DataScientest

Challenge Rakuten

Rapport d'exploration, de data visualisation de pre-processing et de modélisation

CHELOUAH Slimane
LOLO MVOUMBI Simplice
MORMICHE Nicolas
ZIDI Riadh

Table des matières

1 - Introduction du projet

1.1 - Contexte

Ce projet s'inscrit dans le cadre des challenges organisés par l'équipe Data de l'Ecole Normale Supérieure de Paris. Ce challenge, proposé par le Rakuten Institute of Technology, porte sur le thème de la classification multimodale (texte et image) de produits e-commerce à grande échelle. L'objectif est de prédire le code type produit tel que défini dans le catalogue de Rakuten France.

D'un point de vue économique, le Multimodal Learning est une technique utile pour les e-commerces car il permet de catégoriser automatiquement des produits en fonction des images et des informations fournies par les commerçants.

Référence du Challenge : https://challengedata.ens.fr/challenges/35

1.2 - Périmètre

On se limite donc aux données fournies par Rakuten avec un total pour les produits de 84916 descriptions, 84916 images et 27 catégories uniques associées pour lesquelles nous devrons réaliser nos prédictions. Pour se faire, nous prendrons comme base les modèles de référence fournis par Rakuten :

- Pour les données texte, un modèle CNN simplifié
- Pour les données d'image, un modèle Residual Networks (ResNet)

1.3 - Objectifs

Les objectifs du challenge, formulé par Rakuten, sont d'étudier la classification multimodale dans leur contexte et d'améliorer les résultats de références obtenus par leur équipe.

Rakuten ayant déjà réalisé une classification multimodale de référence à l'aide de divers modèles, le but est de proposer une alternative en testant différents modèles afin d'obtenir un meilleur score F1 pondéré que celui proposé par Rakuten actuellement :

- Classification des textes avec CNN : 0.8113
- Classification des images avec ResNet50 : 0,5534

A noter que pour valider les résultats, Rakuten demande aux candidats du challenge de fournir un fichier y_test.csv (pour un jeu X_test.csv), dans le même format que le fichier y_train.csv reçu, où chaque identifiant produit est associé à un code type :

1	Unnamed: 0	Un identifiant entier pour le produit. Cet identifiant permet d'associer le produit à son code type de produit correspondant.
2	prdtypecode	Code type de produit (catégorie)

Ne disposant pas des valeurs ‘y réelles’ correspondant au jeu de test X_test.csv, la génération et la fourniture des prédictions du fichier y_test.csv sortira du cadre de notre projet.

1.4 - Notions importantes

Le Multimodal Learning est une notion centrale du projet. Il existe plusieurs méthodes pour fusionner les informations textuelles et visuelles :

- Les réseaux de neurones multimodaux
  (modèle de fusion CNN / RNN / TNN)

- L’apprentissage par transfert
  (transférer des connaissances apprises d’une modalité à une autre)

- Les techniques de co-apprentissage ou d’apprentissage adversarial
  (représentation des données multimodales)

- Les réseaux de neurones auto-encodeurs
  (reconstruire les données d’entrée en passant par une représentation latente)

Concernant la classification des données multimodales (texte et image), nous aurons plusieurs choix de modèle à traiter afin de vérifier le score F1 pondéré de chacun :

- Classification des textes (ex : BERT ou CNN)
- Classification des images (ex : CNN, RNN ou TNN)

Pour former et entraîner un modèle de Multimodal Learning, on peut utiliser des techniques de Deep Learning tel que les réseaux de neurones où l’on utilise des encodeurs :

- Former un modèle multimodal à double encodeur (ex : BERT et ResNet-50)
- Entraîner un modèle multimodal à encodeur commun (ex : ALBEF)

2 - Exploration des données

Rakuten met à disposition un jeu de données et fournit les images et les fichiers CSV pour X_train, y_train et X_test (sans les données y_test correspondantes). Après avoir analysé le dataset (annexe n°1 et 2), nous avons décidé de fusionner X_train et y_train pour partir d'un jeu de données complet, de 84916 lignes, sur lequel réaliser l'exploration des données.

Nous avons dans un premier temps confirmé qu'il n'y avait pas de doublon, mais nous avons constaté plus de 35% de valeurs nulles dans la colonne 'description'. Aussi, pour éviter de supprimer autant de lignes, nous avons décidé de fusionner les colonnes 'description' et 'designation' afin de créer une nouvelle colonne 'descriptif'.

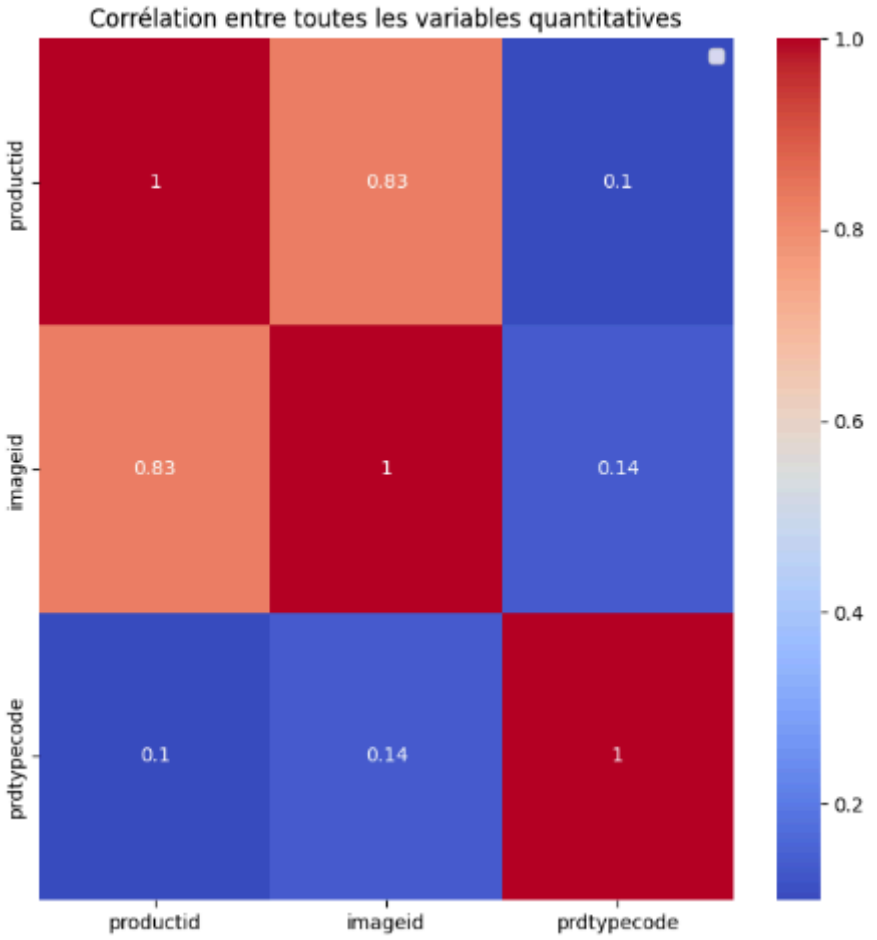
Concernant l'étude de la variable cible 'prdtypecode', nous avons identifié un ensemble de 27 catégories de produit, représentées par un code type unique (ex : 10, 1140, 2060, etc..). Cette variable étant définie par la description du produit et par son image, nous avons également étudié le dossier des images où l'on retrouve les mêmes catégories, avec des images au format : 'image_imageid_product_productid'.

Le détail du tableau est disponible ici : exploration des données

Nom	Type	Taux de NA	Distribution
Unnamed: 0	int64	0.00 %	Valeur unique
productid	int64	0.00 %	Valeur unique
imageid	int64	0.00 %	Valeur unique
designation	object : string non-nullable	0.00 %	Catégorielle
description	object : string non-nullable	35.09 %	Catégorielle
prdtypecode	int64	0.00 %	Catégorielle

