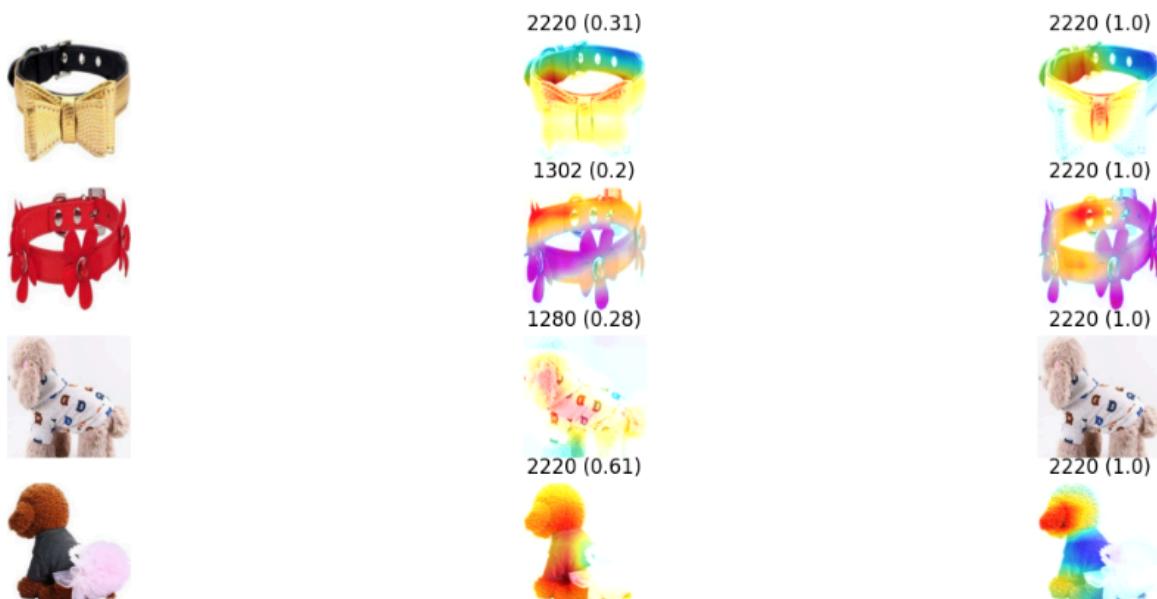
Figurine squelette :

- **Légère amélioration :** ✓ 1281 "Jeux société enfants" (0.34) - prédiction correcte mais confiance très faible
- **Attention concentrée :** Focus intense sur le centre de la figurine (torse/tête)
- Bien que correcte, la prédiction reste peu confiante

Conclusion

Le fine-tuning a aidé à identifier correctement ces objets comme des jeux de société pour enfants, mais la confiance reste modérée car cette **catégorie semble très hétérogène**. La diversité des emballages, styles graphiques et objets dans cette catégorie semble expliquer pourquoi le modèle, même après fine-tuning, maintient une confiance modérée. On remarque également que certains objets occupent une **place ambiguë** dans la taxonomie des produits. La Figurine squelette et l'avion auraient aussi pu être rangés initialement dans d'autres catégories (1180 - "Figurines et jeux de rôles" ou 1300 - "Modélisme").

Si on compare ces résultats avec ceux obtenus sur la catégorie 2220 - "animalerie" qui a également profité d'une grosse amélioration de performance avec le fine tuning on constate que celle-ci semble un peu plus homogène et cela se traduit dans le niveau de confiance important des prédictions ainsi que dans le f1 score de cette catégorie (0.84).



Ce cas renforce l'hypothèse que le fine-tuning est **particulièrement efficace** pour les corrections catégorielles complexes, mais que certaines classes restent **intrinsèquement difficiles** à classifier avec une haute confiance en raison de leur hétérogénéité visuelle. La classe 1281 "Jeux société enfants" semble être un cas d'école de catégorie e-commerce complexe : regroupant des objets très variés sous un critère fonctionnel plutôt que visuel.

Cela met en évidence l'importance de définir une taxonomie appropriée afin d'optimiser la classification automatique. Dans le cas présent, une taxonomie à plusieurs niveaux par exemple pourrait grandement aider.

Exemple :

Classe 1281 "Jeux société enfants" : Subdiviser en sous-catégories plus homogènes visuellement

- 1281a : Boîtes de jeux de plateau
- 1281b : Figurines et accessoires de jeux
- 1281c : Jeux éducatifs

Pour résoudre ce problème, nous avons également imaginé des solutions techniques à ce problème sans avoir malheureusement eu le temps de les mettre en œuvre.

Par exemple,

- Tester un modèle comme **Vision Transformers (ViT)** : qui à priori pourrait offrir une meilleure capture des relations spatiales globales
- Combiner plusieurs architectures (CNN + ViT) pour exploiter leurs forces respectives

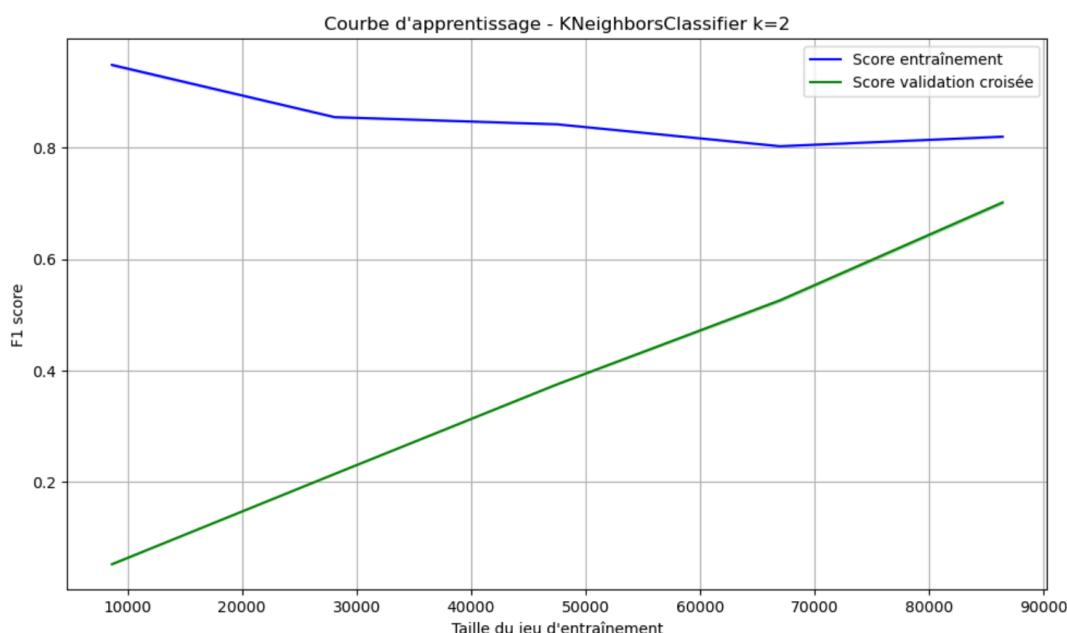
Choix du modèle pour l'analyse du texte

Nous avons commencé par tester plusieurs modèles simples tels que le modèle KNN, Decision Tree, ou encore Linear SVM. Les différents modèles testés sont décrits ci-après.

Modèles simples - KNN

Le premier modèle testé sur le texte est le modèle KNN. Nous avons joué sur le paramètre `n_neighbors` qui détermine le nombre de voisins utilisés pour la classification. Pour chacun des `n_neighbors` testés (2, 5, et 10), nous avons tracé la courbe d'apprentissage :

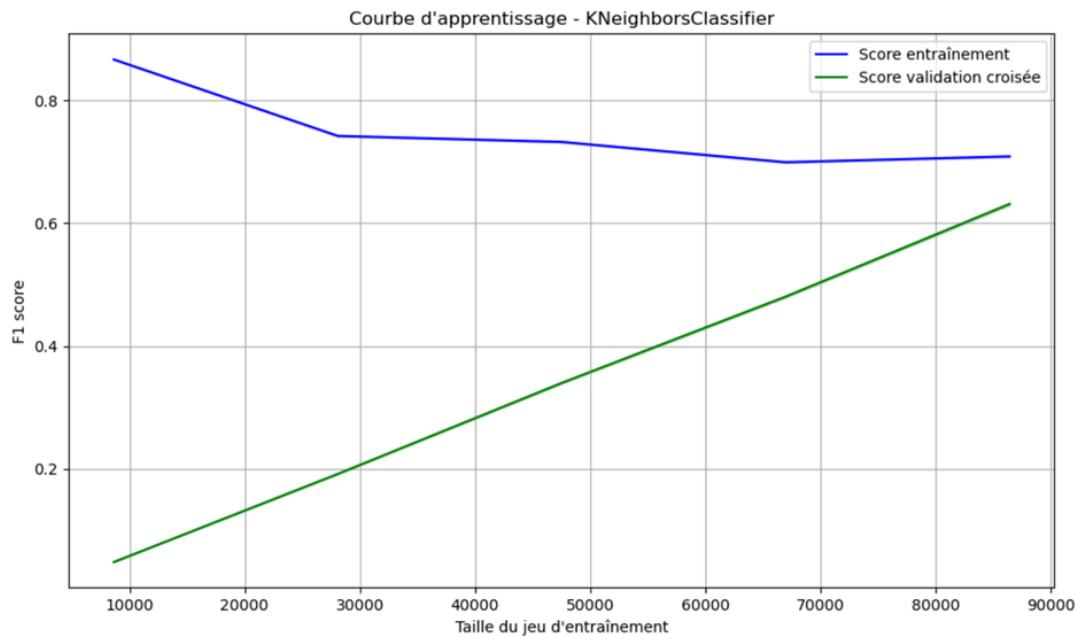
Courbe d'apprentissage pour `n_neighbors` = 2



- Le score d'entraînement est élevé / bon au départ, puis chute à mesure que la taille de l'échantillon augmente, avant de remonter légèrement pour finir à 0.82. Le modèle s'adapte *relativement* bien aux données d'entraînement.
- Le score de validation commence très faible (0.05) et croît de pair avec l'augmentation de la taille de l'échantillon jusqu'à atteindre 0.70

L'écart entre le score d'entraînement et le score de validation est conséquent (82% vs 72%). Cela traduit un overfitting sur les données d'entraînement et une mauvaise généralisation du modèle.

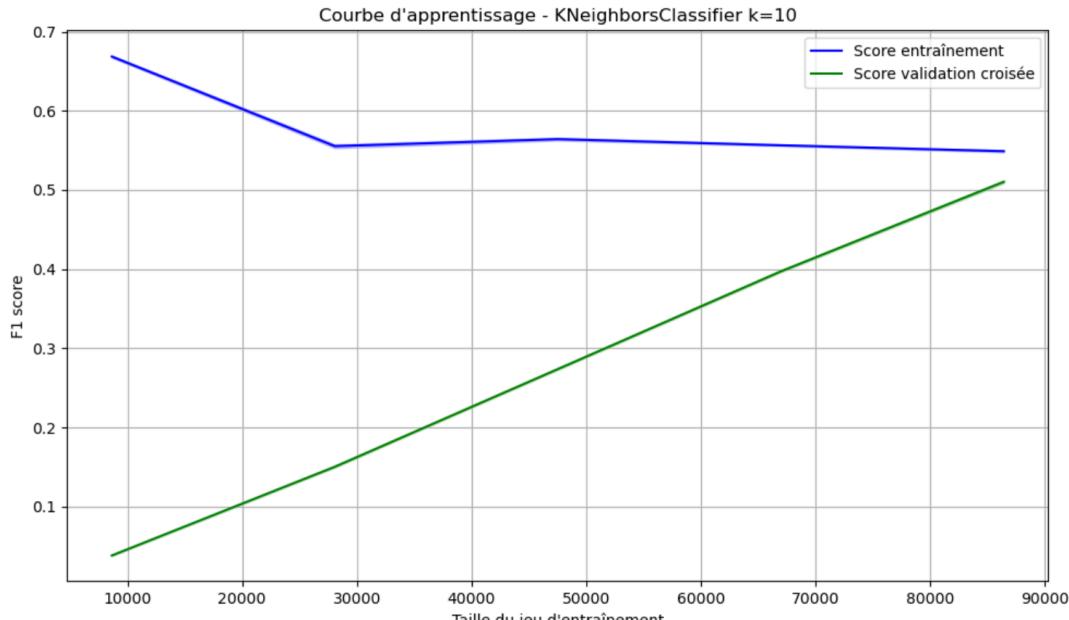
Courbe d'apprentissage pour n_neighbors = 5



- Le score d'entraînement est élevé au départ, puis chute à mesure que la taille de l'échantillon augmente en passant de 0.87 à 0.70. Le modèle s'adapte *relativement* bien aux données d'entraînement même si son score final est inférieur au test précédent
- Le score de validation commence très faible (0.05) et croît de pair avec l'augmentation de la taille de l'échantillon jusqu'à atteindre 0.62

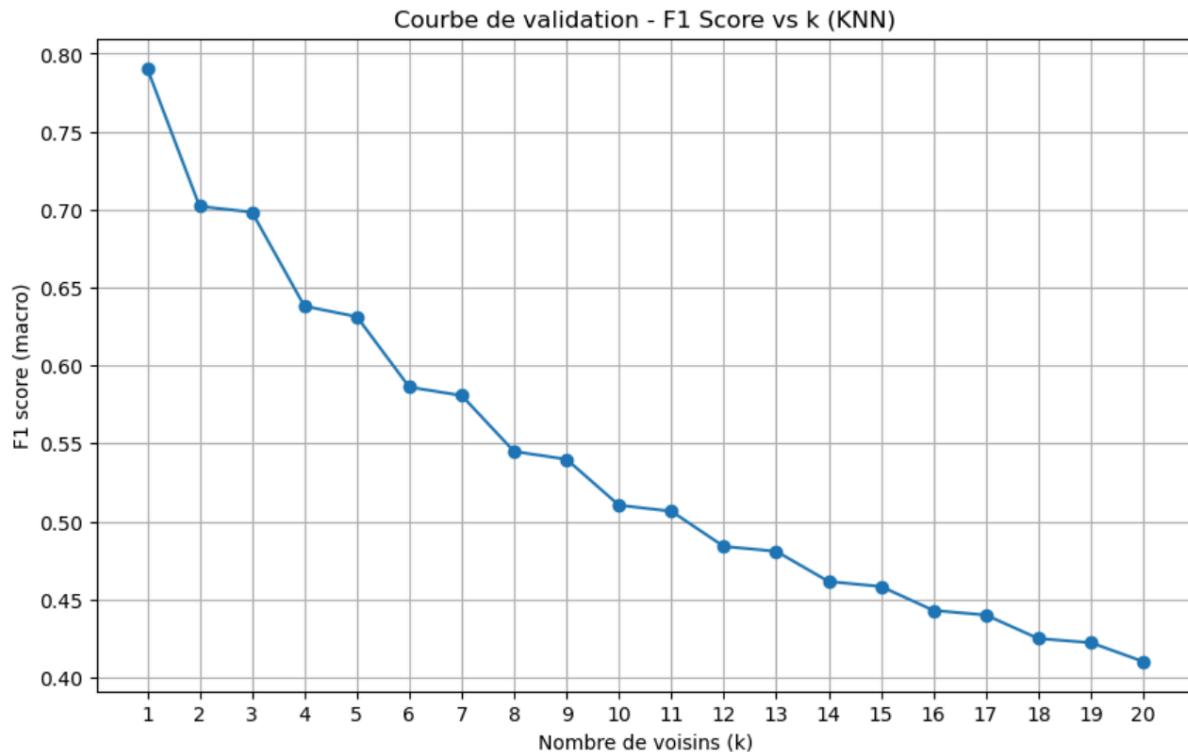
On observe que l'écart entre le score d'entraînement et le score de validation est nettement plus faible que pour le test précédent. Il y a beaucoup moins d'overfitting, ce modèle semble être un meilleur compromis bien que les scores aient chuté.

Courbe d'apprentissage pour n_neighbors = 10



On observe qu'avec l'augmentation du paramètre `n_neighbors`, l'écart entre le score d'entraînement et le score de validation continue de diminuer. Overfitting est réduit mais le score de validation du modèle continue de chuter et n'atteint plus que les 0.52.

En complément de ces courbes d'apprentissage, nous avons tracé l'évolution du F1-Score moyen en fonction de `n_neighbors` :



On observe que la courbe diminue très rapidement à mesure que `n_neighbors` augmente. Au vu des différentes analyses précédentes, on en conclut que :

- Plus `n_neighbors` est petit, et plus le F1-score est élevé,
- mais également, que plus `n_neighbors` est petit, et plus l'over-fitting vu sur les courbes d'apprentissage est élevé.

Nous avons donc fait le compromis de choisir `n_neighbors = 5` (faible diminution du F1 score par rapport à 4) afin d'avoir un modèle relativement robuste.

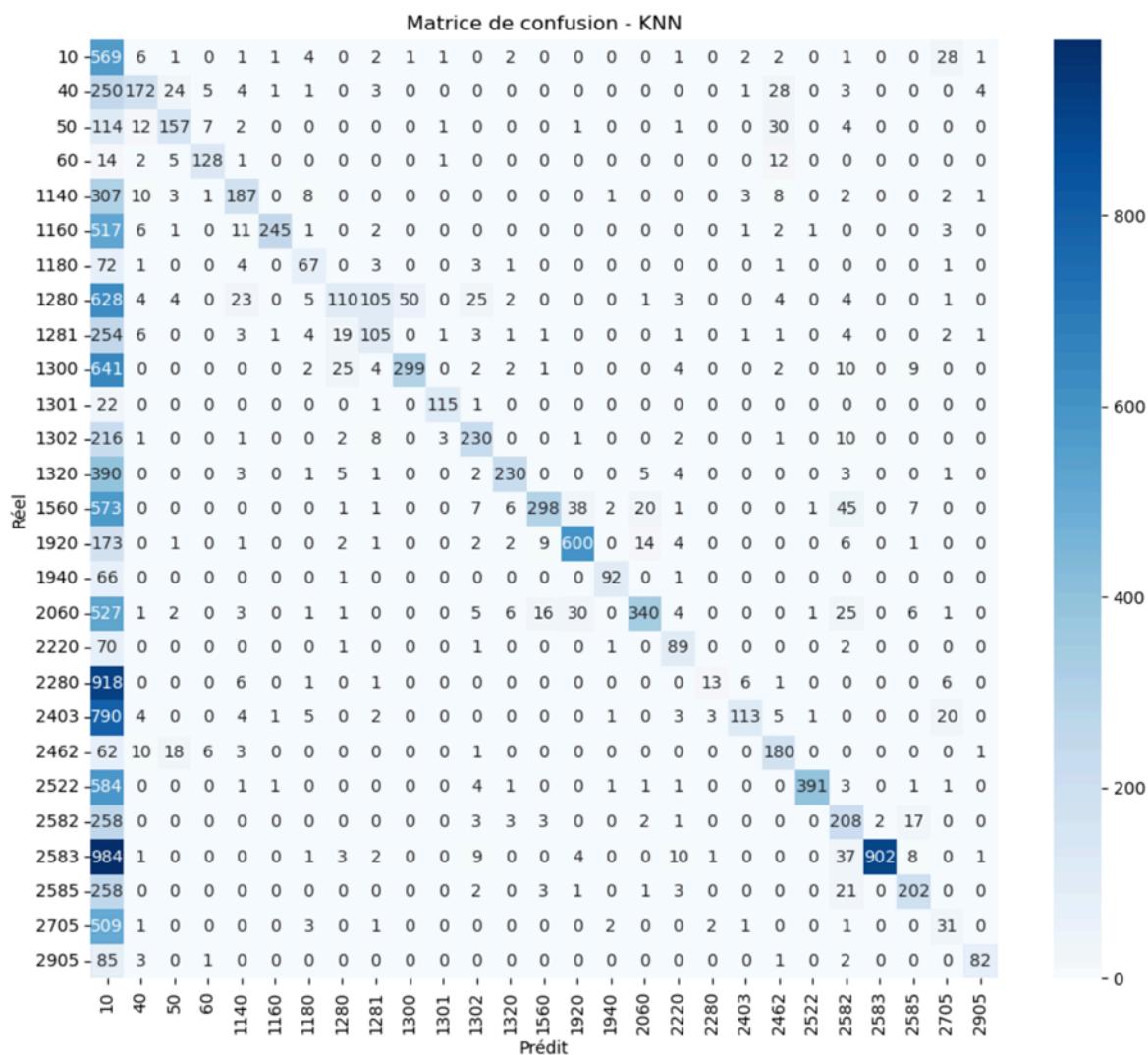
Rapport de classification pour n_neighbors = 5

Classification Report (KNN):				
	precision	recall	f1-score	support
10	0.06	0.91	0.11	623
40	0.72	0.35	0.47	496
50	0.73	0.48	0.58	329
60	0.86	0.79	0.82	163
1140	0.72	0.35	0.47	533
1160	0.98	0.31	0.47	790
1180	0.64	0.44	0.52	153
1280	0.65	0.11	0.19	969
1281	0.43	0.26	0.32	408
1300	0.85	0.30	0.44	1001
1301	0.94	0.83	0.88	139
1302	0.77	0.48	0.59	475
1320	0.90	0.36	0.51	645
1560	0.90	0.30	0.45	1000
1920	0.89	0.74	0.80	816
1940	0.92	0.57	0.71	160
2060	0.89	0.35	0.50	969
2220	0.67	0.54	0.60	164
2280	0.68	0.01	0.03	952
2403	0.88	0.12	0.21	952
2462	0.65	0.64	0.64	281
2522	0.99	0.39	0.56	990
2582	0.53	0.42	0.47	497
2583	1.00	0.46	0.63	1963
2585	0.80	0.41	0.54	491
2705	0.32	0.06	0.10	551
2905	0.90	0.47	0.62	174
accuracy			0.37	16684
macro avg	0.75	0.42	0.49	16684
weighted avg	0.79	0.37	0.45	16684

Le rapport de classification nous permet d'observer une forte disparité entre les scores des différentes classes, allant de seulement **0.10** et **0.11** pour les classes 2705 et 10, jusqu'à **0.88** pour la classe 1301.

Globalement, le modèle KNN n'est pas satisfaisant avec un F1-score moyen de seulement **0.49** et espérons obtenir de meilleurs résultats sur les prochains modèles.

Matrice de confusion

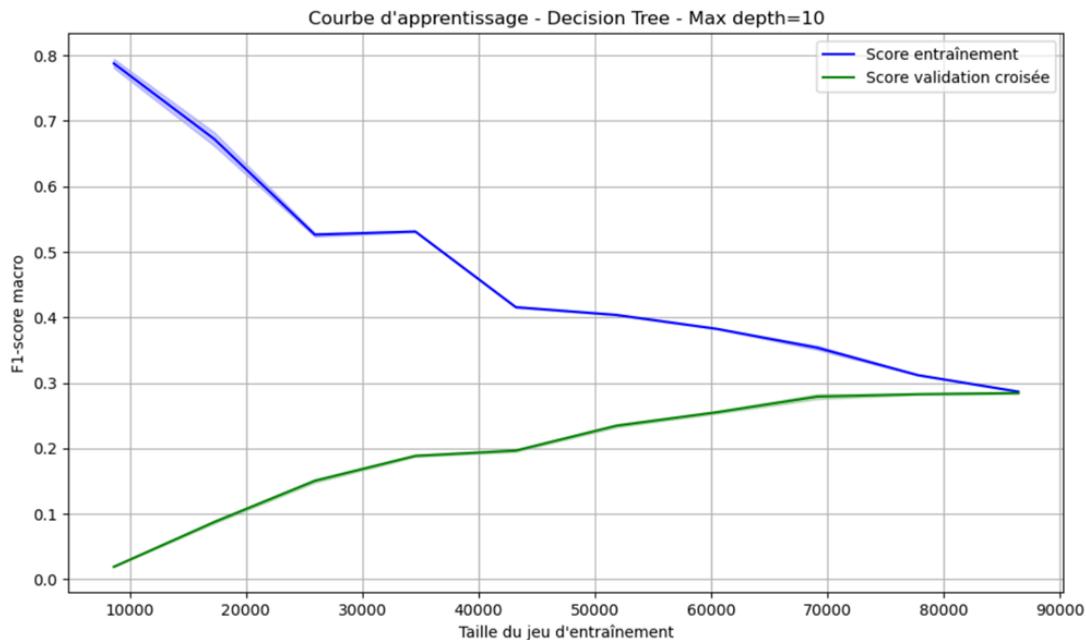


La matrice de confusion met en évidence le fait que le modèle KNN a tendance à sur-prédir la classe 10 dans quasiment tous les cas. Par exemple : pour les 952 occurrences de la classe 2280, seuls 13 ont été correctement prédis et 918 ont été prédis en classe 10.

Modèles simples - Decision Tree

Nous avons joué sur le paramètre `max_depth` qui détermine la profondeur maximale de l'arbre de décision. Pour chacun des `max_depth` testés (10, 50, et sans limite), nous avons tracé la courbe d'apprentissage :

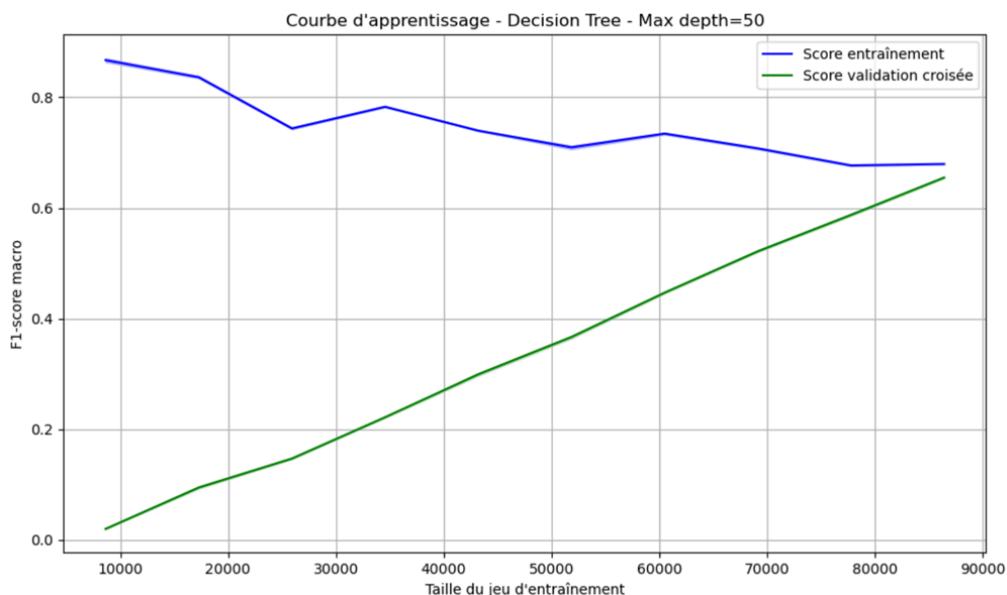
Courbe d'apprentissage pour `max_depth = 10`



- Le score d'entraînement chute très rapidement passant de ~0.80 à ~0.28. La chute de performance sur ses propres données indique que le modèle est trop contraint / sous-ajusté.
- Le score de validation croisée croît légèrement, mais reste très faible. Il finit par rejoindre le score d'entraînement en passant de 0.05 à ~0.28.

Le modèle est trop simple pour apprendre (pas assez de profondeur), même avec près de 90000 données. Les deux courbes finissent par se rapprocher à un faible niveau de performance. On conclut donc qu'il y a un fort under-fitting.

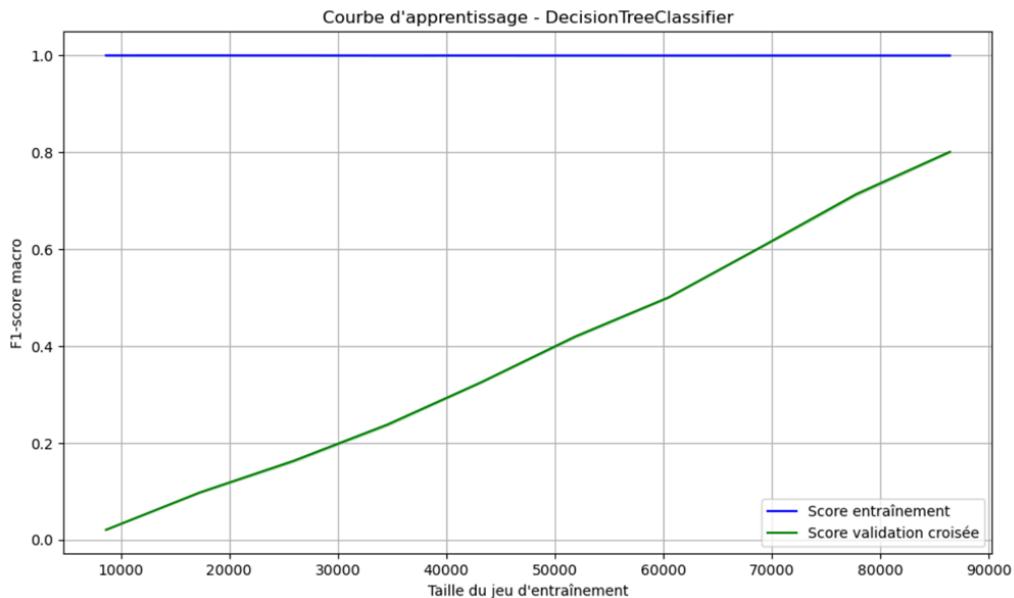
Courbe d'apprentissage pour max_depth = 50



- Le score d'entraînement décroît mais reste élevé en passant de 0.85 à 0.68. La chute est beaucoup moins prononcée que pour le modèle avec $\text{max_depth} = 10$. Le modèle mémorise bien les données d'entraînement, même avec de grands volumes.
- Le score validation croisée croît progressivement en passant de 0.02 à 0.65. À mesure que l'on ajoute plus de données, le modèle généralise de mieux en mieux.

Les deux courbes se rejoignent quasiment, et le score final de 0.65 est bien meilleur que pour le modèle précédent qui convergeait à 0.29. Il ne semble pas y avoir d'overfitting ou d'underfitting prononcé. Cependant, même si le score est meilleur, 0.65 reste tout de même faible.

Courbe d'apprentissage sans max_depth



Pour finir, nous avons testé le modèle sans spécifier de profondeur maximale.

- Comme nous pouvions nous y attendre, le score d'entraînement est toujours égal à 1. L'arbre de décision s'adapte exactement aux données d'entraînement jusqu'à mémoriser les données. Il y a un fort problème de surapprentissage.
- Le score de validation est très faible avec peu de données et croît avec l'augmentation de la taille des données. Malheureusement, il ne parvient pas à rejoindre le score d'entraînement et plafonne à 0.81.