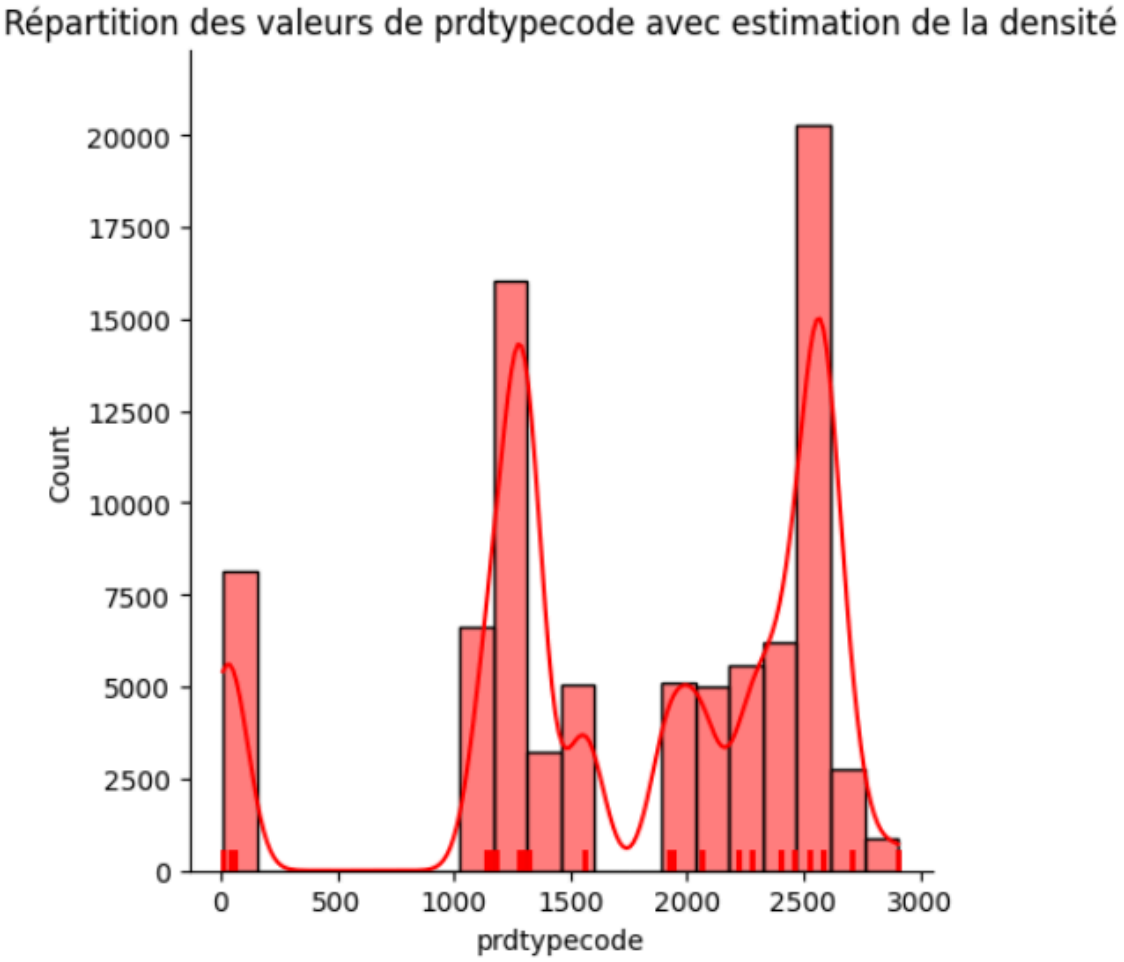
3 - Visualisation des données

3.1 Heatmap : corrélation entre les variables quantitatives



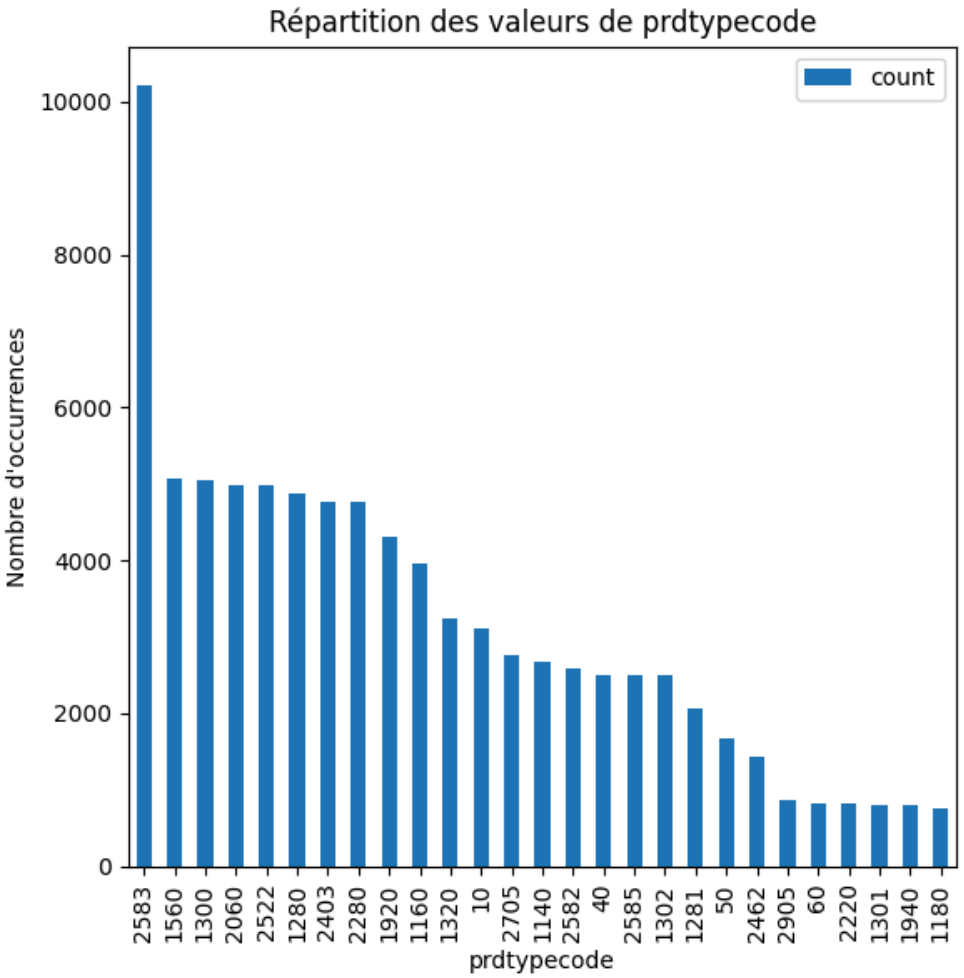
Conclusion : on ne retrouve aucune corrélation intéressante entre les variables quantitatives mis à part entre productid et imageid qui ne semble pas être significatif pour le projet.

3.2 Histogramme et estimation de la densité : prdtypecode



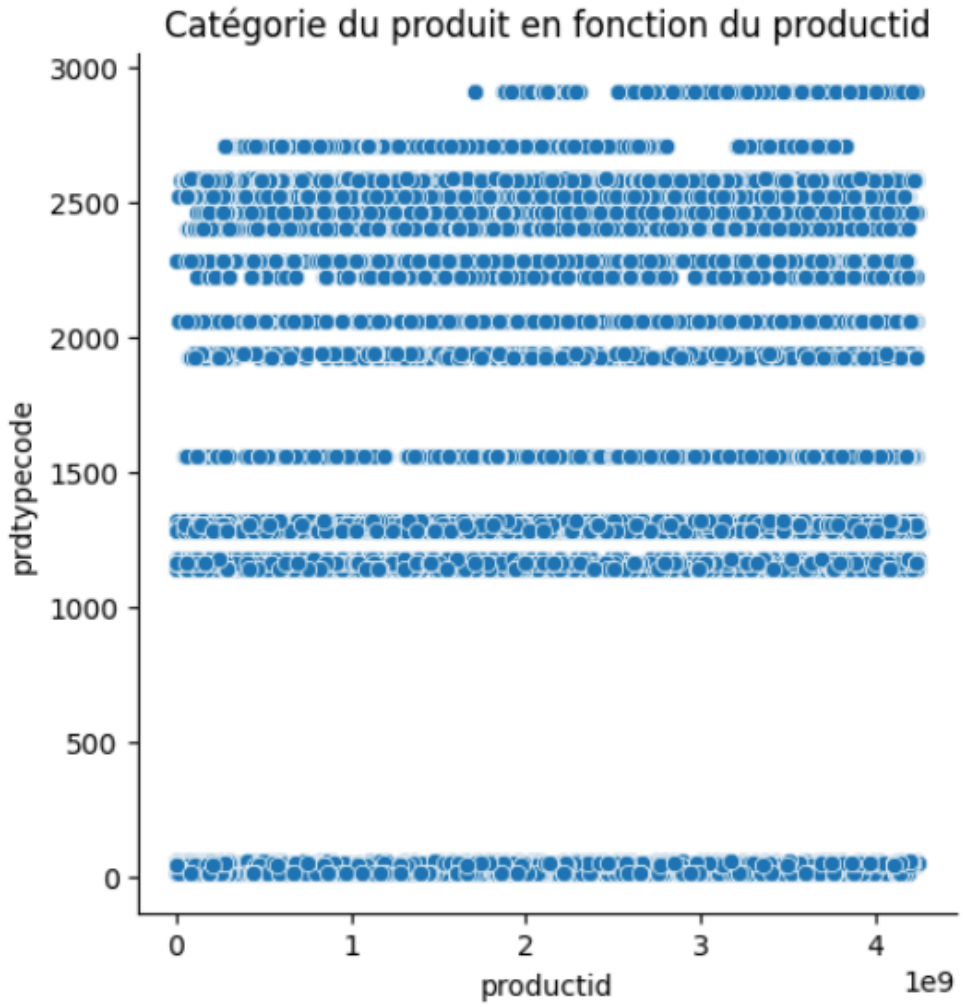
Conclusion : on constate que les valeurs de codes type produit se répartissent sur 3 plages de valeurs principales (ex : entre 0 et 50, entre 1000 et 1500 et entre 2000 et 2900).

3.3 Histogramme : prdtypecode



Conclusion : étant donnée la représentation des 27 catégories de produits, ordonnées par ordre décroissant, on constate que la catégorie 2583 se détache fortement en terme de sur-représentation et que les catégories 2905, 60, 2220, 1301, 1940 et 1180 se détachent fortement en termes de sous-représentation. On peut donc en déduire qu'il s'agit d'un problème de classification multiclasses sur des données déséquilibrées.