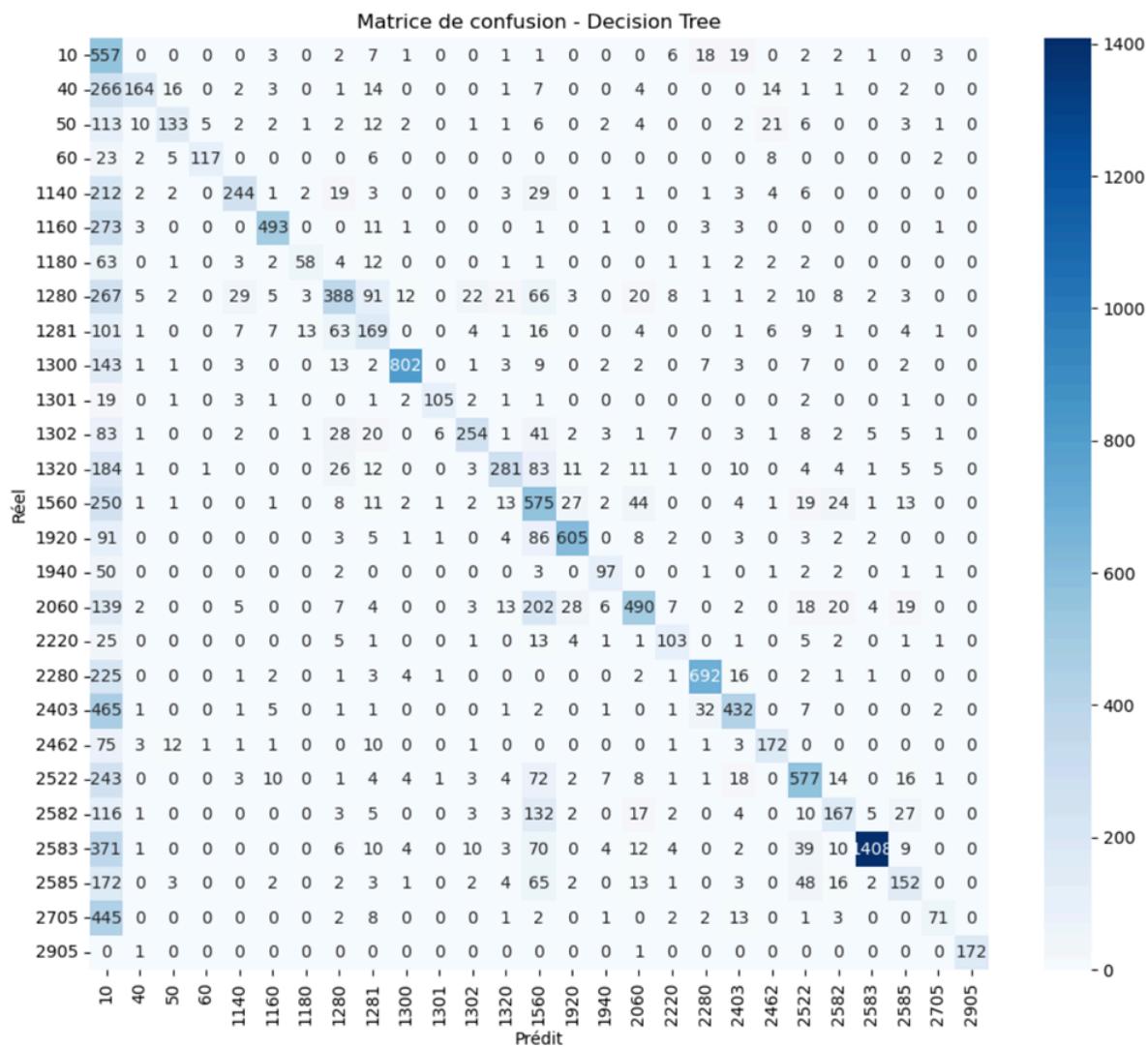
L'écart entre le score d'entraînement et le score de validation est significatif (0.19). Il y a un overfitting clair. Pour la suite, nous retiendrons donc le modèle avec $\text{max_depth} = 50$ qui semble être le meilleur compromis.

Rapport de classification

Classification Report (Decision Tree):				
	precision	recall	f1-score	support
10	0.11	0.89	0.20	623
40	0.82	0.33	0.47	496
50	0.75	0.40	0.53	329
60	0.94	0.72	0.82	163
1140	0.80	0.46	0.58	533
1160	0.92	0.62	0.74	790
1180	0.74	0.38	0.50	153
1280	0.66	0.40	0.50	969
1281	0.40	0.41	0.41	408
1300	0.96	0.80	0.87	1001
1301	0.91	0.76	0.83	139
1302	0.81	0.53	0.65	475
1320	0.78	0.44	0.56	645
1560	0.39	0.57	0.46	1000
1920	0.88	0.74	0.81	816
1940	0.75	0.61	0.67	160
2060	0.76	0.51	0.61	969
2220	0.70	0.63	0.66	164
2280	0.91	0.73	0.81	952
2403	0.79	0.45	0.58	952
2462	0.74	0.61	0.67	281
2522	0.73	0.58	0.65	990
2582	0.60	0.34	0.43	497
2583	0.98	0.72	0.83	1963
2585	0.58	0.31	0.40	491
2705	0.79	0.13	0.22	551
2905	1.00	0.99	0.99	174
accuracy			0.57	16684
macro avg	0.75	0.56	0.61	16684
weighted avg	0.76	0.57	0.62	16684

Comme pour le modèle KNN précédent, nous observons une forte disparité entre les classes, allant de 0.20 pour la classe 10, jusqu'à 0.99 pour la classe 2905. Globalement, les scores du modèle Decision Tree sont meilleurs que ceux du modèle KNN avec un **F1-score moyen de 0.61**. Cependant ce score reste faible et non satisfaisant.

Matrice de confusion

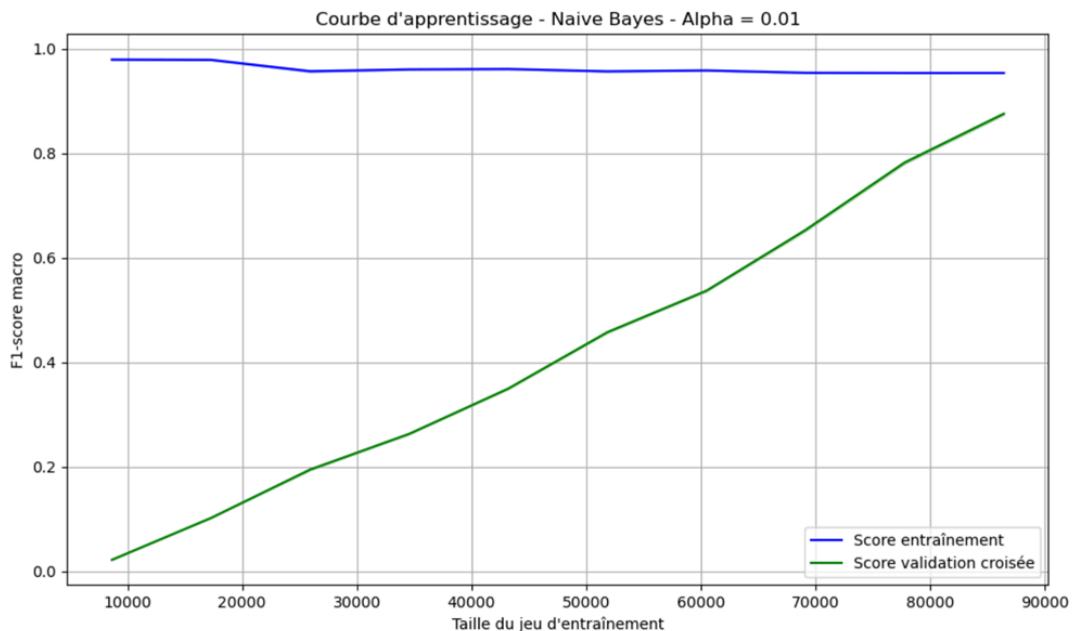


La matrice de confusion met en évidence le même problème que celui observé avec le modèle KNN : il y a une forte confusion avec la classe 10 qui est sur-prédite.

Modèles simples - Naive Bayes

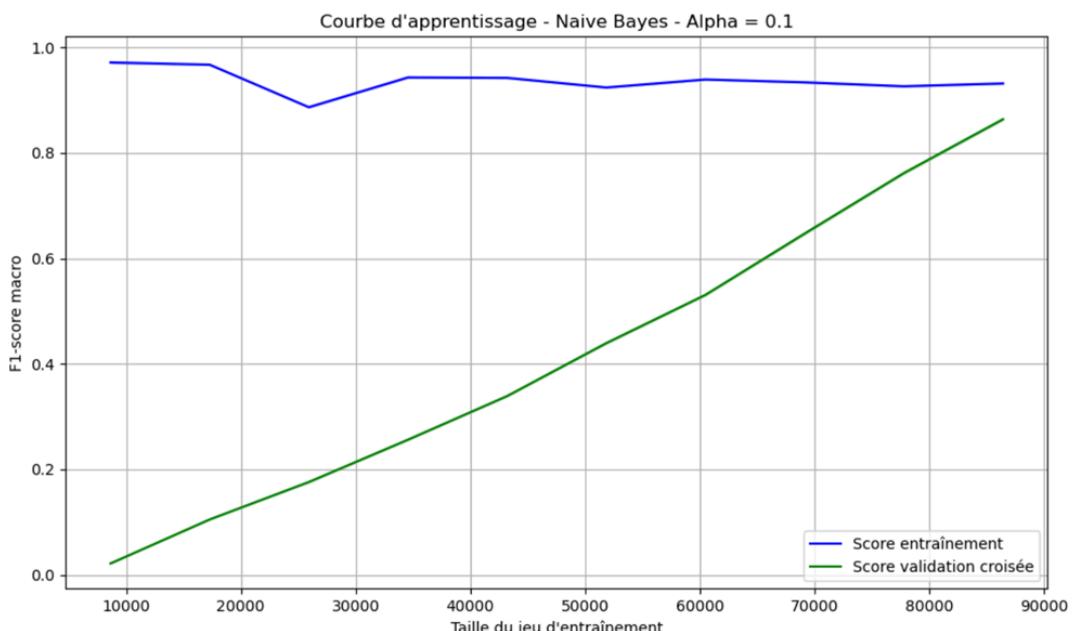
Nous avons joué sur le paramètre alpha de lissage. Voici les courbes d'apprentissage pour alpha = [0.01, 0.1, 1, 2] :

Courbe d'apprentissage pour alpha = 0.01



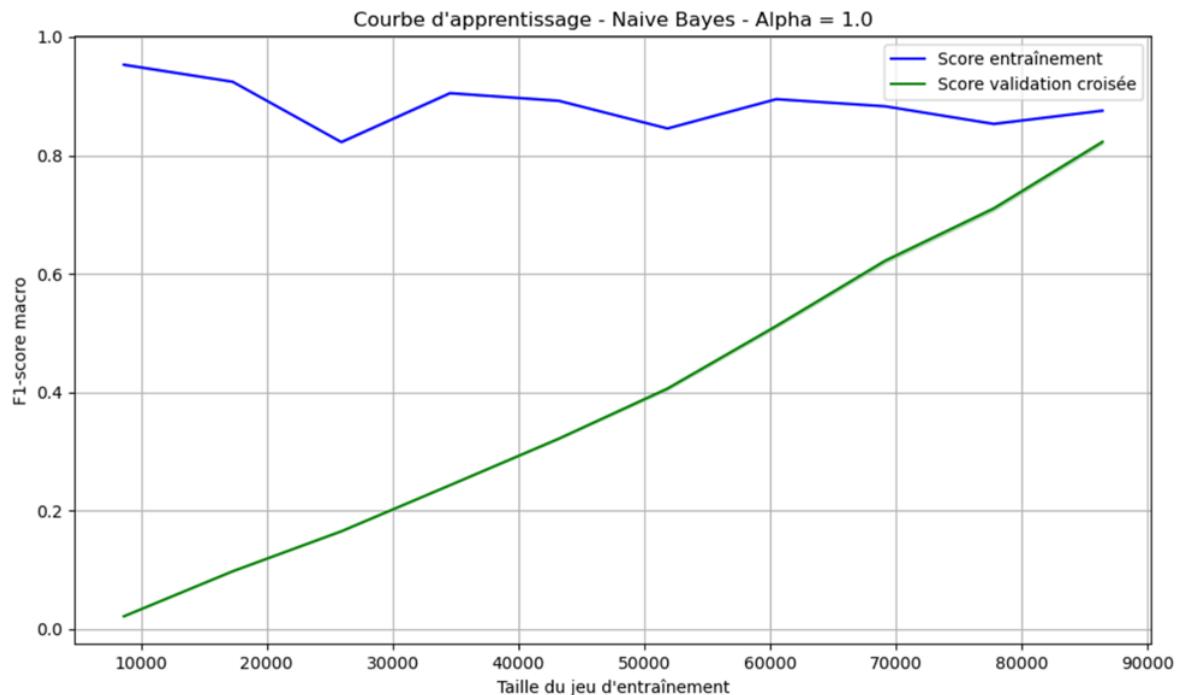
- Le score d'entraînement est très proche de 1. Le modèle surapprend / overfitting
- Le score de validation est faible (même avec beaucoup de données, il ne rejoint pas la courbe d'apprentissage) : la généralisation avec alpha = 0.01 est mauvaise.

Courbe d'apprentissage pour alpha = 0.1



- Le score d'entraînement est moins proche de 1 que le précédent. Il y a moins d'overfitting, ce qui semble plus réaliste
- Le score de validation ne rejoint toujours pas la courbe d'entraînement. La généralisation est toujours mauvaise et il y a encore de l'overfitting

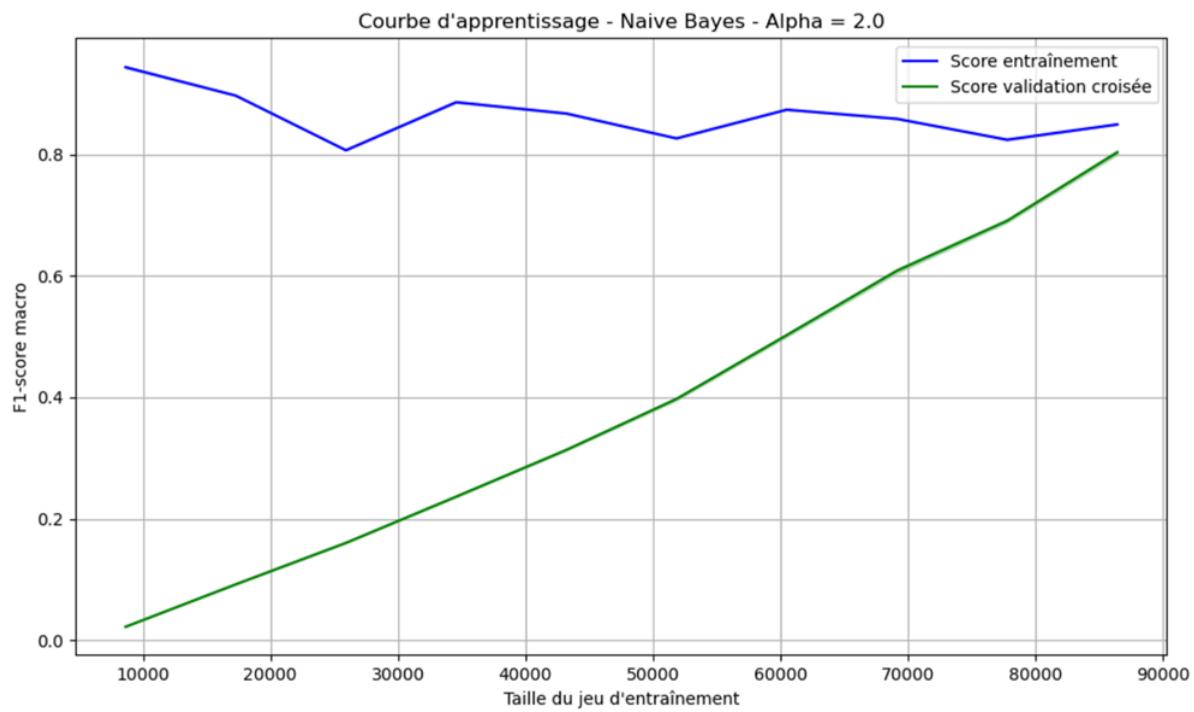
Courbe d'apprentissage pour alpha = 1



- Le score entraînement tourne autour des 0.9, et est donc plus faible que précédemment
- Le score de validation fini autour des 0.82 et reste plus faible que le modèle avec alpha = 0.1

Le modèle avec alpha = 1 semble moins performant que celui avec alpha = 0.1

Courbe d'apprentissage pour alpha = 2



- Le score d'entraînement continue de chuter : environ 0.85
- Le score de validation continue également de chuter : environ 0.81

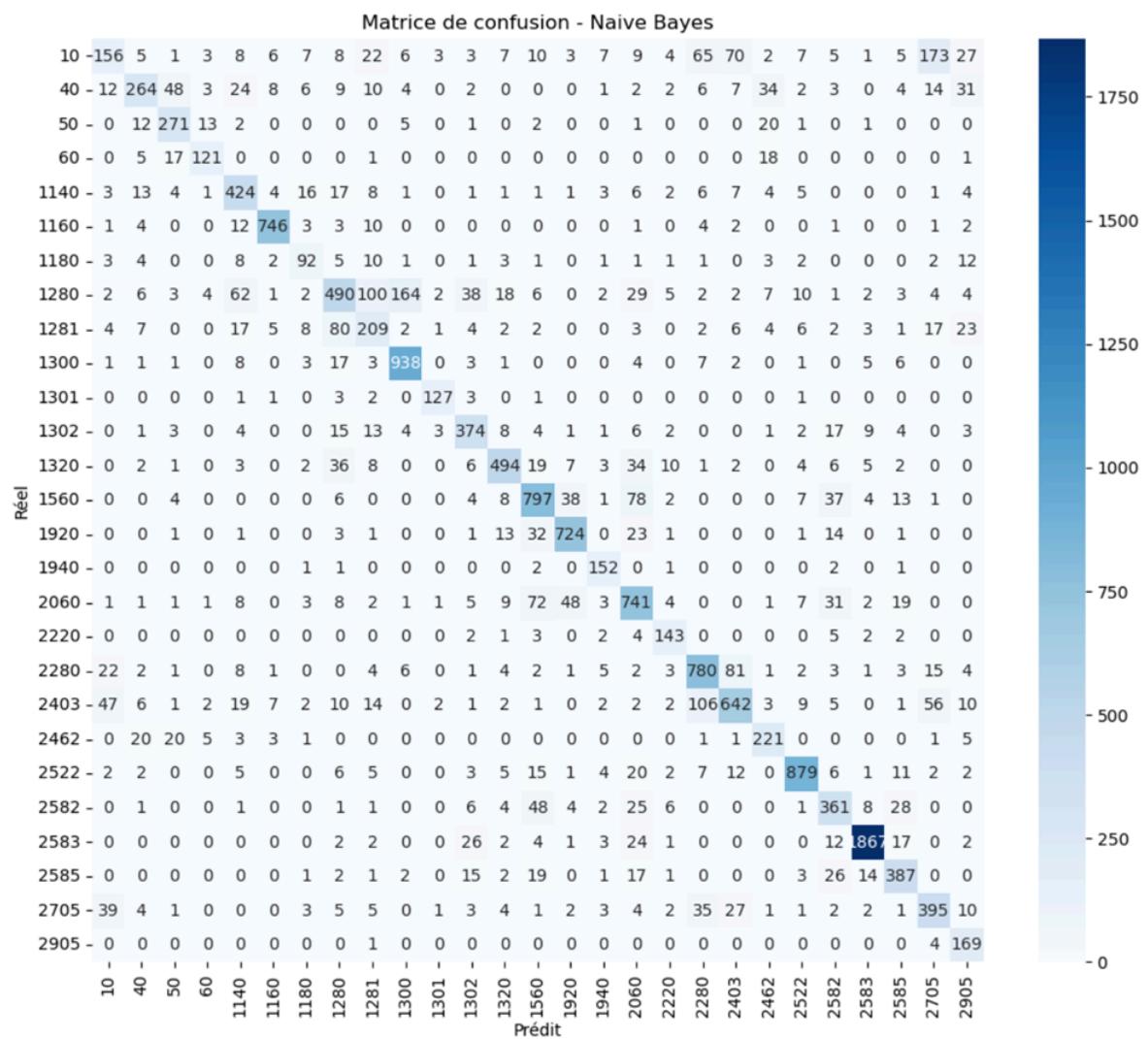
En conclusion, le modèle perd en puissance. Il y a certainement un sous-apprentissage par rapport au modèle alpha = 0.1. Nous allons donc sauvegarder le modèle avec alpha = 0.1 qui semble être le meilleur compromis.

Rapport de classification pour alpha = 0.1

Classification Report (Naive Bayes):				
	precision	recall	f1-score	support
10	0.53	0.25	0.34	623
40	0.73	0.53	0.62	496
50	0.72	0.82	0.77	329
60	0.79	0.74	0.77	163
1140	0.69	0.80	0.74	533
1160	0.95	0.94	0.95	790
1180	0.61	0.60	0.61	153
1280	0.67	0.51	0.58	969
1281	0.48	0.51	0.50	408
1300	0.83	0.94	0.88	1001
1301	0.91	0.91	0.91	139
1302	0.74	0.79	0.76	475
1320	0.84	0.77	0.80	645
1560	0.76	0.80	0.78	1000
1920	0.87	0.89	0.88	816
1940	0.78	0.95	0.85	160
2060	0.72	0.76	0.74	969
2220	0.74	0.87	0.80	164
2280	0.76	0.82	0.79	952
2403	0.75	0.67	0.71	952
2462	0.69	0.79	0.74	281
2522	0.92	0.89	0.91	990
2582	0.67	0.73	0.70	497
2583	0.97	0.95	0.96	1963
2585	0.76	0.79	0.77	491
2705	0.58	0.72	0.64	551
2905	0.55	0.97	0.70	174
accuracy			0.78	16684
macro avg	0.74	0.77	0.75	16684
weighted avg	0.78	0.78	0.77	16684

Comme pour les modèles précédents, nous observons une forte disparité entre les classes, allant de 0.34 pour la classe 10, jusqu'à 0.96 pour la classe 2583. Globalement, les scores du modèle Naive Bayes sont très nettement supérieurs aux deux modèles KNN et Decision Tree précédents, avec un **F1-score moyen de 0.75**.

Matrice de confusion pour alpha = 0.1

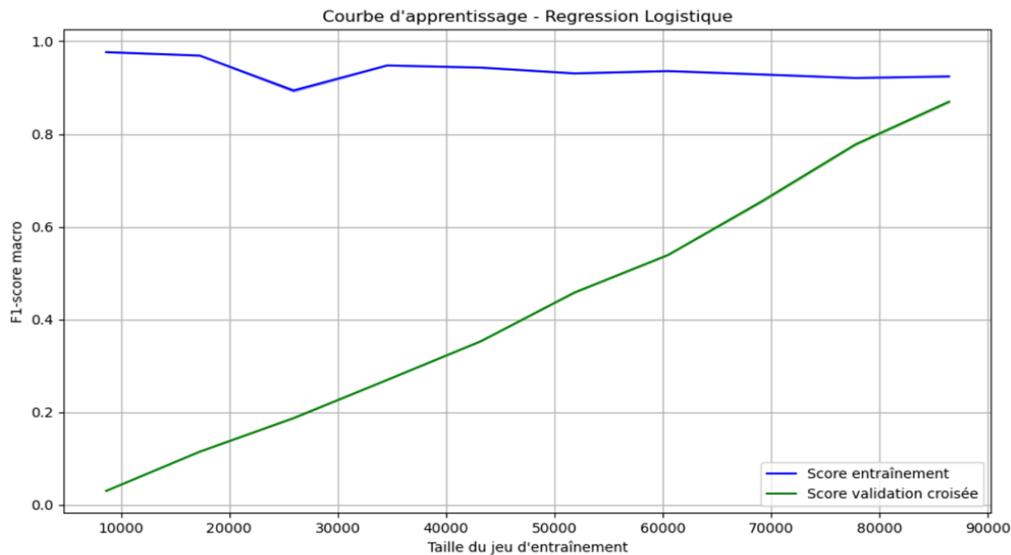


Cette augmentation du F1 score par rapport aux modèles précédents se traduit sur la matrice de confusion par une capacité nettement supérieure à traiter la classe 10 qui posait problème jusqu'à présent.

Modèles simples - Regression Logistique

Comme pour les précédents modèles, nous avons tracé la courbe d'apprentissage, et sorti le rapport de classification et la matrice de confusion.

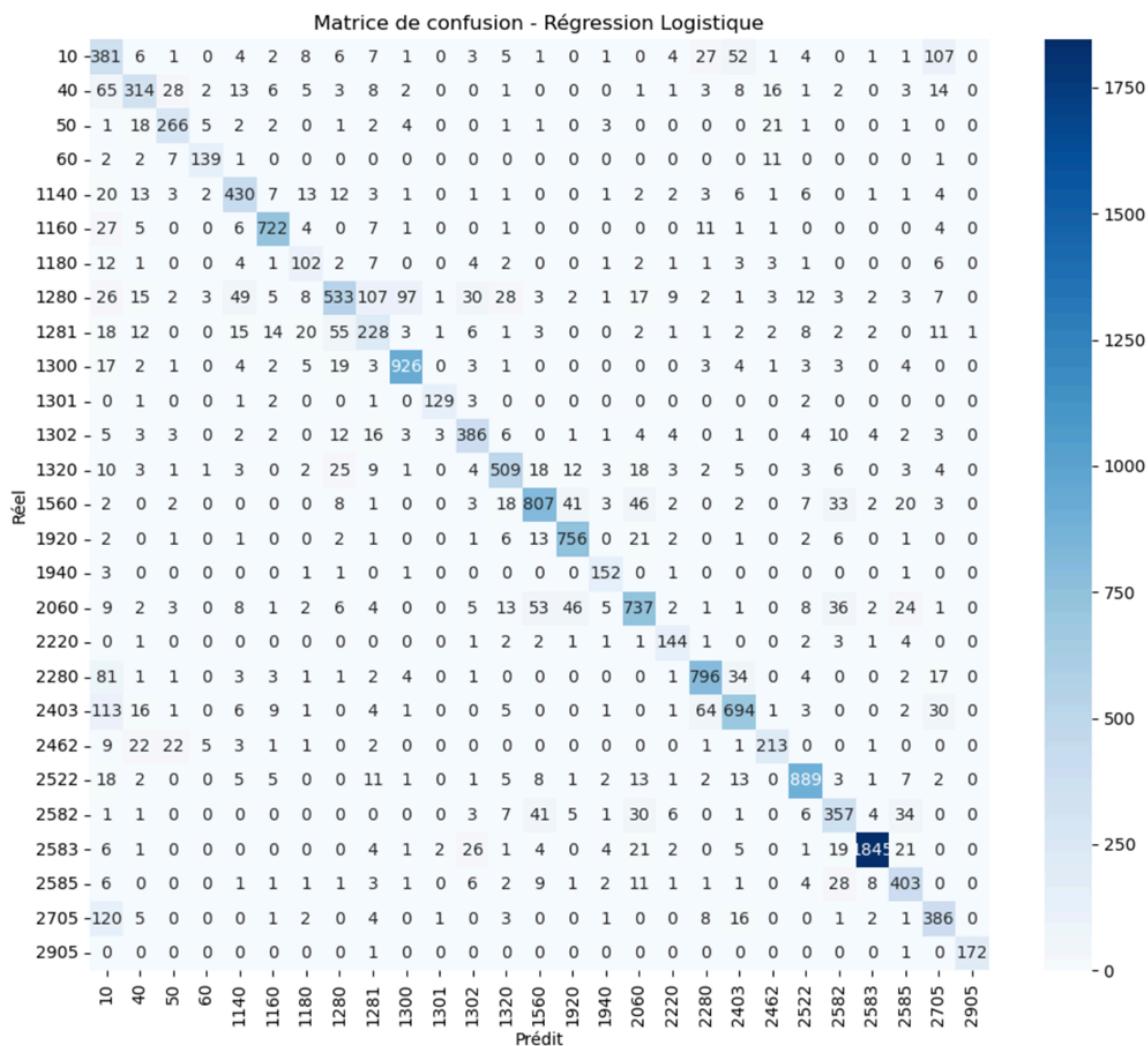
Courbe d'apprentissage



Rapport de classification

Classification Report:					
	precision	recall	f1-score	support	
10	0.40	0.61	0.48	623	
40	0.70	0.63	0.67	496	
50	0.78	0.81	0.79	329	
60	0.89	0.85	0.87	163	
1140	0.77	0.81	0.79	533	
1160	0.92	0.91	0.92	790	
1180	0.58	0.67	0.62	153	
1280	0.78	0.55	0.64	969	
1281	0.52	0.56	0.54	408	
1300	0.88	0.93	0.90	1001	
1301	0.94	0.93	0.93	139	
1302	0.79	0.81	0.80	475	
1320	0.82	0.79	0.81	645	
1560	0.84	0.81	0.82	1000	
1920	0.87	0.93	0.90	816	
1940	0.83	0.95	0.89	160	
2060	0.80	0.76	0.78	969	
2220	0.77	0.88	0.82	164	
2280	0.86	0.84	0.85	952	
2403	0.81	0.73	0.77	952	
2462	0.78	0.76	0.77	281	
2522	0.92	0.90	0.91	990	
2582	0.70	0.72	0.71	497	
2583	0.98	0.94	0.96	1963	
2585	0.75	0.82	0.78	491	
2705	0.64	0.70	0.67	551	
2905	0.99	0.99	0.99	174	
accuracy			0.80	16684	
macro avg	0.79	0.80	0.79	16684	
weighted avg	0.81	0.80	0.81	16684	