3.4 Nuage de point : productid et prdtypecode



Conclusion : on remarque que les codes type se répartissent par plage discrètes de valeurs, d'où la répartition en lignes. Dans la majorité des cas, pour chaque code type, les codes produits s'étendent sur l'ensemble de la plage des productid.

4 - Compréhension et manipulation des données

4.1 - Cadre et pertinence

Le jeu de données fourni par Rakuten étant divisé en deux ensembles d'entraînement et de test, on constate qu'il nous manque y_test. Ces données n'étant pas disponibles, nous avons utilisé l'ensemble d'entraînement comme base de travail. Etant donné un déséquilibre constaté sur les données (cf § 3.3 Histogramme : prdtypecode), nous avons aussi investigué sur le ré-échantillonage des données. On retrouve ainsi la variable cible 'prdtypecode' et les variables explicatives de notre futur modèle de Multimodal Learning :

- pour le texte : descriptif
- pour les images : image_imageid_product_productid.jpg

La particularité du jeu de données réside dans le Multimodal Learning où nous devrons mettre en relation texte et image pour déterminer le prdtypecode correspondant. De plus, nous avons procédé à un traitement naturel du langage NLP pour la partie texte.

4.2 - Pre-processing et feature engineering

- **Gestion des valeurs nulles**

Etant donnée la variable 'description' qui contient 35% de valeurs nulles et pour conserver toutes les informations textuelles sur les produits, nous avons fusionné les colonnes 'designation' et 'description' dans une nouvelle colonne 'descriptif'.

- **Standardisation et dichotomisation**

Le jeu de données étant vraisemblablement pré-traité côté Rakuten, il ne sera pas utile de le standardiser car nous n'avons qu'une variable explicative de type chaine de caractère. La dichotomisation sera inutile car la seule variable explicative est une chaîne de caractère.

- **Enrichissement des données**

Aucune nouvelle donnée n'est apportée pour enrichir notre dataset car cela sort du cadre du Challenge Rakuten qui nous demande de s'intéresser uniquement aux données fournies.

- **Détection des langues**

Nous avons analysé les différentes langues présentes dans la colonnes ‘descriptif’ et on constate qu’on retrouve un grand nombre de langues (français et anglais prédominant) :

Français	62518	Portugais	344	Lituanien	81
Anglais	12456	Suédois	274	Slovaque	79
Italien	2476	Indonésien	247	Albanais	76
Catalan	1084	Estonien	213	Swahili	73
Roumain	886	Tagalog	199	Letton	49
Espagnol	625	Croate	190	Tchèque	39
Néerlandais	512	Finnois	182	Turc	33
Norvégien	488	Slovène	177	Hongrois	31
Afrikaans	440	Gallois	157	Erreur_langue	9
Danois	409	Polonais	107	Vietnamien	4
Allemand	375	Somali	83		

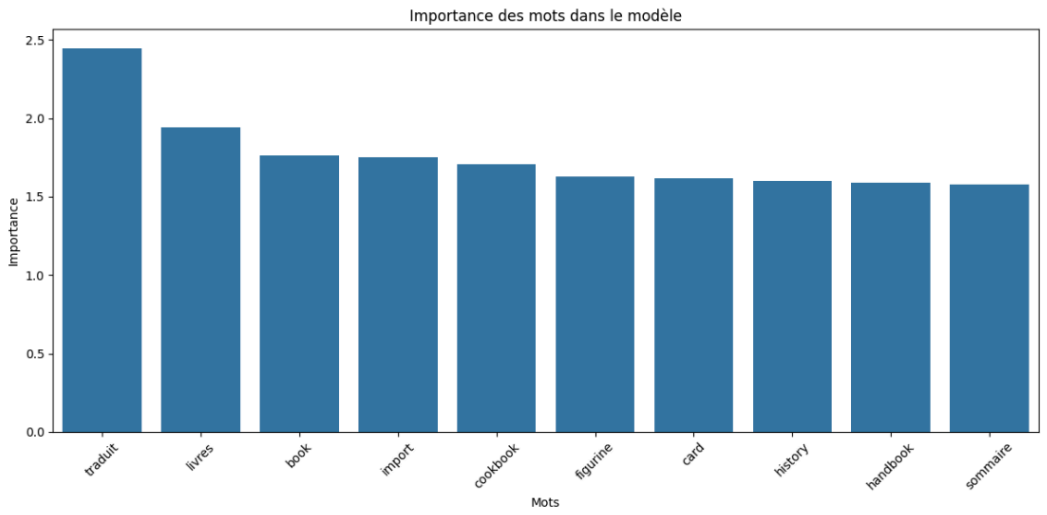
Lors du nettoyage de la colonne, les résultats obtenus nous ont permis de constater la présence de 9 modalités (ex : erreur_langue) pour lesquelles le commerçant n’avait pas renseigné de description ou désignation avec des mots suffisamment explicite.

- **Traitement naturel du langage**

Nous avons appliqué un nettoyage sur le texte en utilisant les méthodes NLP qui nous ont permis de réduire les données afin de les rendre exploitable pour les futurs modèles :

- Supprimer les espaces excessifs, les balises HTML, les nombres et caractères spéciaux, les mots vides et les mots de moins de 4 lettres (ex : les, tes, ces, etc …)
- Lemmatisation et tokenisation du texte
- Vectorisation pour transformer le texte en valeur numérique

On peut voir sur l’histogramme suivant les 10 mots les plus présents avec leur importance :



- **Traitement des images**

Concernant les images fournies par Rakuten, elles sont séparées en deux ensembles d'entraînement dans les dossiers /images/image_train et /images/image_test où on constate que les images sont toutes au format jpg, de taille 500x500 et de mode RGB.

Pour avoir les mêmes informations côté texte que côté image, nous n'avons utilisé que les images fournies dans le dossier /images/image_train, que nous avons séparé en jeu de données d'entraînement avec 67932 images et de test avec 16984 images, pour lesquelles nous avons créé un nouveau dataframe, avec pour chaque image :

- filepath : chemin d'accès à l'image
- prdcodetype : code de la catégorie associée

Etant donné le nombre important d'images et pour réduire le temps d'exécution des futurs modèles, nous avons réduit la taille des images à 125x125 en les passant en nuances de gris et en les enregistrant dans un dossier /images/image_train_preprocessed.

Nous avons ensuite réalisé une augmentation des données sur l'ensemble d'entraînement afin d'éviter au maximum le sur-apprentissage : redimensionnement, zoom, rotation, retournement horizontal, décalage en hauteur, décalage en largeur.

4.3 - Visualisations et statistiques

L'étape du traitement du langage nous a permis de passer d'une liste de mots de 222906 à 137099 après nettoyage. Aussi, pour mieux comprendre la variable cible, nous avons réalisé une étude sur les mots les plus fréquents présents par catégorie.

Nous avons ensuite pu réaliser un ensemble de 27 nuages de mots concernant les différentes catégories afin de se faire une idée plus claire sur le contenu de chacune. Ainsi, on visualise mieux à quoi correspond chaque catégorie, par exemple :

- 1920 : semble correspondre à la literie (oreiller, couverture, canapé, lit, maison etc)
- 1940 : semble correspondre à l'épicerie (chocolat, sucre, café, bio, fruit, etc)
- 2462 : semble correspondre aux jeux vidéos (jeux, xbox, playstation, manette, etc)

5 - Réalisation du projet

5.1 - Classification du problème

Le challenge Rakuten fait référence à un problème de classification multiclasse où le but est de catégoriser des produits en fonction de leur image et de leur description avec :

- pour le texte : utilisation d'algorithmes de machine learning
- pour les images : utilisation d'algorithmes de réseaux de neurones

La métrique de performance utilisée par Rakuten est le f1-score, utilisée sur des données déséquilibrées (cf § 3.3 Histogramme : prdtypecode), que nous utiliserons pour comparer nos résultats à ceux du client. Nous étudierons également l'accuracy, qui évalue la capacité d'un modèle de classification à prédire la bonne catégorie d'un produit e-commerce.

5.2 - Choix du modèle et optimisation

5.2.1 - Classification des produits (texte)

A) Orientations générales

- **Stratégies de classification (OneVSRest et OneVsOne)**

Nous avons écarté les stratégies de classification, souvent utilisées dans les problèmes multi-classe, car elles ne sont pas adaptées aux grands ensembles de données.

- **Optimisation des hyperparamètres (GridSearchCV et BayesSearchCV)**

Nous avons utilisé les grilles de recherche pour tester et trouver les meilleurs hyperparamètres. Ainsi, nous avons pu améliorer le score sur plusieurs de nos modèles.

Les approches GridSearchCV et BayesSearchCV se distinguent de la manière suivante :

- GridSearchCV effectue une recherche exhaustive en testant toutes les combinaisons possibles de paramètres dans les plages spécifiées.

- BayesSearchCV utilise l'optimisation bayésienne, qui apprend au fur et à mesure des essais pour cibler les zones prometteuses de l'espace des hyperparamètres.

En théorie, BayesSearchCV devrait surpasser GridSearchCV en termes de performances mais cela ne se vérifie pas toujours dans la pratique. Une étude comparative est nécessaire.

B) Ré-échantillonnage des données

Nous avons utilisé différentes méthodes de ré-échantillonnage des données afin d'optimiser les scores de performance et d'étudier leurs impacts sur nos modèles pour la partie texte.