Matrice de confusion



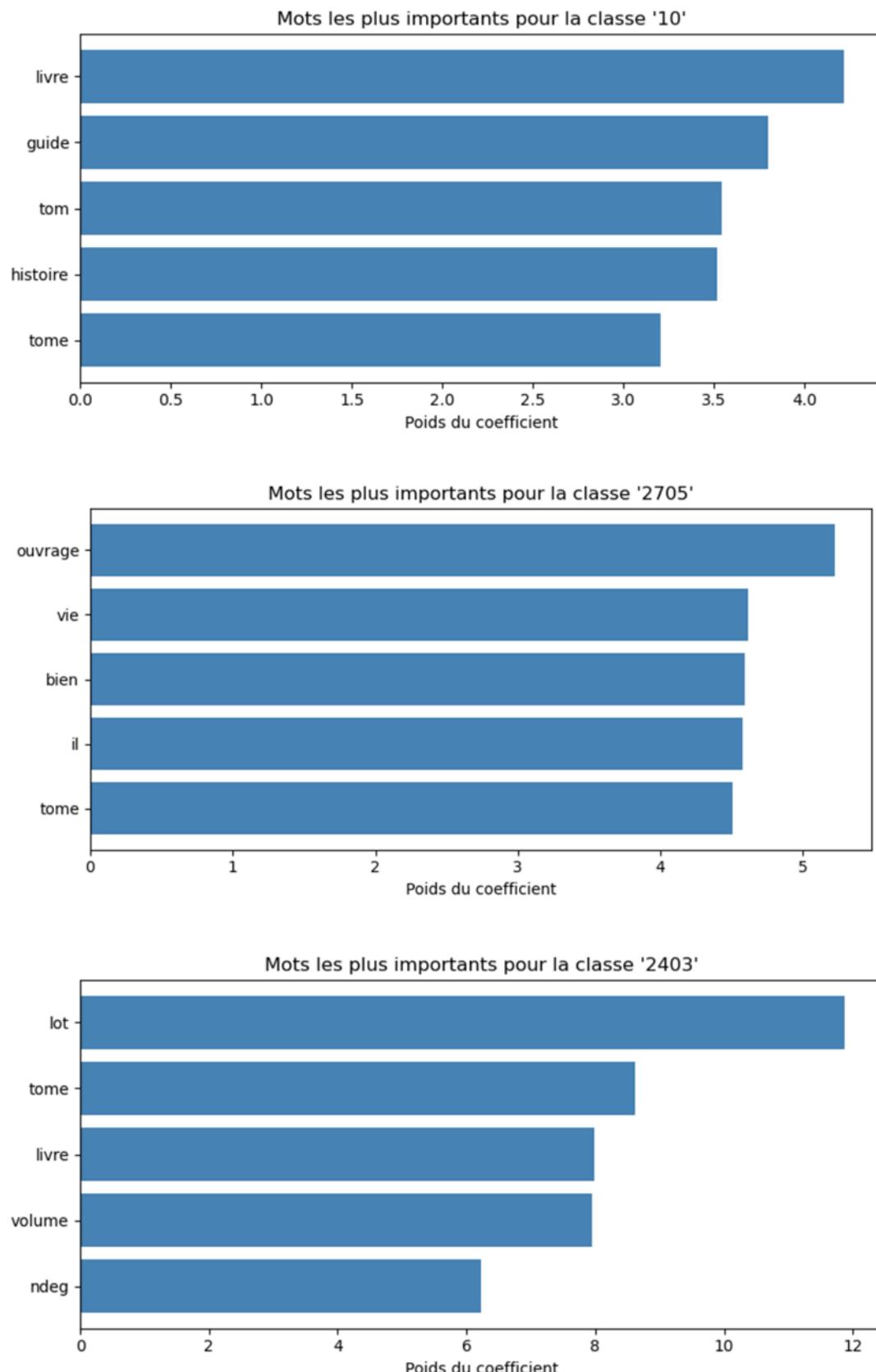
Le modèle Régression Logistique permet de gagner encore quelques points de F1-score par rapport au modèle Naïve Bayes, en passant de 0.75 à 0.79. Cependant on observe toujours une confusion avec la classe 10 :

- 113 occurrences de la classe 2403 ont été classées à tort dans la classe 10,
- et inversement, 52 occurrences de la classe 10 ont été classées en 2403,

- 120 occurrences de la classe 2705 ont été classées à tort dans la classe 10,
- et inversement, 107 occurrences de la classe 10 ont été classées en 2705.

Nous avons donc fait une étude sur l'importance des mots afin de déterminer d'où venait le problème.

Importance des mots des classes 10, 2403 et 2705



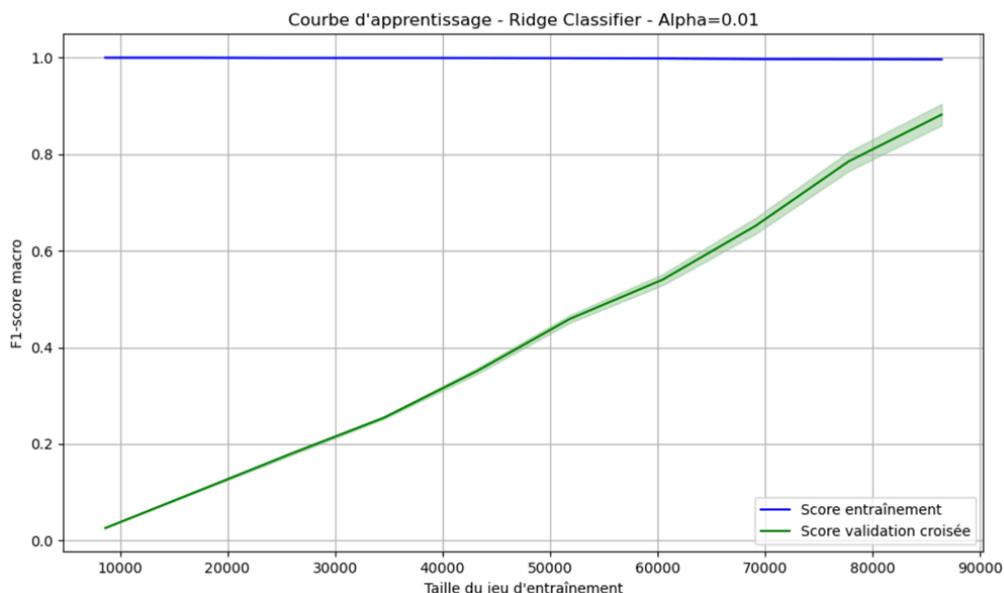
Nous observons que ces 3 classes appartiennent au même champ lexical, avec notamment des mots communs parmi les mots les plus importants de la classe, tels que : **livre** ou **tome**.

Ces 3 classes sont donc très proches. Faute de temps, nous ne sommes pas allés plus loin, mais nous avons envisagé plusieurs solutions comme réduire légèrement le poids des mots communs à ces classes, ou tout simplement fusionner ces 3 classes en une seule.

Modèles simples - Ridge Classifier

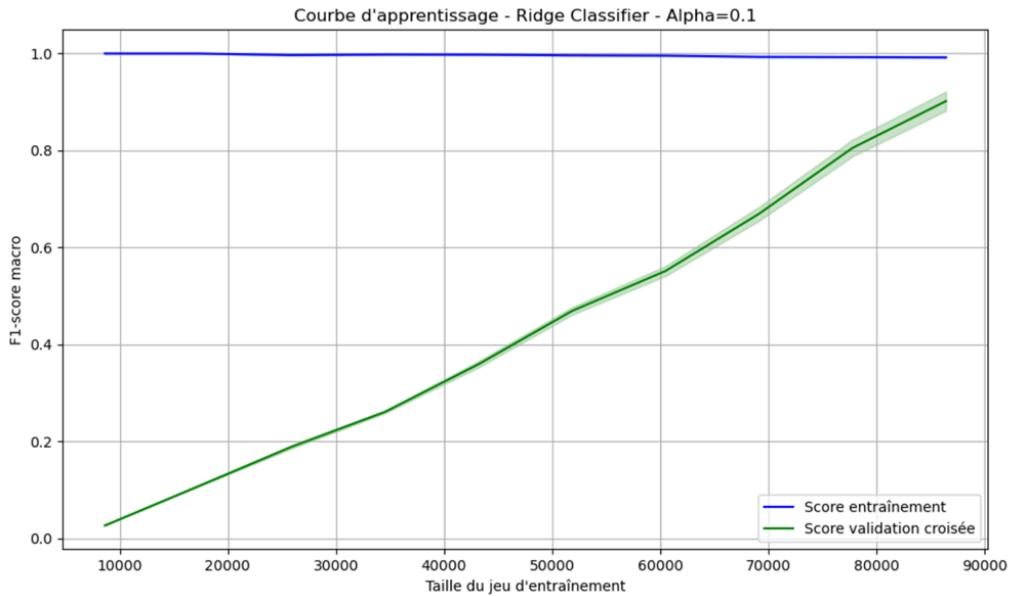
Comme pour le modèle Naive Bayes, nous allons jouer sur le paramètre alpha (0.01, 0.1, 1, 10, 100) pour trouver le meilleur modèle possible de Ridge Classifier.

Courbe d'apprentissage alpha = 0.01



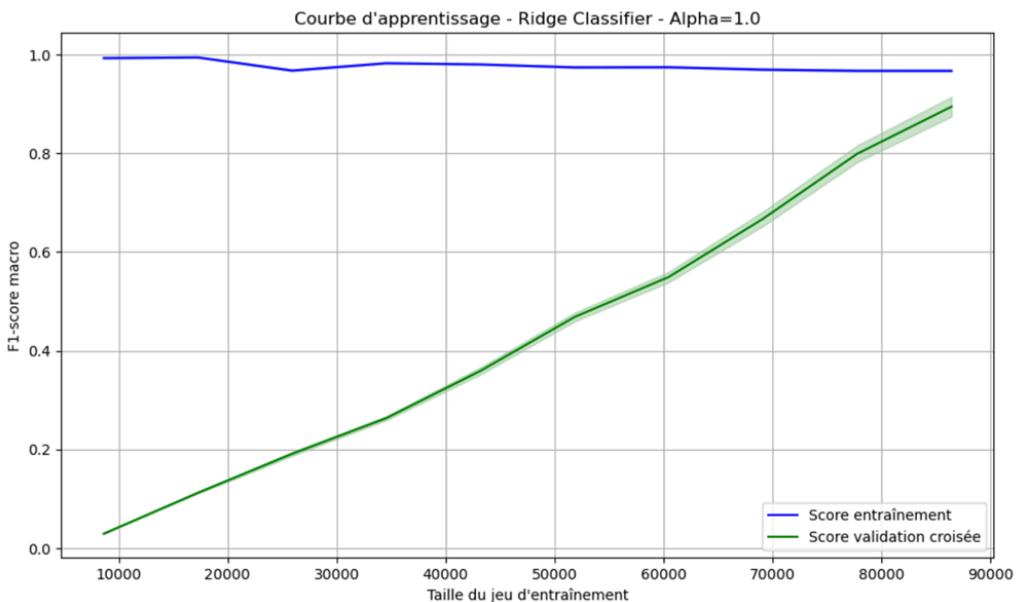
- Avec le paramètre Alpha = 0.01, la régularisation est trop forte. Le score d'entraînement ne montre aucune variation tout au long de l'apprentissage et reste strictement égal à 1. Il y a donc un overfitting fort.
- Le score validation croisée démarre bas, puis monte progressivement jusqu'à atteindre environ 0.87. Cependant, l'écart avec le score d'entraînement reste significatif. Le modèle n'arrive pas à généraliser.

Courbe d'apprentissage alpha = 0.1



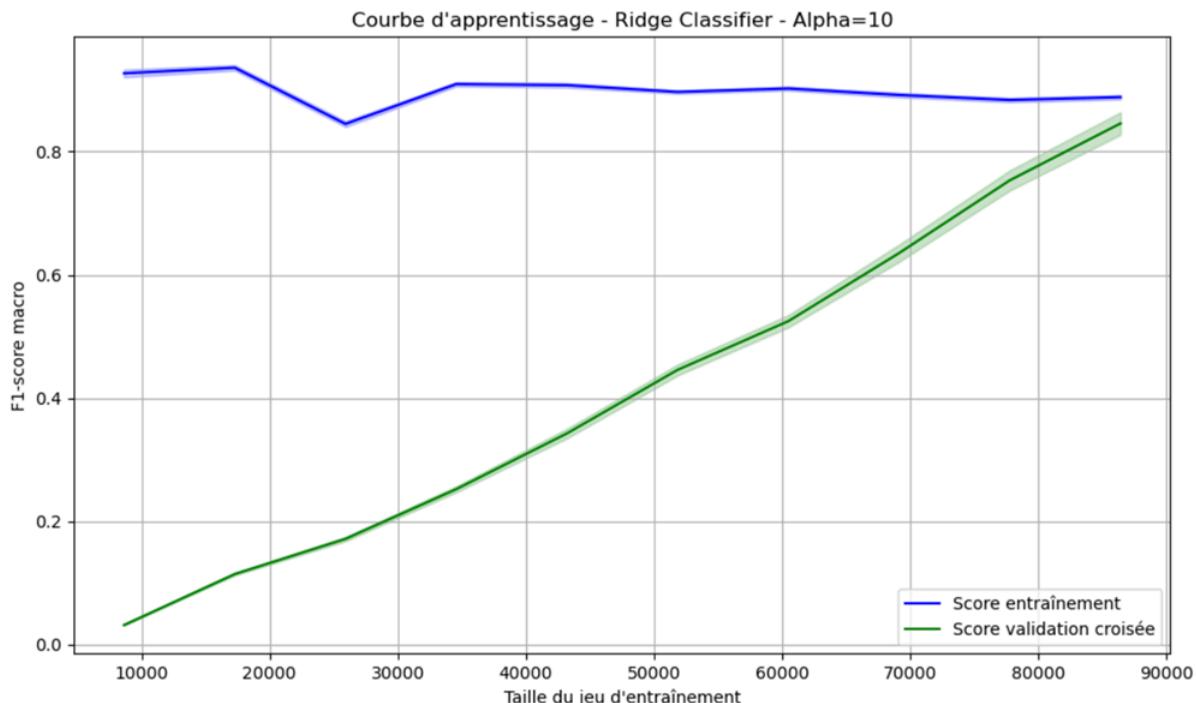
- Les observations sont sensiblement les mêmes avec le paramètre Alpha = 0.1 qu'avec le paramètre Alpha = 0.01. On constate que le score d'entraînement diminue très légèrement sur la fin pour tomber à 0.99. L'overfitting reste cependant trop prononcé.
- Le score validation croisée démarre bas, puis monte progressivement jusqu'à atteindre environ 0.88. Cependant, l'écart avec le score d'entraînement reste significatif. Le modèle n'arrive pas à généraliser.

Courbe d'apprentissage alpha = 1



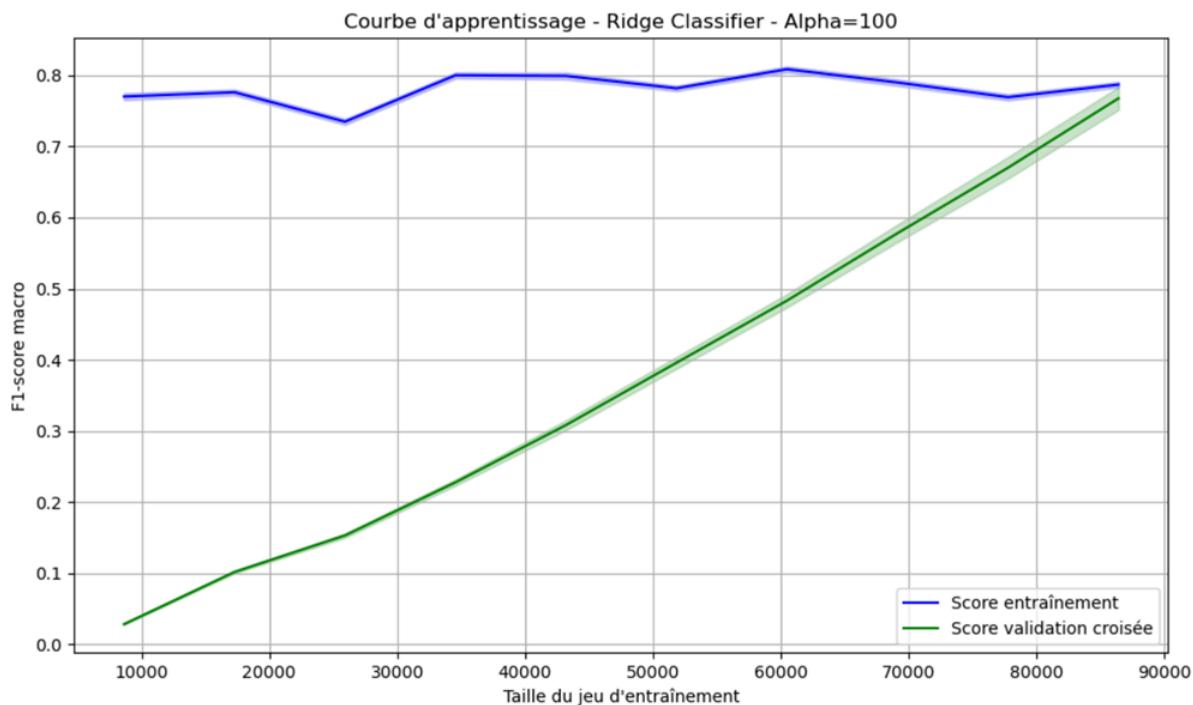
- Le score d'entraînement reste élevé, mais est plus réaliste que les 2 modèles précédent, en terminant à 0.95
- Le score de validation est également bon : 0.90
- Le modèle est meilleur que les 2 précédents. Il reste toutefois un écart non négligeable entre entraînement et validation.

Courbe d'apprentissage alpha = 10



- Le score d'entraînement diminue encore un peu par rapport au modèle précédent, mais reste très bon : 0.90
- Le score de validation se rapproche très fortement de la courbe d'apprentissage 0.85. L'écart entre le score d'apprentissage et le score de validation est encore réduit par rapport aux modèles précédents
- Il y a moins d'overfitting que pour les courbes précédentes.

Courbe d'apprentissage alpha = 100



- Le score d'apprentissage est trop faible par rapport aux tests précédents (~ 0.78). Il y a sans doute un sous-apprentissage
- Idem pour le score de validation. Bien qu'il rejoigne le score d'apprentissage, le score atteint est trop faible par rapport aux tests précédents (~0.76)

Nous en concluons que le meilleur compromis est celui avec le paramètre **alpha = 10** qui a une courbe d'entraînement élevée et écart avec le score de validation relativement faible.

Rapport de classification pour alpha = 10

Classification Report :

	precision	recall	f1-score	support
10	0.46	0.43	0.44	623
40	0.73	0.59	0.65	496
50	0.70	0.79	0.74	329
60	0.82	0.83	0.83	163
1140	0.71	0.82	0.76	533
1160	0.91	0.90	0.90	790
1180	0.50	0.68	0.57	153
1280	0.77	0.46	0.57	969
1281	0.51	0.53	0.52	408
1300	0.83	0.93	0.88	1001
1301	0.85	0.94	0.89	139
1302	0.76	0.80	0.78	475
1320	0.82	0.77	0.79	645
1560	0.83	0.79	0.81	1000
1920	0.84	0.93	0.88	816
1940	0.69	0.96	0.81	160
2060	0.79	0.73	0.76	969
2220	0.70	0.92	0.79	164
2280	0.82	0.85	0.83	952
2403	0.77	0.70	0.73	952
2462	0.68	0.80	0.73	281
2522	0.91	0.91	0.91	990
2582	0.70	0.73	0.71	497
2583	0.98	0.94	0.96	1963
2585	0.74	0.81	0.77	491
2705	0.60	0.72	0.65	551
2905	0.97	0.99	0.98	174
accuracy			0.79	16684
macro avg	0.75	0.79	0.77	16684
weighted avg	0.79	0.79	0.79	16684

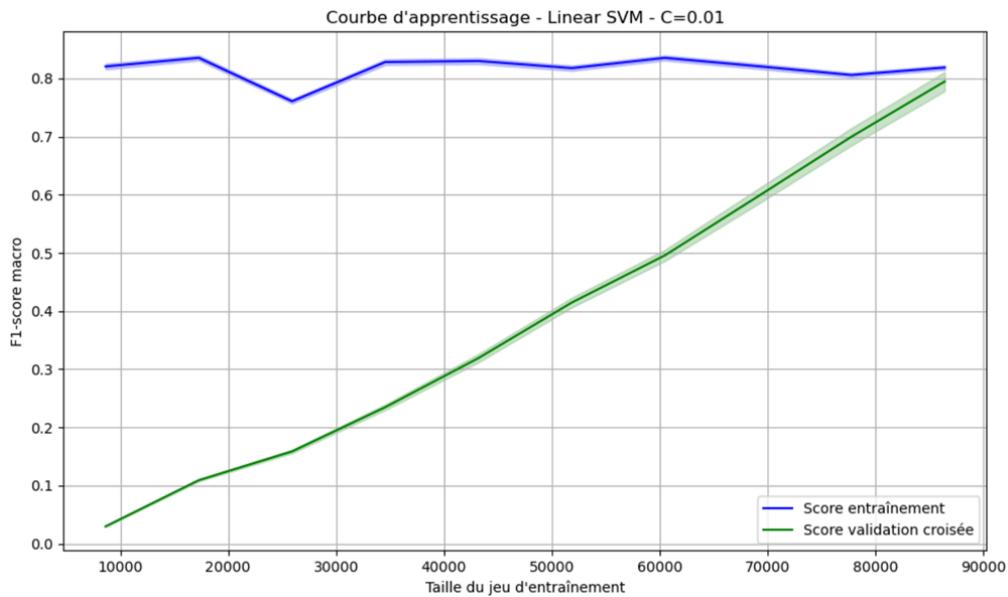
Le F1-score moyen est légèrement moins bon pour le Ridge Classifier (0.77) que pour le Linear Regression (0.79). Encore une fois, on retrouve un score très faible pour la classe 10.

Globalement, le rapport de classification et la matrice de confusion du Ridge Classifier sont très proches du Linear Regression.

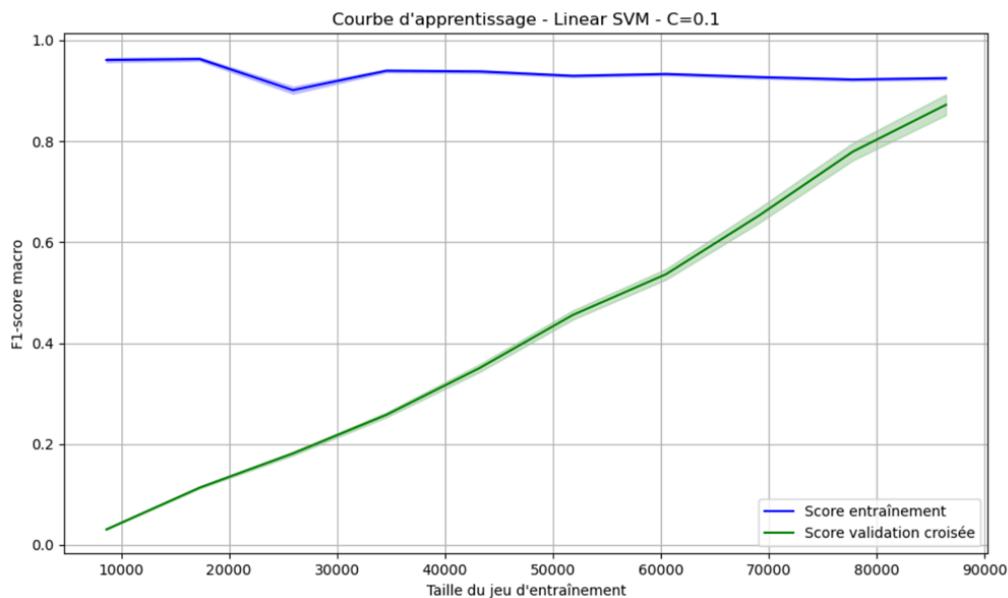
Modèles simples - Linear SVM

Nous allons jouer sur le paramètre C (0.01, 0.1, 1, 10, 100) pour trouver le meilleur modèle possible.

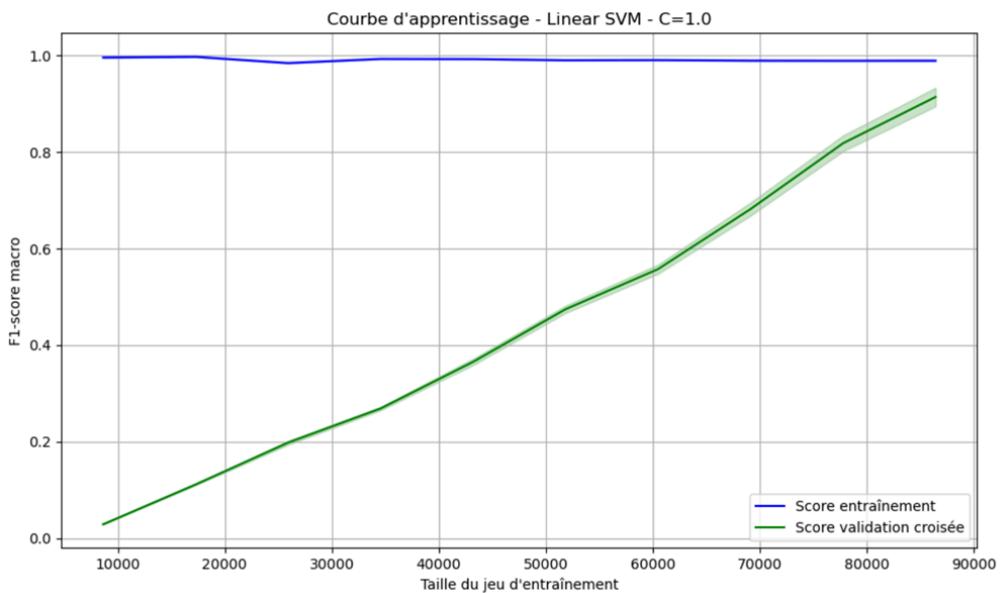
Courbe d'apprentissage pour C = 0.01



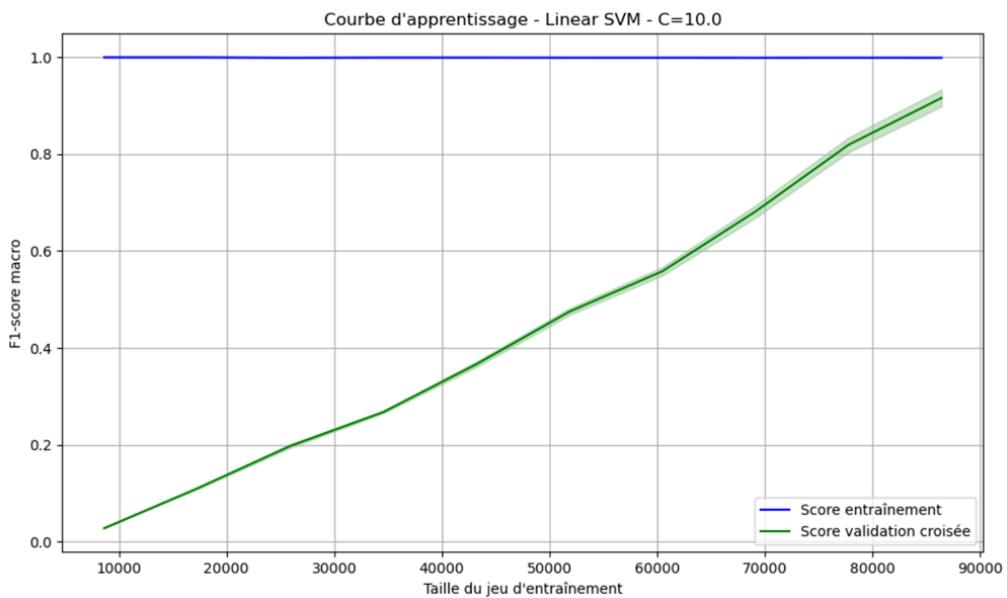
Courbe d'apprentissage pour C = 0.1



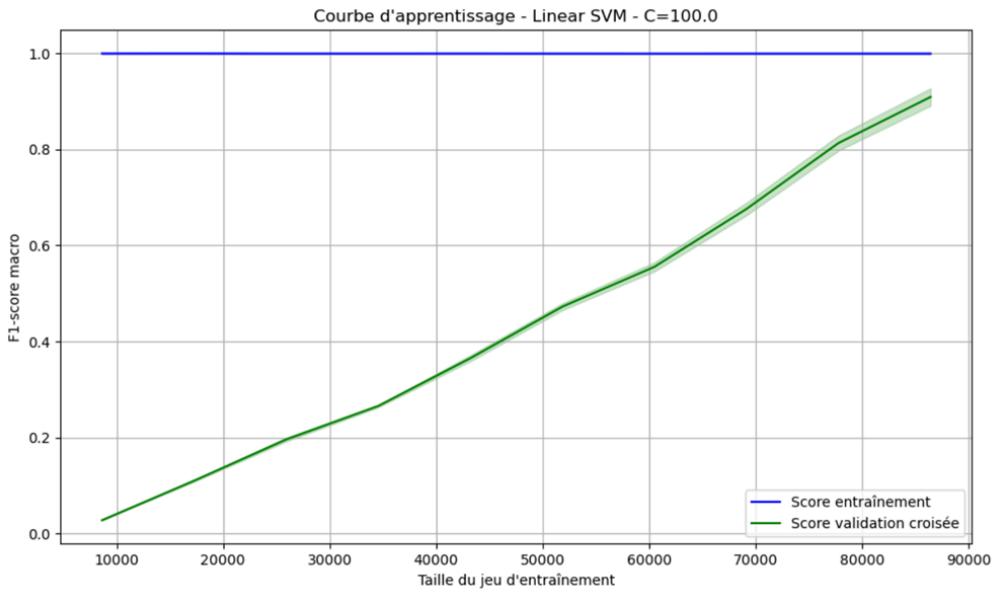
Courbe d'apprentissage pour C = 1



Courbe d'apprentissage pour C = 10



Courbe d'apprentissage pour C = 100



- Le paramètre C a l'effet inverse du paramètre alpha vu dans les modèles précédents. Plus C est élevé, et plus l'overfitting est fort avec un score d'entraînement devenant strictement égal à 1 lorsque C = 100.
- L'écart entre les scores d'entraînement et les scores de validation augmentent à mesure que C augmente.