Sous-échantillonnage : les méthodes RandomUnderSampler et EditedNearestNeighbour donnent lieu à des dégradations significatives des scores avec -10 à -20% environ par rapport aux tests sans ré-échantillonnage.

Sur-échantillonnage : les méthodes SMOTE et ADASYN malgré de bons résultats n'ont pas permis d'améliorer les scores qui restent inférieur à ceux obtenus sans ré-échantillonnage.

En conclusion, les méthodes de ré-échantillonnages ne s'avèrent pas efficaces dans notre contexte et ne seront donc pas retenues pour l'entraînement des différents modèles.

Exemple du ré-échantillonnage : DecisionTree (sans optimisation des hyperparamètres) :

	Ré-échantillonnage	Accuracy	F1-score
DecisionTree	Aucun	0.7265	0.7286
	Undersampling-RandomUnderSampler	0.6327	0.6437
	Undersampling-EditedNearestNeighbour	0.5611	0.5743
	Oversampling-SMOTE	0.7116	0.7135
	Oversampling-ADASYN	0.7153	0.7169

C) Présentation des modèles

Nous avons essayé plusieurs modèles en fonction de notre problématique de classification multiclasse afin de comparer les scores de performance (accuracy et f1-score) de chacun :

- **Régression logistique (annexe n°3 et 4)**

La régression logistique est une technique de classification employée pour prédire des classes binaires. Les vecteurs TF-IDF s'avèrent souvent très performants, avec un score de 0.7983 (f1-score = 0.7983) et un score moindre de 0.7062 (f1-score = 0.6935) avec optimisation des hyperparamètres.

- **Naive Bayes : MultinomialNB (annexe n°5 et 6)**

Ce modèle est l'une des variantes de Bayes utilisées dans la classification de textes où les vecteurs TF-IDF sont connus pour bien fonctionner. Les résultats n'étaient pas très bons avec un score de 0.6691 (f1-score = 0.6422) mais l'optimisation des hyperparamètres nous a permis d'obtenir un score amélioré de 0.7658 (f1-score = 0.7603).

- **Naive Bayes : ComplementNB (annexe n°7 et 8)**

Ce modèle est une adaptation du MultinomialNB, plus stable et plus performant dans les tâches de classification de texte. Le score est nettement supérieur à celui du MultinomialNB avec 0.7404 (f1-score = 0.7275). En revanche, la grille de recherche n'a pas beaucoup augmenté le score passant à 0.7416 (f1-score = 0.7289).

- **SVM (annexe n°9 et 10)**

Le modèle SVM semble plutôt performant sans optimisation des hyperparamètres avec un score de 0.8168 (f1-score = 0.8190). Cependant, étant donné le temps excessivement long de d'exécution suite au grand nombre de données du dataset, il est difficile de l'améliorer.

- **LinearSVM (annexe n°11 et 12)**

Le modèle LinearSVM est plutôt performant avec les paramètres par défaut et un score de 0.8181 (f1-score = 0.8162). Étant donné le temps d'exécution, nous avons opté pour une optimisation bayésienne des hyperparamètres avec un score de 0.8193 (f1-score = 0.8171).

- **SGDClassifier (annexe n°13 et 14)**

Le modèle SGDClassifier est connu pour des problèmes d'apprentissage à grande échelle et plus particulièrement dans la classification de textes et le traitement du langage naturel. On constate un score relativement correct de 0.7786 (f1-score = 0.7696) et un score de 0.8088 (f1-score = 0.8052) après optimisation bayésienne des hyperparamètres.

- **KNeighborsClassifier (annexe n°15 et 16)**

La méthode des K-plus proches voisins est une méthode d'apprentissage supervisée extrêmement simple où l'algorithme calcule les voisins les plus proches de l'entrée et déduit la classe qui constitue la majorité de ces voisins afin d'en déduire la classe du nouveau point. On obtient un score approximatif de 0.4403 (f1-score = 0.5219). Nous n'avons pas pu optimiser les hyperparamètres car le temps d'execution était trop long.

- **DecisionTreeClassifier (annexe n°17 et 18)**

Le modèle par arbres de décision permet de traiter des problèmes de classification multi-classes et est robuste aux données bruitées, ce qui est souvent le cas dans les problèmes de classification de texte avec des données volumineuses. Ce modèle est efficace en termes de temps de calcul et de mémoire, avec des temps de traitement relativement faibles. On obtient ici des scores relativement corrects et peu améliorés par l'utilisation d'une grille GridSearchCV avec un score de 0.7275 (f1-score = 0.7294).

D) Tableau récapitulatif

Résultats des modèles avec et sans optimisation :

	Sans paramètres		Avec paramètres optimisés	
	Accuracy	**F1-score**	**Accuracy**	**F1-score**
LogisticRegression	0.7983	0.7983	0.7062	0.6935
	C=0.1			
MultinomialNB	0.6691	0.6422	0.7658	0.7603
	alpha=0.1, fit_prior=False, force_alpha=True			
ComplementNB	0.7404	0.7275	0.7416	0.7289
	alpha=0.5, fit_prior=True, force_alpha=True			
SVM	0.8168	0.8190	/	/
LinearSVM	0.8181	0.8162	0.8193	0.8171
	C=0.7399651076649312, max_iter=10000			
SGDClassifier	0.7786	0.7696	0.8088	0.8052
	alpha=0.0001, loss='modified_huber', max_iter=2000, penalty='l2'			
KNeighborsClassifier	0.4403	0.5219	/	/
DecisionTreeClassifier	0.7265	0.7286	0.7275	0.7294
	criterion='gini'			

Après avoir retenu les modèles les plus performants sur notre jeu de données (LinearSVM et SGDClassifier avec optimisation des hyperparamètres), nous avons également confirmé que le ré-échantillonnage des données dégradait aussi les scores pour ces modèles.

Nous avons également utilisé deux algorithmes d'optimisation pour améliorer les scores de performance de nos modèles. Malheureusement, les résultats obtenus ont été équivalent aux modèles de base, voir inférieur dans certains cas :

		Accuracy	F1-score
LinearSVM	Classique	0.8193	0.8171
	AdaBoostClassifier	0.6372	0.6703
	BaggingClassifier	0.8181	0.8162
SGDClassifier	Classique	0.8088	0.8052
	AdaBoostClassifier	0.6265	0.6621
	BaggingClassifier	0.7600	0.7505

Nous avons donc tenté de trouver les meilleurs hyperparamètres à l'aide d'une grille de recherche avec GridSearchCV mais le temps d'exécution étant excessivement long, nous n'avons obtenu aucun résultat. Voici un bref récapitulatif des hyperparamètres utilisés :

- AdaBoostClassifier
    - `'learning_rate': [0.001, 0.01, 0.1],`
    - `'n_estimators': [100, 200, 500, 1000],`
- BaggingClassifier
    - `'n_estimators': [100, 200, 500, 1000],`
    - `'max_samples': [0.5, 1.0],`
    - `'max_features': [0.5, 1.0],`

E) Modèle retenu

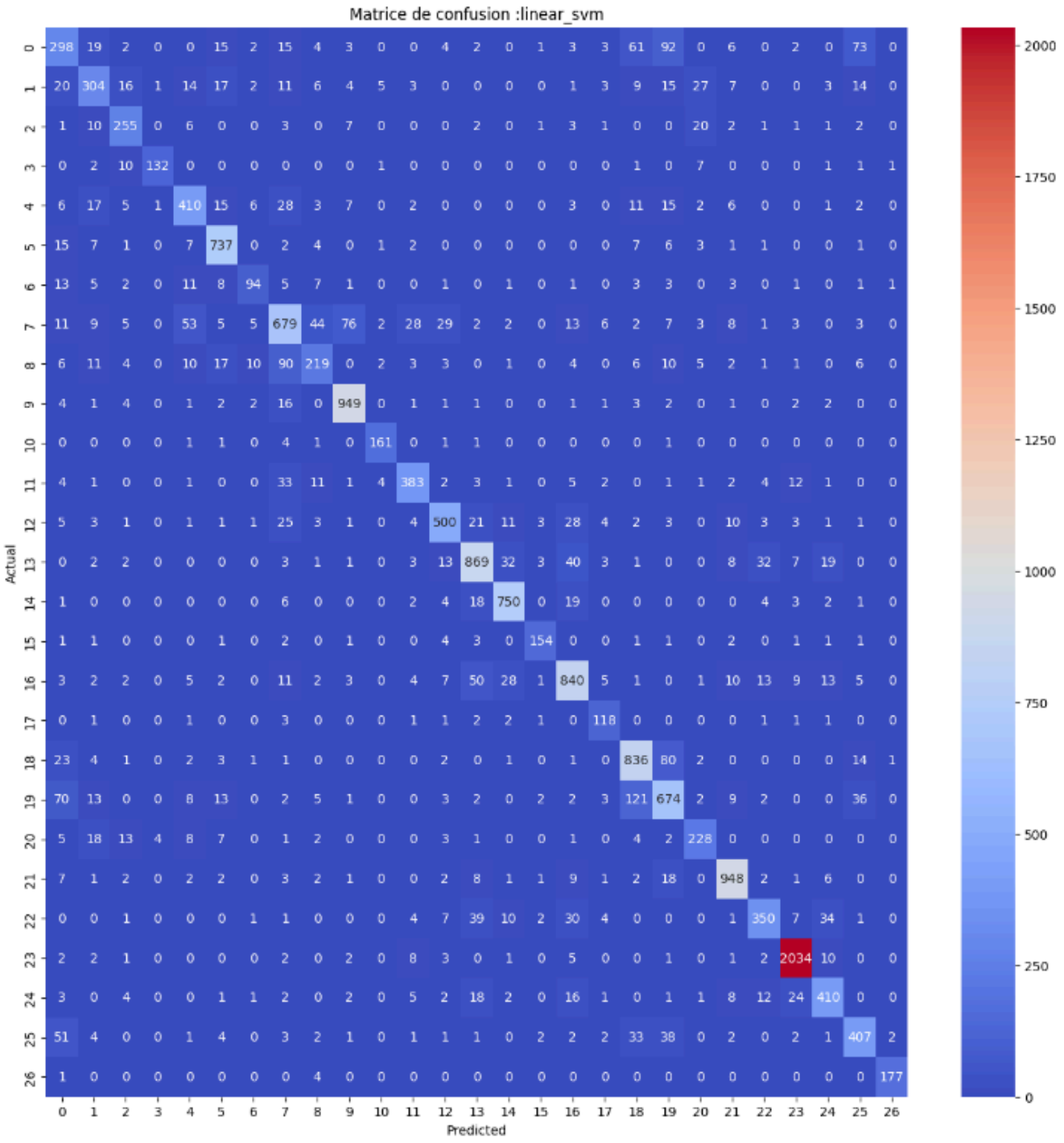
Finalement, nous avons décidé de ne retenir que le modèle le plus performant avec LinearSVM sans ré-échantillonnage ni optimisation et pour paramètres et résultats :