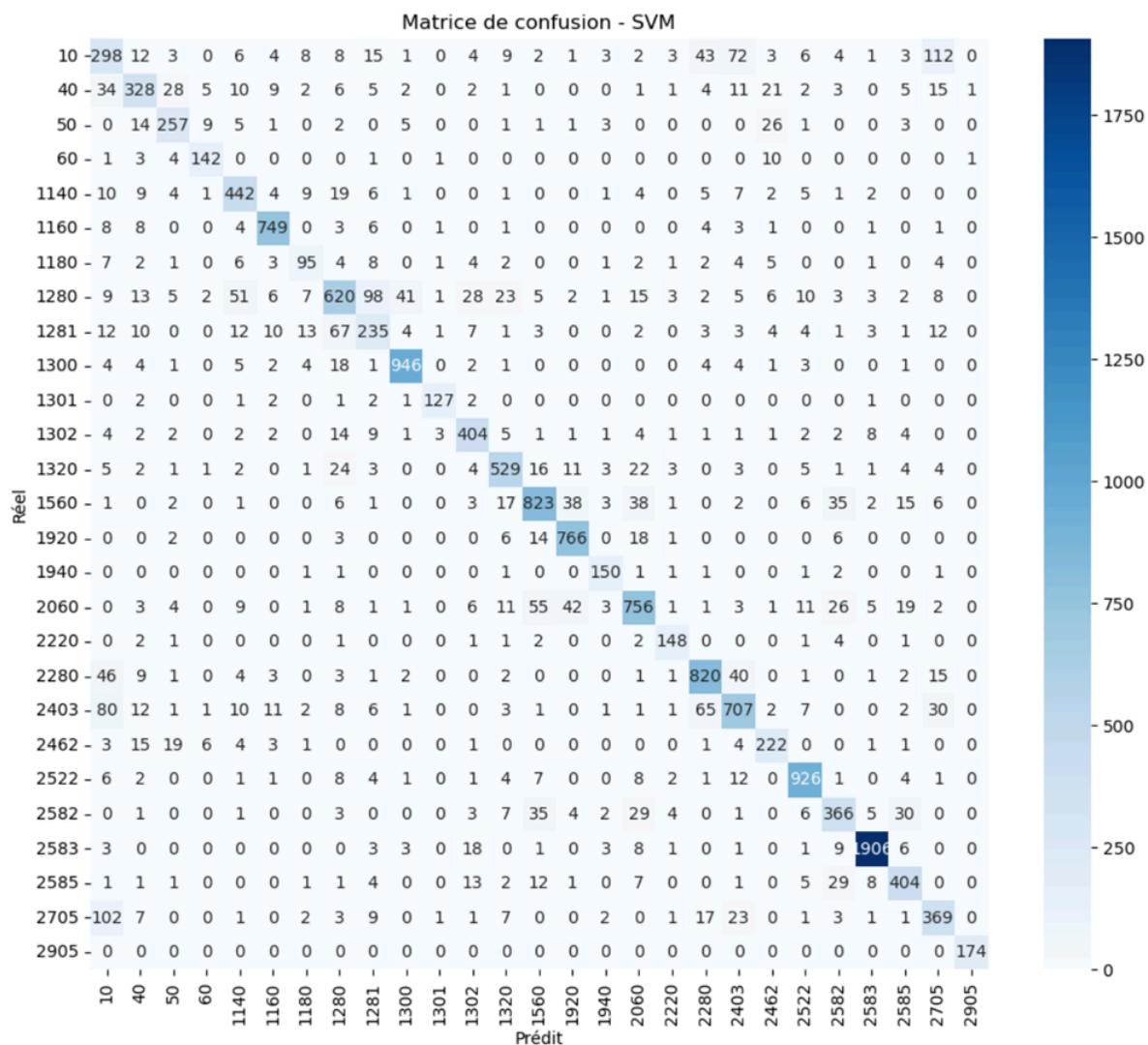
Parmi les 5 tests, le meilleur compromis semble être C = 1. En effet, le gain de performance apporté par C = 10 ou C = 100 ne semble pas significatif et apporte un fort over-fitting.

Rapport de classification du modèle Linear SVM avec C = 1

Classification Report SVM :				
	precision	recall	f1-score	support
10	0.47	0.48	0.47	623
40	0.71	0.66	0.69	496
50	0.76	0.78	0.77	329
60	0.85	0.87	0.86	163
1140	0.77	0.83	0.80	533
1160	0.92	0.95	0.94	790
1180	0.65	0.62	0.63	153
1280	0.75	0.64	0.69	969
1281	0.56	0.58	0.57	408
1300	0.94	0.95	0.94	1001
1301	0.93	0.91	0.92	139
1302	0.80	0.85	0.83	475
1320	0.83	0.82	0.83	645
1560	0.84	0.82	0.83	1000
1920	0.88	0.94	0.91	816
1940	0.85	0.94	0.89	160
2060	0.82	0.78	0.80	969
2220	0.85	0.90	0.88	164
2280	0.84	0.86	0.85	952
2403	0.78	0.74	0.76	952
2462	0.73	0.79	0.76	281
2522	0.92	0.94	0.93	990
2582	0.74	0.74	0.74	497
2583	0.98	0.97	0.97	1963
2585	0.80	0.82	0.81	491
2705	0.64	0.67	0.65	551
2905	0.99	1.00	0.99	174
accuracy			0.82	16684
macro avg	0.80	0.81	0.80	16684
weighted avg	0.82	0.82	0.82	16684

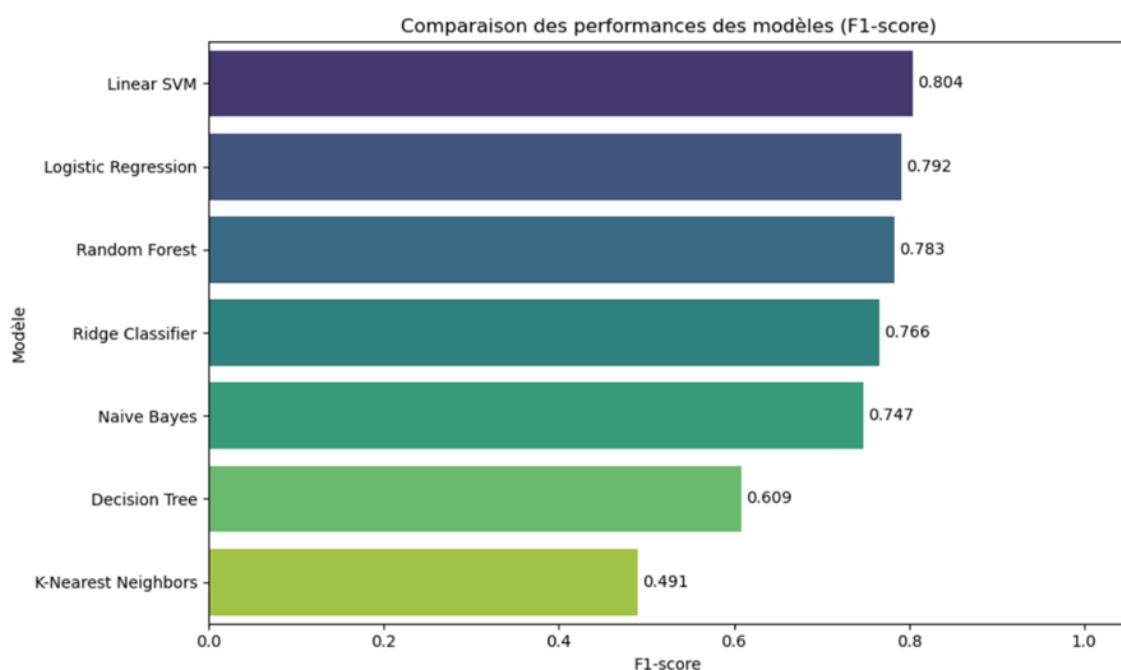
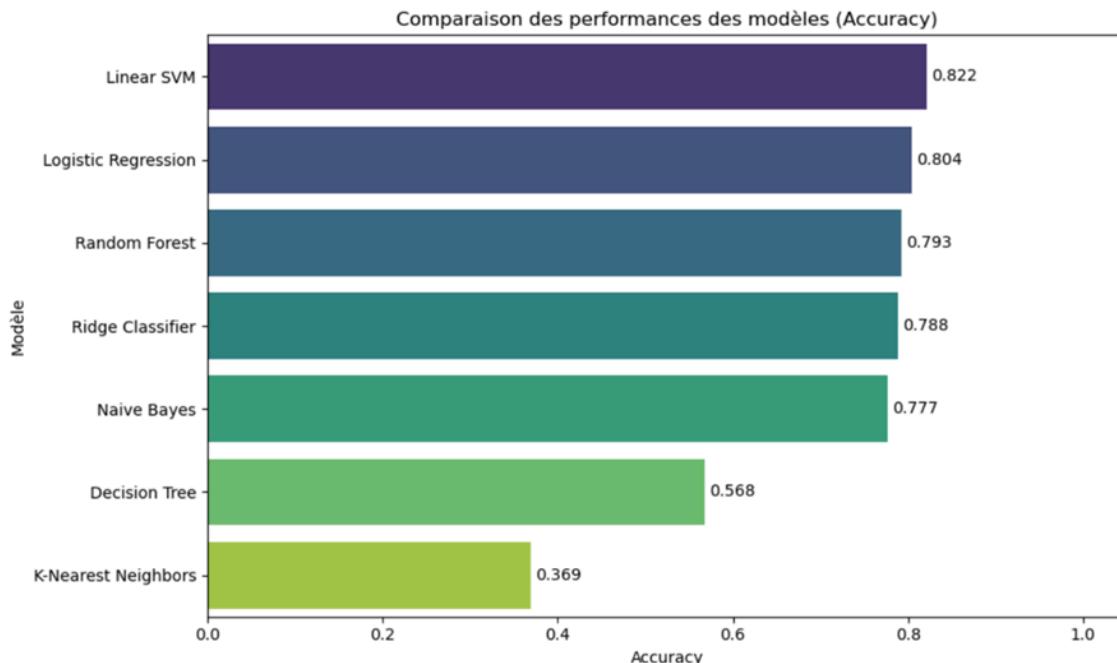
Le F1-score est le meilleur obtenu jusqu'à présent. Il est très bon avec une moyenne de 0.80. Comme pour les autres modèles, il y a toujours des difficultés avec la classe 10 pour laquelle le F1-score n'est que de 0.47.

Matrice de confusion du Linear SVM avec C = 1



La matrice de confusion rejoint les observations des modèles précédents avec une confusion entre les classes 10, 2403, 2705, et dans une moindre mesure 2280.

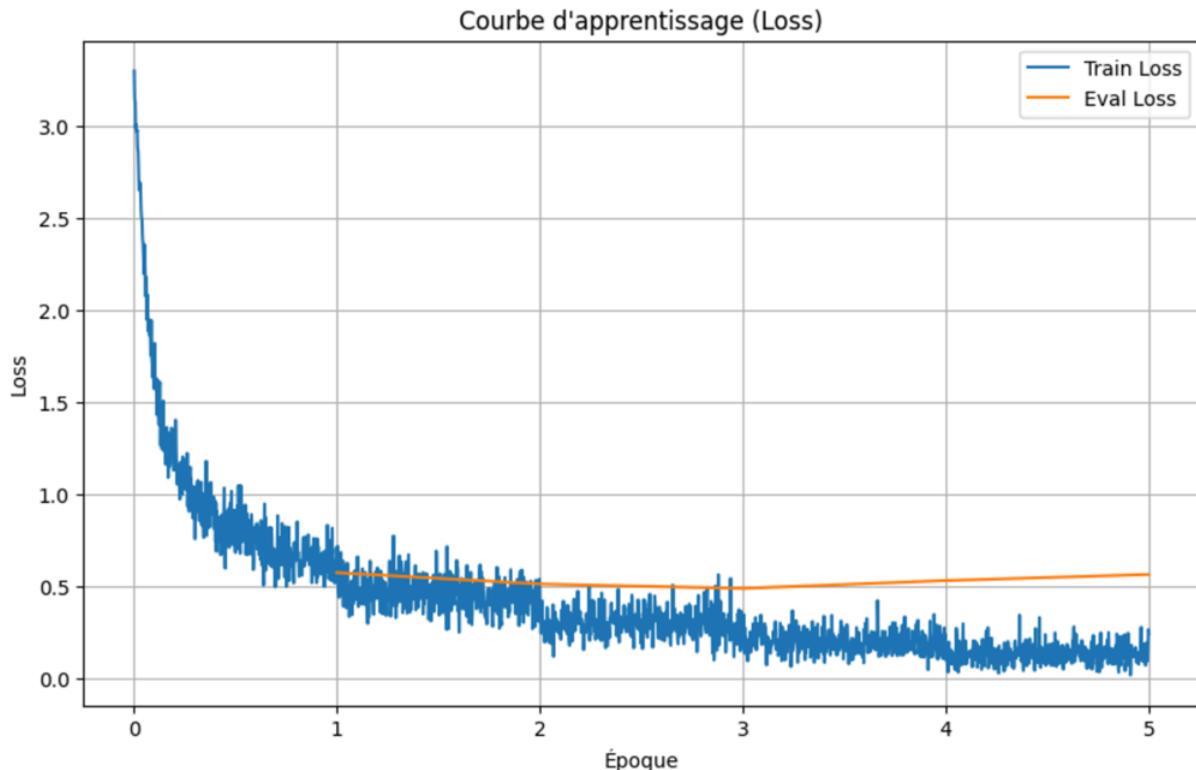
Synthèse des F1-score des différents modèles testés.



Modèles complexe pré-entraîné - BERT

En compléments des modèles classiques simples présentés précédemment, nous avons testé le modèle de BERT.

Courbe d'apprentissage (Loss) de BERT



Epoch	Training Loss	Validation Loss
1	0.670000	0.573672
2	0.539100	0.512338
3	0.277200	0.488195
4	0.212800	0.530827
5	0.243900	0.563854

La courbe de Loss de l'entraînement baisse très rapidement au début, ce qui montre que le modèle apprend bien sur les données d'entraînement.

La courbe de validation s'écarte de la courbe d'entraînement, ce qui pourrait indiquer un léger surapprentissage.