- `Paramètres : C=0.7399651076649312, max_iter=10000`
- `Accuracy : 0.8193, F1-score : 0.8171`

Rapport de classification :

	precision	recall	f1-score	support
10	0.54	0.49	0.52	605
40	0.70	0.63	0.66	482
50	0.77	0.81	0.79	316
60	0.96	0.85	0.90	156
1140	0.76	0.76	0.76	540
1160	0.87	0.93	0.90	795
1180	0.75	0.58	0.66	161
1280	0.71	0.68	0.70	996
1281	0.68	0.53	0.60	411
1300	0.89	0.95	0.92	994
1301	0.91	0.94	0.93	171
1302	0.84	0.81	0.83	472
1320	0.84	0.79	0.81	635
1560	0.83	0.84	0.83	1039
1920	0.89	0.93	0.91	810
1940	0.90	0.89	0.89	174
2060	0.82	0.83	0.82	1017
2220	0.75	0.89	0.81	133
2280	0.76	0.86	0.81	972
2403	0.69	0.70	0.70	968
2462	0.75	0.77	0.76	297
2522	0.91	0.93	0.92	1019
2582	0.82	0.71	0.76	492
2583	0.96	0.98	0.97	2074
2585	0.81	0.80	0.80	513
2705	0.72	0.73	0.72	560
2905	0.97	0.97	0.97	182
accuracy			0.82	16984
macro avg	0.81	0.80	0.80	16984
weighted avg	0.82	0.82	0.82	16984

Matrice de confusion :



On peut voir que la classe avec le plus de produits est la classe 23 qui correspond au prdtypecode 2583 sur laquelle le modèle réalise des prédictions excellentes étant donné qu'il s'agit de la classe majoritaire (cf § 3.3 Histogramme : prdtypecode).

5.2.2 - Classification des produits (image)

A) Orientations générales

Pour la partie image, nous avons choisi les modèles de Deep Learning suivants :

- un modèle de base de Keras : Sequential
- le modèle utilisé par Rakuten : ResNet50

Le modèle Sequential a été importé de la bibliothèque Keras et a été initialisé avec :

- couche de convolution Conv2D, pooling MaxPooling2D et régularisation Dropout x3
- couche d'aplatissement Flatten
- couche dense de 512 neurones avec une fonction d'activation ReLU
- couche de régularisation Dropout
- couche de sortie avec une fonction d'activation softmax

Le modèle ResNet50 a été importé de la bibliothèque Keras et est pré-entraîné sur l'ensemble de donnée ImageNet, sur lequel nous avons ajouté :

- couche dense de 128 neurones avec une fonction d'activation ReLU
- couche de sortie avec une fonction d'activation softmax

Les deux modèles ont été entraînés dans un premier temps avec une faible augmentation des données suivi d'une forte augmentation des données afin de comparer les résultats obtenus, avec les paramètres suivants :

- `epochs = 20, batch_size = 8, learning_rate = 0.001`

Faible augmentation des données :

- test : `rescale=1./255`
- train : `rescale=1./255, horizontal_flip=True, zoom_range=0.2`

Forte augmentation des données :

- test : `rescale=0.1`
- train : `rescale=0.1, zoom_range=0.1, rotation_range=10, width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True`

Nous utilisons pour chaques modèles une fonction callback ReduceLROnPlateau avec un learning rate minimum de 0.0001 qui permet de réduire le taux d'apprentissage lorsqu'une mesure cesse de s'améliorer. Le nombre d'époques autorisées sans amélioration après lesquelles le taux d'apprentissage sera réduit a été initialisé à 4 époques, d'après nos observations sur l'évolution de la fonction de perte.

B) Présentation des modèles

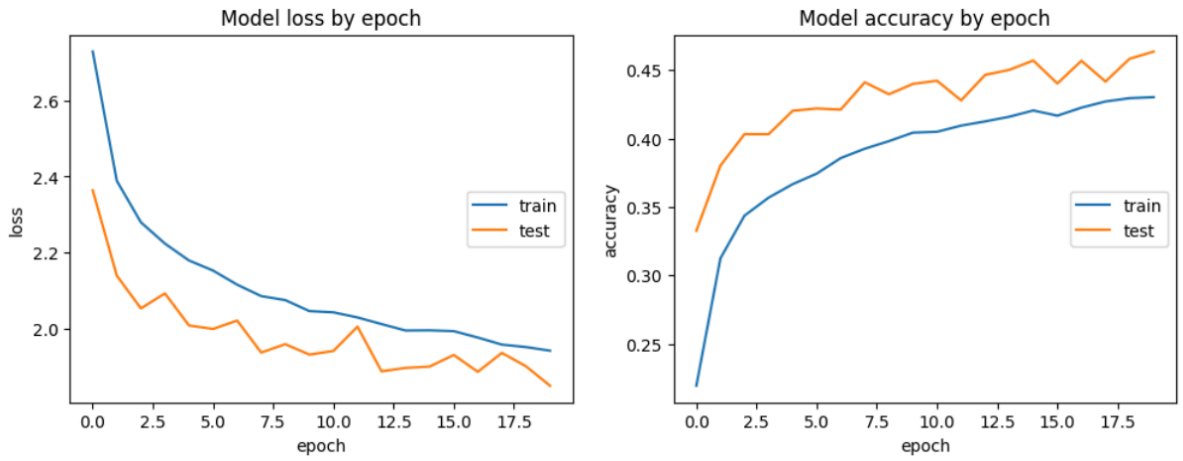
1) Modèle Sequential (faible augmentation des données)

- Total params: 11183195 (42.66 MB)
- Trainable params: 11183195 (42.66 MB)
- Non-trainable params: 0 (0.00 Byte)

```
Epoch 10/20
2123/2123 [==============================] - 1019s 480ms/step -
loss: 2.0451 - accuracy: 0.4044 - f1_score: 0.3040 -
val_loss: 1.9304 - val_accuracy: 0.4400 - val_f1_score: 0.3227

Epoch 20/20
2123/2123 [==============================] - 914s 430ms/step -
loss: 1.9407 - accuracy: 0.4303 - f1_score: 0.3383 -
val_loss: 1.8483 - val_accuracy: 0.4636 - val_f1_score: 0.3583

531/531 [==============================] - 57s 105ms/step -
loss: 1.8483 - accuracy: 0.4636 - f1_score: 0.3583
```



On peut voir sur les graphiques que les prédictions réalisées à partir du modèle retournent des résultats équivalents à ceux obtenus durant la phase d'entraînement, ce qui nous permet d'écarter l'éventualité d'un sur-apprentissage du modèle.

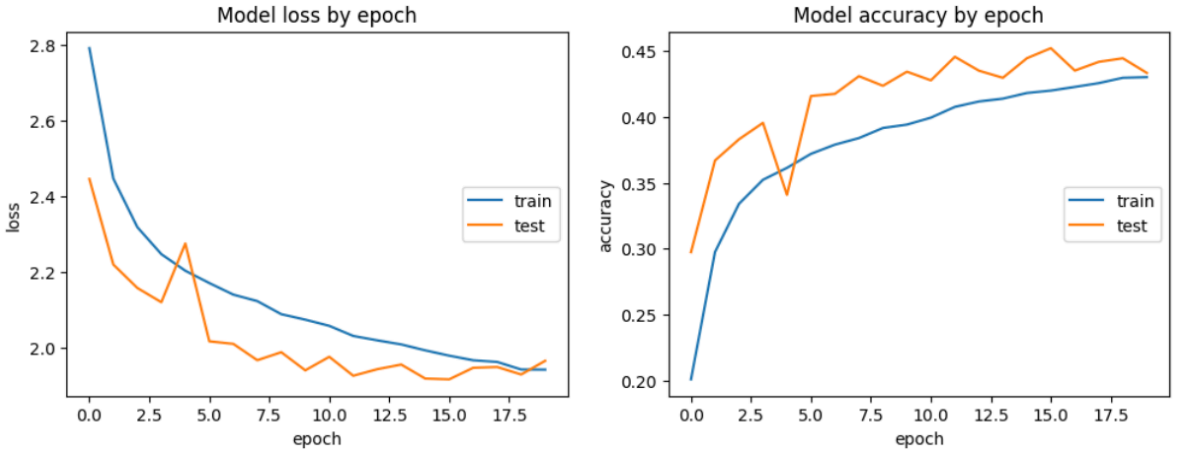
2) Modèle Sequential (forte augmentation des données)

- Total params: 11183195 (42.66 MB)
- Trainable params: 11183195 (42.66 MB)
- Non-trainable params: 0 (0.00 Byte)

```
Epoch 10/20
2123/2123 [==============================] - 1465s 690ms/step -
loss: 2.0745 - accuracy: 0.3942 - f1_score: 0.2891 -
val_loss: 1.9408 - val_accuracy: 0.4344 - val_f1_score: 0.3113

Epoch 20/20
2123/2123 [==============================] - 1009s 475ms/step -
loss: 1.9426 - accuracy: 0.4302 - f1_score: 0.3387 -
val_loss: 1.9659 - val_accuracy: 0.4335 - val_f1_score: 0.3174

531/531 [==============================] - 59s 109ms/step -
loss: 1.9659 - accuracy: 0.4335 - f1_score: 0.3174
```



On remarque cette fois encore que le modèle ne présente pas de sur-apprentissage, malgré un pic significatif aux alentours de la quatrième époque. On relève des scores plus faibles, mais, on constate que la pente de croissance de l'accuracy est encore forte. L'augmentation du nombre d'epochs permettrait alors d'améliorer les scores.

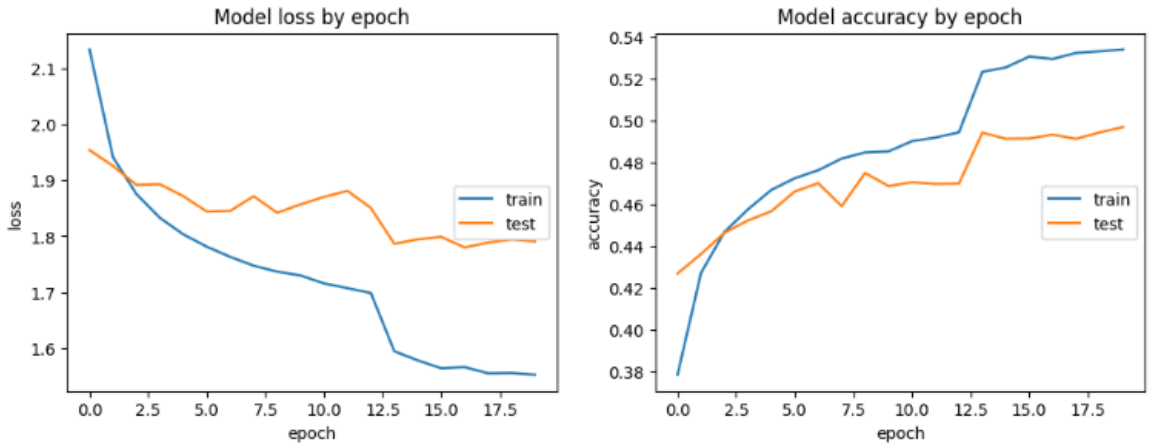
3) Modèle ResNet50 (forte augmentation des données)

- Total params: 23853467 (90.99 MB)
- Trainable params: 265755 (1.01 MB)
- Non-trainable params: 23587712 (89.98 MB)

```
Epoch 10/20
8492/8492 [==============================] - 2841s 335ms/step -
loss: 1.7300 - accuracy: 0.4851 - f1_score: 0.4022 -
val_loss: 1.8567 - val_accuracy: 0.4686 - val_f1_score: 0.3902

Epoch 20/20
8492/8492 [==============================] - 2500s 294ms/step -
loss: 1.5529 - accuracy: 0.5338 - f1_score: 0.4621 -
val_loss: 1.7908 - val_accuracy: 0.4968 - val_f1_score: 0.4191

2123/2123 [==============================] - 329s 154ms/step -
loss: 1.7908 - accuracy: 0.4968 - f1_score: 0.4191
```



En comparant les courbes entre l'ensemble d'entraînement et celui de test, on constate que les résultats obtenus sur les prédictions sont nettement inférieurs ce qui indique la présence d'un phénomène de sur-apprentissage du modèle.

C) Tableau récapitulatif

Résultats des modèles avec faible ou forte augmentation des données :

	Faible augmentation des données		Forte augmentation des données	
	Accuracy	**F1-score**	**Accuracy**	**F1-score**
Sequential	0.4636	0.3583	0.4335	0.3174
	Pas de sur-apprentissage		Pas de sur-apprentissage	
ResNet50	/	/	0.4968	0.4191
	/		Problème de sur-apprentissage	

D) Modèle retenu

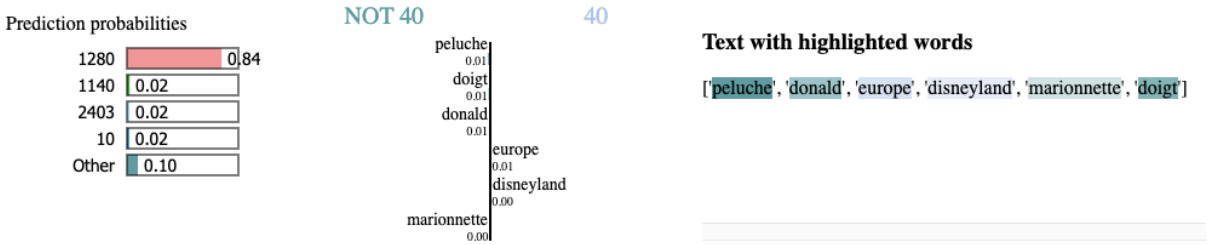
Finalement, on ne retiendra que le modèle Sequential avec une faible augmentation des données car il présente les meilleurs scores tout en évitant le sur-apprentissage du modèle :

- Paramètres : epochs=20, batch_size=8, learning_rate=0.001
- Accuracy : 0.4636, F1-score : 0.3583

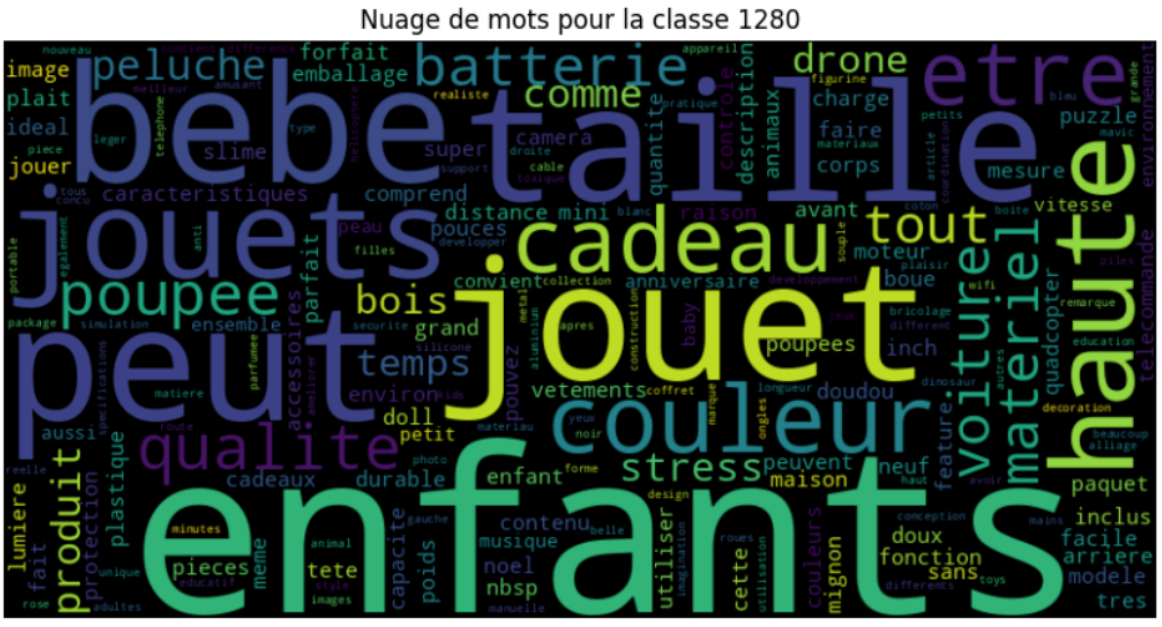
Il serait également possible d'améliorer notre modèle en augmentant significativement le nombre d'époques tout en ajoutant une callback de type EarlyStopping. Nous pourrions aussi tenter d'optimiser les hyperparamètres à l'aide d'une grille de recherche.