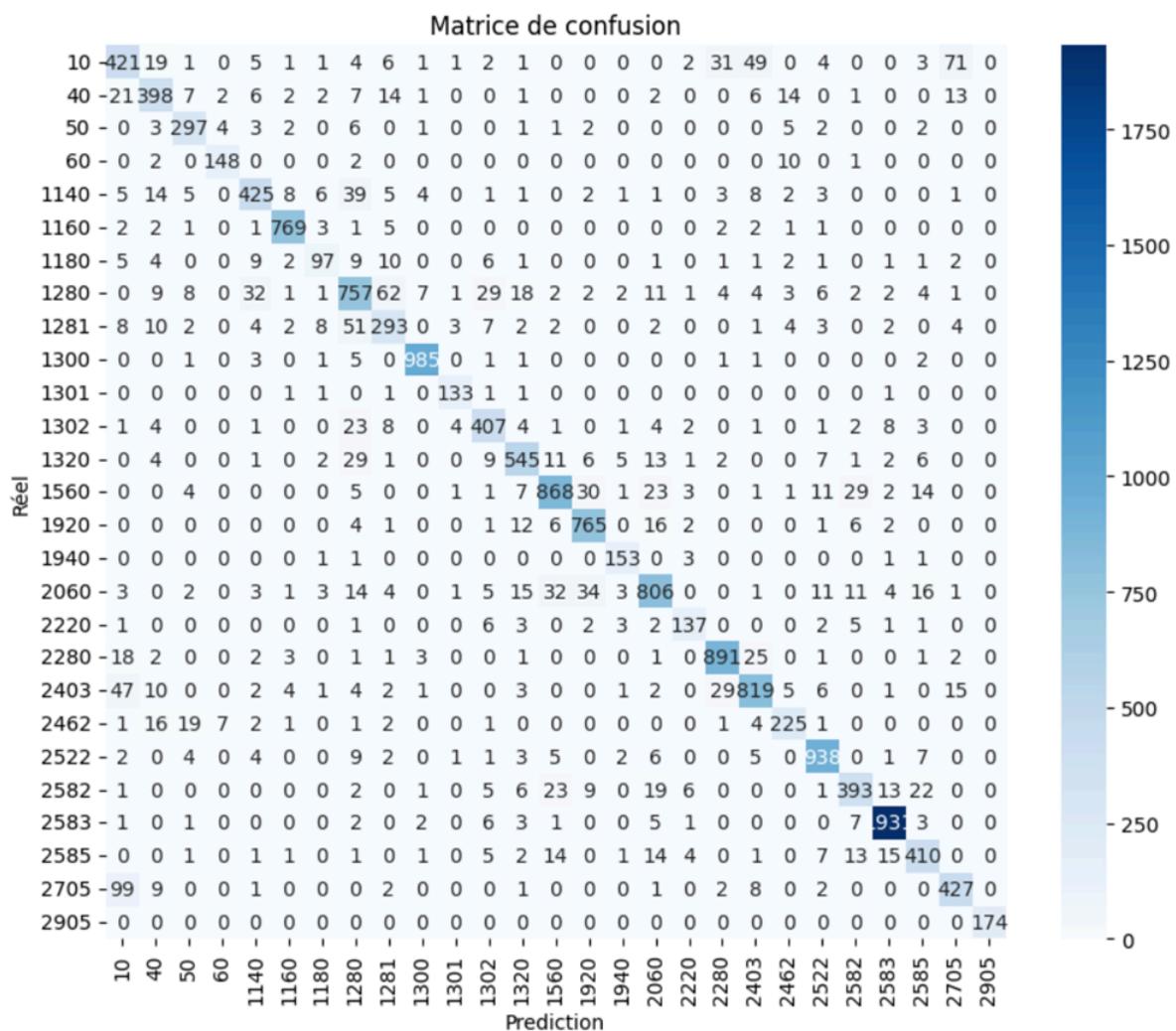
Rapport de classification du modèle de BERT

	precision	recall	f1-score	support
10	0.66	0.68	0.67	623
40	0.79	0.80	0.79	497
50	0.84	0.90	0.87	329
60	0.92	0.91	0.91	163
1140	0.84	0.80	0.82	534
1160	0.96	0.97	0.97	790
1180	0.76	0.63	0.69	153
1280	0.77	0.78	0.78	969
1281	0.70	0.72	0.71	408
1300	0.98	0.98	0.98	1001
1301	0.92	0.96	0.94	139
1302	0.82	0.86	0.84	475
1320	0.86	0.84	0.85	645
1560	0.90	0.87	0.88	1001
1920	0.90	0.94	0.92	816
1940	0.88	0.96	0.92	160
2060	0.87	0.83	0.85	970
2220	0.85	0.84	0.84	164
2280	0.92	0.94	0.93	952
2403	0.87	0.86	0.87	952
2462	0.83	0.80	0.81	281
2522	0.93	0.95	0.94	990
2582	0.83	0.78	0.81	501
2583	0.97	0.98	0.98	1963
2585	0.83	0.84	0.83	491
2705	0.80	0.77	0.78	552
2905	1.00	1.00	1.00	174
accuracy			0.88	16693
macro avg	0.86	0.86	0.86	16693
weighted avg	0.88	0.88	0.87	16693

Le modèle de BERT obtient des résultats significativement supérieurs aux modèles simples. Alors que le gain entre Random Forest, Ridge Classifier et Linear SVM est de l'ordre de 1%, le modèle de BERT nous permet de gagner encore 6% supplémentaires. Au final, nous obtenons :

- une Accuracy de 0.86, ce qui indique que 86% des prédictions sont correctes.
- un F1-score macro de 0.86, ce qui signifie que le modèle est performant de manière équilibrée pour toutes les classes, et ne favorise aucune classe en particulier.
- Le score F1 moyen pondéré est de 0.88, ce qui montre que le modèle est robuste, même en cas de déséquilibre des classes.

Matrice de confusion du modèle de BERT



La matrice de confusion met en évidence les mêmes difficultés que celles rencontrées dans les modèles simples. Encore une fois, la principale source d'erreur provient de la confusion entre les classes 10, 2403 et 2705.

Conclusion générale pour le modèle de BERT :

Le modèle BERT semble globalement bien performer, avec un bon score de précision et de rappel. Cependant, il y a des indicateurs d'un léger sur-apprentissage, et quelques classes difficiles à prédire mises en évidence par la matrice de confusion.

Modèles complexe pré-entraîné - PYTORCH

Utilisation de Pytorch afin d'avoir un modèle un peu plus complexe:

RAPPORT DÉTAILLÉ DE CLASSIFICATION AMÉLIORÉ

CONFIGURATION:

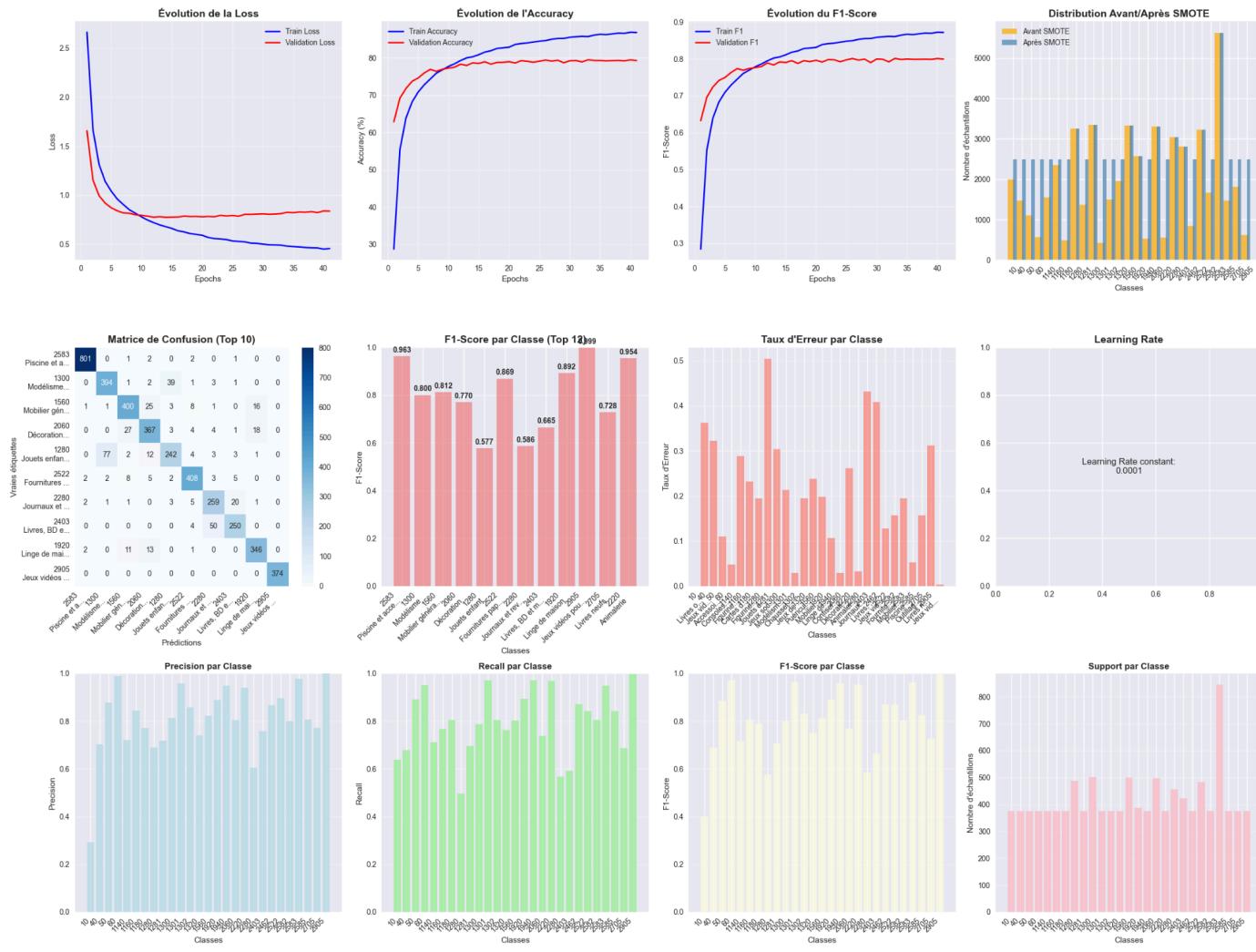
- Dataset: clean_dataset.csv
- Stratégie SMOTE: 2500 (adaptative)
- Epochs réalisées: 41
- Batch size: 32
- Learning rate initial: 0.0001
- Early stopping patience: 15

RÉSULTATS FINAUX:

- Test Accuracy: 79.66%
- Test F1-Score: 0.8022
- Meilleure Val Accuracy: 79.43%
- Meilleure Val F1-Score: 0.8009
- Écart Train-Test Accuracy: 7.15%

AMÉLIORATION VS VERSION PRÉCÉDENTE:

- Réduction de l'overfitting: écart train-test = 7.2%
- F1-Score pondéré: 0.8022
- Classes équilibrées après SMOTE: 10 tailles différentes
- Accuracy: 79.66%
- F1-Score: 0.8022
- Réduction significative de l'overfitting
- Meilleur équilibrage des classes



PERFORMANCE DU MODÈLE:

Le modèle montre une excellente convergence avec une précision finale d'environ 80% et un F1-score autour de 0,8. La courbe de perte diminue de manière stable, ce qui indique une bonne généralisation sans surapprentissage significatif

ANALYSE DES MÉTRIQUES:

Les courbes de perte, de précision et de F1-score montrent une progression cohérente. L'écart entre les performances d'entraînement et de validation reste modéré, ce qui suggère un bon équilibre entre apprentissage et généralisation.

EQUILIBRAGE DES CLASSES AVEC SMOTE:

L'application de SMOTE a permis d'obtenir une distribution plus homogène des classes, ce qui est bénéfique pour l'entraînement du modèle.

Conclusion générale pour le modèle de PYTORCH :

Au final on obtient une bonne performance avec le réseau neuronal (accuracy de 79,66% et F1 de 80,22%) mais le modèle BERT reste meilleur.

Combinaison des modèles

Nous avons essayé de combiner les performances des plusieurs modèles travaillant chacun sur un modalité. Pour ce faire nous avons utilisé une approche dite de “**late fusion**” qui consiste à combiner les résultats de plusieurs modèles ou modalités **après** que chaque modèle a effectué une prédiction indépendamment. Notre modèle “fusionné” utilisait la moyenne des probabilités pour prédire le résultat final. Nous avons des tests avec un paramètre *alpha* permettant de favoriser un modèle ou l'autre mais il s'avère que globalement la performance était souvent celle du modèle le plus performant, à savoir le modèle d'analyse du texte. Cette approche n'a donc pas permis d'obtenir un vrai gain. (cf notebook)

Bilan du projet

Ce projet nous a permis de mettre en œuvre de manière concrète l'ensemble des compétences en data science (NLP, traitement d'image, deep learning, interprétabilité).

Nous avons appris à **concevoir, entraîner et optimiser des modèles de classification automatique multimodaux**, combinant analyse d'images et de textes, dans un contexte e-commerce complexe. Nous avons particulièrement travaillé sur :

- **Le nettoyage et la préparation linguistique des données textuelles multilingues**, avec traduction automatique, vectorisation TF-IDF et lemmatisation.
- **Le traitement des images** : normalisation, suppression des doublons, centrage des objets et standardisation.
- **Le rééquilibrage des classes** via SMOTE et techniques d'augmentation
- **L'évaluation comparative** de nombreux modèles de machine learning et deep learning (SVM, Naive Bayes, BERT, EfficientNet).
- **Fine-tuning des modèles CNN** (notamment EfficientNetB0) : dégel partiel de couches pour une spécialisation sur le domaine Rakuten.
- **Optimisation des modèles texte** : transition de modèles simples (SVM, Ridge) vers **BERT**, puis vers un **réseau neuronal PyTorch** plus performant
- **La fusion des approches** image et texte a été abordée, bien que non finalisée.

Nous avons rencontré quelques difficultés pour lesquels nous avons trouvé des solutions:

- Complexité du nettoyage textuel (langues multiples, bruit HTML)
- Fort déséquilibre des classes nécessitant une stratégie de rééchantillonnage adaptée
- Ambiguïté de certaines classes taxonomiques (notamment la classe "jeux société enfants")

Nos objectifs sont globalement atteints puisque nous avons réussi à:

- Améliorer la qualité des données (nettoyage, traduction, équilibrage ...)
- Exploiter efficacement les données textuelles et visuelles (en mettant en place des pipelines complet des deux côtés)
- Classer automatiquement les produits Rakuten avec une bonne performance (macro-F1 score > 80%)

Nous avons également identifié des pistes d'amélioration possible pour notre modèle:

- **Utilisation de Vision Transformers (ViT)** pour la partie image
- **Revue de la taxonomie produit** : subdiviser les classes trop hétérogènes comme "Jeux société enfants"
- **Techniques de data augmentation plus ciblées** par classe
- Creuser les possibilités de fusion des approches ou l'usage d'un modèle multimodal (ex : CLIP, ViLT)

Annexes

Description des fichiers de code.

```
└── LICENSE
└── README.md      <- The top-level README for developers using this project.
└── data
    ├── interim      <- Dataset partiellement traités (avant traduction automatique)
    ├── processed     <- Dataset finaux pour la modélisation
    └── raw           <- Le dump de données original et immuable
└── models          <- Trained and serialized models, model predictions, or model summaries
└── notebooks       <- Notebook de travail
└── references      <- Data dictionaries, manuals, links, and all other explanatory materials.
└── reports         <- Rapport de projet et artefacts
    └── figures        <- Graphiques et statistiques générés et destinés à être
        ├── analyse_exploratoire_images   <- artefacts de l'analyse exploratoire des images
        ├── benchmark_images_models_results <- artefacts du benchmark des modèles pour les images
        ├── benchmark_text_models_results    <- artefacts du benchmark des modèles pour le text
        └── models_training                <- Graphs d'apprentissage du modèle
└── requirements.txt  <- fichier de requirements pour reproduire l'environnement du projet
└── src              <- Code source du projet
    ├── features        <- Scripts to turn raw data into features for modeling
        ├── image_features.py            <- Méthodes utilisées pour le preprocessing des images
        ├── images_dataset_resampling.py <- Méthodes utilisées pour le resampling des images
        ├── images_preprocessing.py      <- Pipeline de preprocessing des images
        └── translation_script.py       <- Script de traduction des descriptions produits
    ├── models           <- Scripts to train models and then use trained models to make predictions
        ├── text_model_pytorch
            ├── data_utils.py           <- Contient les fonctions pour charger et prétraiter le
            ├── main.py                 <- Le script principal qui utilise les fonctions des au
            ├── evaluate.py            <- Les fonctions pour évaluer les modèles et visualiser
            ├── models.py               <- Définitions des modèles
            └── train.py                <- Fonctions pour entraîner les modèles
        ├── images_models
            ├── dataset_utils.py        <- Méthodes utiles pour manipuler le dataset d'image
            ├── dataviz_utils.py       <- Méthodes utiles pour générer des graphiques sur l'en
            └── EfficientNetBo_model_train.py <- Script d'entraînement du modèle
        ├── benchmark_images_models.py <- Script de benchmark des modèles pour les images
        └── predict_model.py
    └── visualization
        ├── analyse_exploratoire_images.py <- Script de création de visualisation pour l'analyse e
        └── grad-cam.py                <- visualisation grad-cam
```