5.3 - Interprétation des résultats

Nous avons utilisé la librairie LIME pour interpréter les modèles de Machine Learning sur notre corpus de texte. Le but était de comprendre comment le modèle réussi à prédire le code type d'un nouveau produit en fonction des mots présents dans leur description :



La restitution obtenue avec Lime ne permet pas d'affichage dynamique des résultats ce qui ne nous a pas permis de vérifier les probabilités sur la classe majoritaire. On peut tout de même constater que notre texte a une probabilité de 0.84 d'appartenir à la classe 1280, que l'on peut retrouver à travers son nuage de mots associé :



L’amélioration des performances est le résultat du choix du modèle, d’un prétraitement des données et d’un ajustement des paramètres. Ces bonnes pratiques ont pu être appliquées différemment dans le cas de la classification textuelle et de la classification des images.

- Classification de texte

Amélioration à l’aide des méthodes NLP et de l’optimisation des hyperparamètres. Au regard de la complexité des modèles de machine learning, nous avons pu étudier de nombreux modèles et appliquer des combinaisons d’hyperparamètres variées permettant d’obtenir de meilleurs résultats.

- Classification des images

Les modèles de deep learning étant complexes et long à entraîner, nous avons dû limiter nos choix en termes de modèles. Nous avons appliqué des phases de prétraitement (réduction des images en 125x125 et en nuances de gris) et d’ajustement lors de l’exécution des entraînements, plus particulièrement à l’aide des fonctions callback.

Cependant, les résultats obtenus ne reste pas très bon et il serait nécessaire de poursuivre l’étude avec d’autres paramétrages, voire d’autres modèles complémentaires.

6 - Conclusion du projet

6.1 - Difficultés rencontrées lors du projet

- Gestion du Github

Nous avons plusieurs fois réalisé des groupes de deux pour réaliser les différentes tâches dans le but d'optimiser le temps alloué par le projet. Ainsi, la gestion du Github était plutôt simple avec un minimum de revue de code et un nombre limité de pull request à réaliser. Malgré tout, nous avons eu de nombreux conflits pas toujours simples à résoudre.

- Temps d'exécution des modèles

Les modèles de machine learning et plus particulièrement de deep learning, nécessitent l'utilisation de machines performantes. Ceci a été pénalisant, en particulier, lors de l'exécution de modèle de deep learning qui nécessitent à minima 10h de traitement pour réaliser 20 epochs. Cela a restreint alors le nombre d'expériences de deep learning exécutées dans le cadre du projet.

- Interprétabilité des modèles

Concernant la partie texte, nous avons rencontré des difficultés sur la partie interprétabilité avec LIME qui utilise la méthode predict_proba pour l'interprétation des modèles alors que LinearSVM ne la supporte pas. Nous avons donc ajouté une fonction de prédiction qui permet de transformer les données d'entrée en un tableau 2D pour LIME.

- Classification des images

Après avoir réalisé des prédictions sur nos modèles (sequential CNN et resnet50), les résultats obtenus pour le rapport de classification et la matrice de confusion ne sont pas en cohérence avec ceux obtenus lors de l'entraînement de nos modèles (ex : accuracy du modèle à 0.4636 et accuracy du rapport à 0.0600). De plus, nous n'avons pas eu le temps d'utiliser Grad-CAM pour l'interprétabilité de nos modèles.

6.2 - Bilan et conclusion

A travers ce cas pratique, nous avons constaté l'importance de la classification multiclasse et multimodale (texte et image) pour atteindre les objectifs du Challenge Rakuten.

Le pre-processing aura été une des parties les plus cruciales pour préparer nos données, que ce soit pour la partie texte ou image, afin qu'elles soient exploitables par nos modèles. Elle nous a également permis de réduire considérablement le temps d'entraînement de nos modèles et d'augmenter leur performance.

Côté modélisation, plusieurs modèles de Machine Learning pour la partie texte et de Deep Learning pour la partie image ont été entraînés. Cela nous a permis d'orienter les différentes étapes pour mieux comprendre les défis associés à ce type de problème.

Finalement, nous avons opté pour le modèle LinearSVM pour le texte et pour le modèle Sequential pour les images. Ces choix ont été motivés du fait de leur bonnes performances (accuracy et f1-score) et du fait qu'ils ne donnent pas lieu à des problèmes de sur-apprentissage.

	Modèle	**Accuracy**	**F1-score**
Classification : texte	LinearSVM	0.8193	0.8171
Classification : image	Sequential	0.4636	0.3583

6.3 - Prospectives sur le projet

A) Gestion du flux de travail

Il serait intéressant de mettre en place un MLFlow sur le projet afin de gérer les flux de travail, d'optimiser le cycle de vie des modèles et de stocker proprement les résultats obtenus lors de leur entraînement. Ainsi, il serait possible de déployer les modèles retenus pour prédire aisément de nouvelles données.

B) Classification de texte

Étant donné les scores plus que correct obtenus sur la partie classification de texte avec plus de 0.80 sur des modèles de base, nous avons essayé de mettre en place des modèles de Deep Learning avec BERT de Hugging Face, sans grand succès par manque de temps. Il serait donc intéressant de tester de nouveaux modèles CNN pour la partie corpus NLP.

C) Classification des images

Pour ce qui est des images, les scores obtenus par nos modèles ne permettent pas de confirmer leur bonne efficacité malgré qu'ils soient proches de ceux proposés par Rakuten. La partie pre-processing devrait certainement être améliorée afin de rendre le jeu de données plus exploitable pour nos modèles. Nous pourrions également mettre en place de nouveaux modèles de Deep Learning (ex : RNN, TNN ou d'autres modèles Hugging Face).

Il serait aussi nécessaire de visualiser quelles parties de l'image ont le plus influencé la décision du modèle, pour mieux comprendre le pourquoi de certaines prises de décision du modèle, une fonction qu'utilise la technique du Grad-CAM en conduisant au diagnostic des erreurs et à l'amélioration du modèle.

D) Multimodal learning

Enfin, nous n'avons pas eu le temps de mettre en place la partie Multimodal Learning pour regrouper nos scores (texte et image) malgré nos recherches sur des modèles multimodaux à double encodeur ou à encodeur commun. Cette dernière partie permettrait d'associer le texte et l'image pour la prédiction des produits e-commerce de Rakuten.

7 - Bibliographie

- **Racinisation VS Lemmatisation**

https://nirajbhoi.medium.com/stemming-vs-lemmatization-in-nlp-efc280d4e845

- **CountVectorizer VS TfidfVectorizer**

https://medium.com/@shandeep92/countvectorizer-vs-tfidfvectorizer-cf62d0a54fa4

- **OneVsRest et OneVsOne**

https://medium.com/@agrawalsam1997/multiclass-classification-onevsrest-and-onevsone-classification-strategy-2c293a91571a

- **Optimisation bayésienne des hyperparamètres**

https://scikit-optimize.github.io/stable/index.html

- **Optimisation des applications Scikit-learn**

https://intel.github.io/scikit-learn-intelex/latest/

- **Modèle Sequential de Keras**

https://keras.io/guides/sequential_model/

- **Modèle ResNet50 de Keras**

https://keras.io/api/applications/resnet/

8 - Annexes

- **Annexe n°1 : informations du DataFrame**

```
<class 'pandas.core.frame.DataFrame'>
Index: 84916 entries, 0 to 84915
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   designation  84916 non-null  object
 1   description  55116 non-null  object
 2   productid    84916 non-null  int64 
 3   imageid      84916 non-null  int64 
 4   prdtypecode  84916 non-null  int64 
dtypes: int64(3), object(2)
memory usage: 5.9+ MB
```

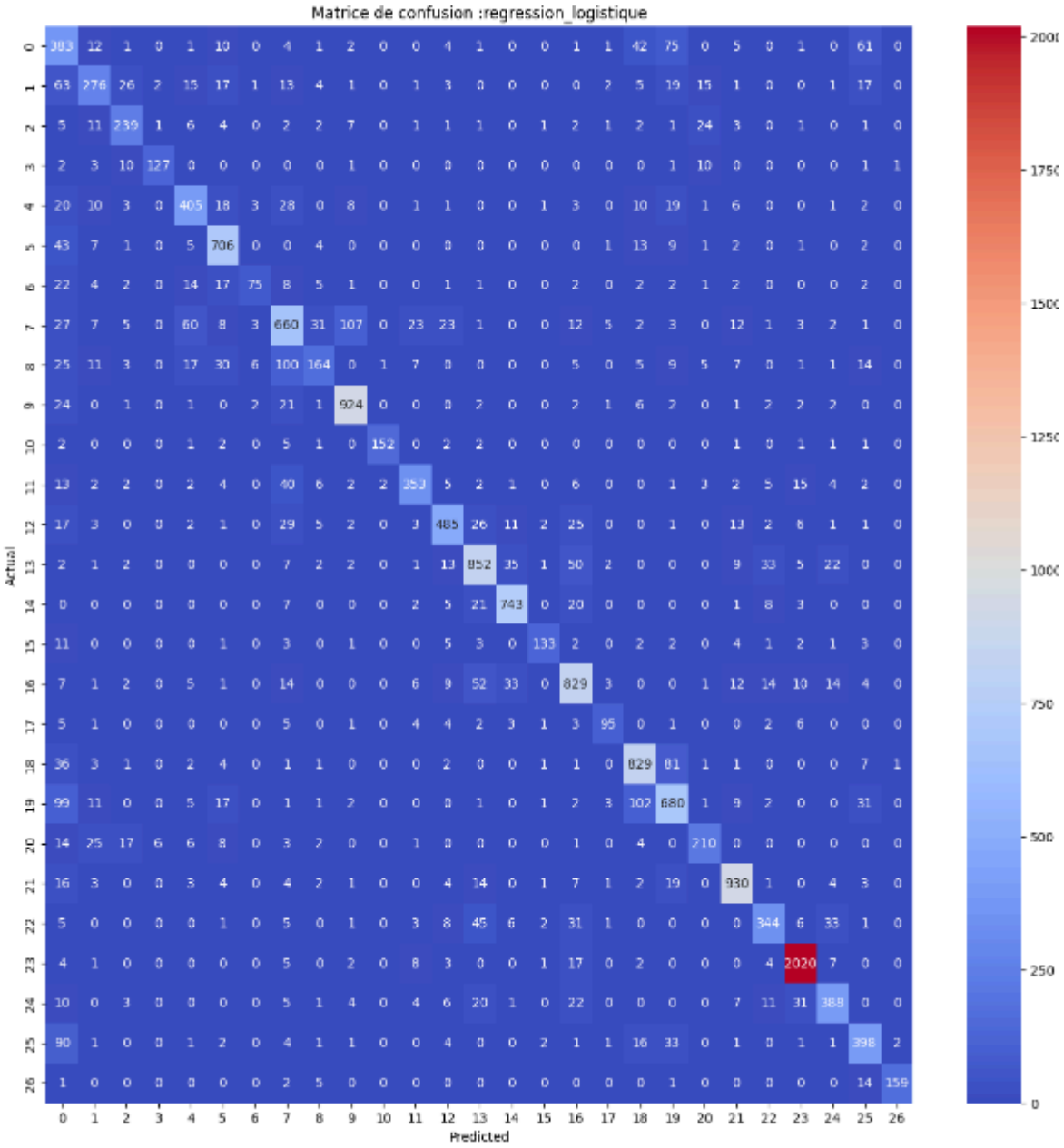
- **Annexe n°2 : statistiques descriptives des variables quantitatives**

	productid	imageid	prdtypecode
count	8.491600e+04	8.491600e+04	84916.000000
mean	2.555468e+09	1.152691e+09	1773.219900
std	1.588656e+09	1.751427e+08	788.179885
min	1.839120e+05	6.728400e+04	10.000000
25%	6.760519e+08	1.056269e+09	1281.000000
50%	3.190506e+09	1.213354e+09	1920.000000
75%	3.995599e+09	1.275646e+09	2522.000000
max	4.252012e+09	1.328824e+09	2905.000000

- **Annexe n°3 : Régression logistique (rapport de classification)**

	precision	recall	f1-score	support
10	0.40	0.63	0.49	605
40	0.70	0.57	0.63	482
50	0.75	0.76	0.75	316
60	0.93	0.81	0.87	156
1140	0.74	0.75	0.74	540
1160	0.83	0.89	0.86	795
1180	0.83	0.47	0.60	161
1280	0.68	0.66	0.67	996
1281	0.69	0.40	0.50	411
1300	0.86	0.93	0.90	994
1301	0.98	0.89	0.93	171
1302	0.84	0.75	0.79	472
1320	0.82	0.76	0.79	635
1560	0.81	0.82	0.82	1039
1920	0.89	0.92	0.90	810
1940	0.90	0.76	0.83	174
2060	0.79	0.82	0.80	1017
2220	0.81	0.71	0.76	133
2280	0.79	0.85	0.82	972
2403	0.71	0.70	0.71	968
2462	0.77	0.71	0.74	297
2522	0.90	0.91	0.91	1019
2582	0.80	0.70	0.75	492
2583	0.96	0.97	0.96	2074
2585	0.80	0.76	0.78	513
2705	0.70	0.71	0.71	560
2905	0.98	0.87	0.92	182
accuracy			0.80	16984
macro avg	0.80	0.76	0.78	16984
weighted avg	0.80	0.80	0.80	16984

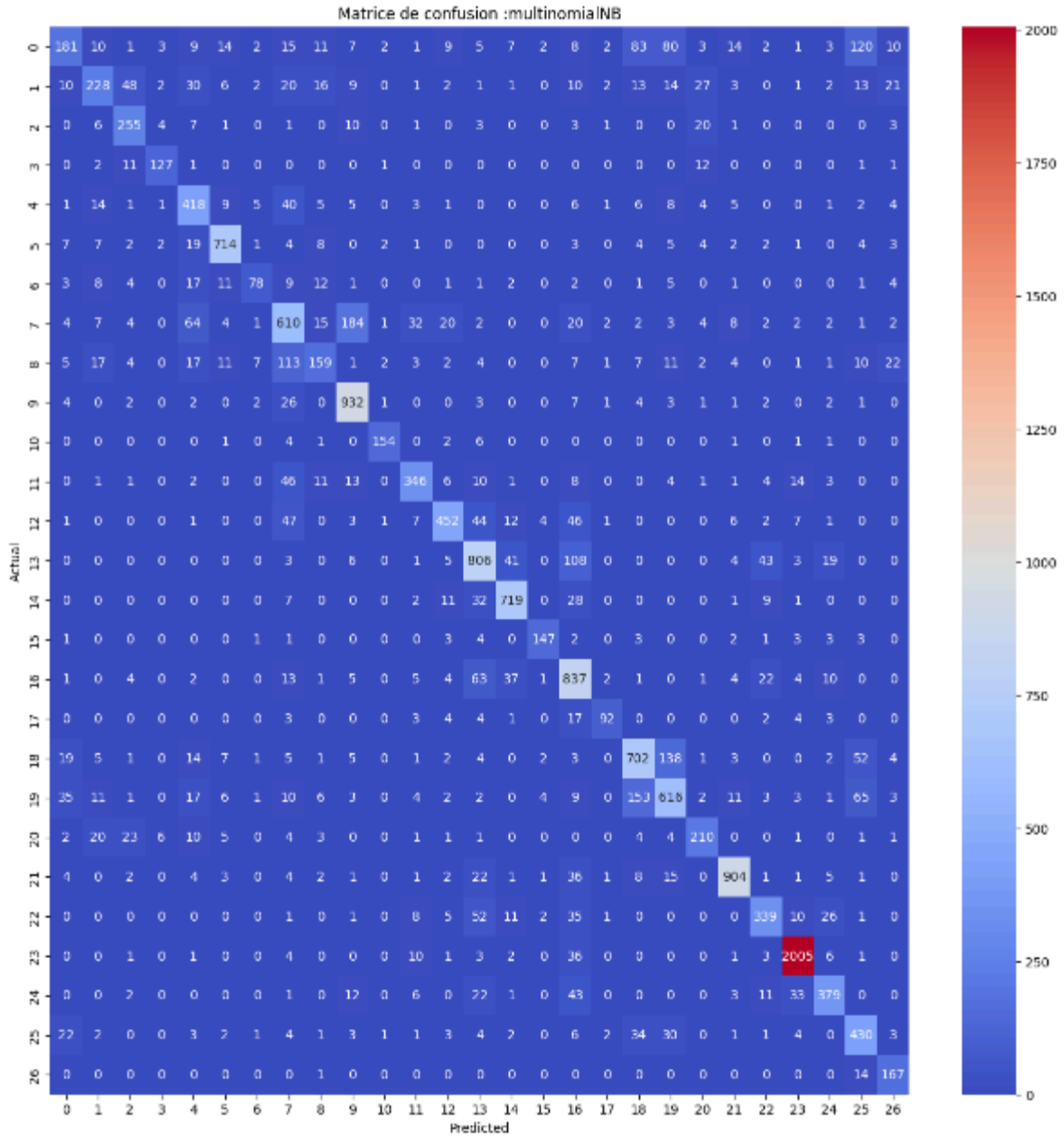
- **Annexe n°4 : Régression logistique (matrice de confusion)**



- **Annexe n°5 : Naive Bayes : MultinomialNB (rapport de classification)**

	precision	recall	f1-score	support
10	0.60	0.30	0.40	605
40	0.67	0.47	0.56	482
50	0.69	0.81	0.75	316
60	0.88	0.81	0.84	156
1140	0.66	0.77	0.71	540
1160	0.90	0.90	0.90	795
1180	0.76	0.48	0.59	161
1280	0.61	0.61	0.61	996
1281	0.63	0.39	0.48	411
1300	0.78	0.94	0.85	994
1301	0.93	0.90	0.92	171
1302	0.79	0.73	0.76	472
1320	0.84	0.71	0.77	635
1560	0.73	0.78	0.75	1039
1920	0.86	0.89	0.87	810
1940	0.90	0.84	0.87	174
2060	0.65	0.82	0.73	1017
2220	0.84	0.69	0.76	133
2280	0.68	0.72	0.70	972
2403	0.66	0.64	0.65	968
2462	0.72	0.71	0.71	297
2522	0.92	0.89	0.90	1019
2582	0.76	0.69	0.72	492
2583	0.95	0.97	0.96	2074
2585	0.81	0.74	0.77	513
2705	0.60	0.77	0.67	560
2905	0.67	0.92	0.78	182
accuracy			0.77	16984
macro avg	0.76	0.74	0.74	16984
weighted avg	0.77	0.77	0.76	16984

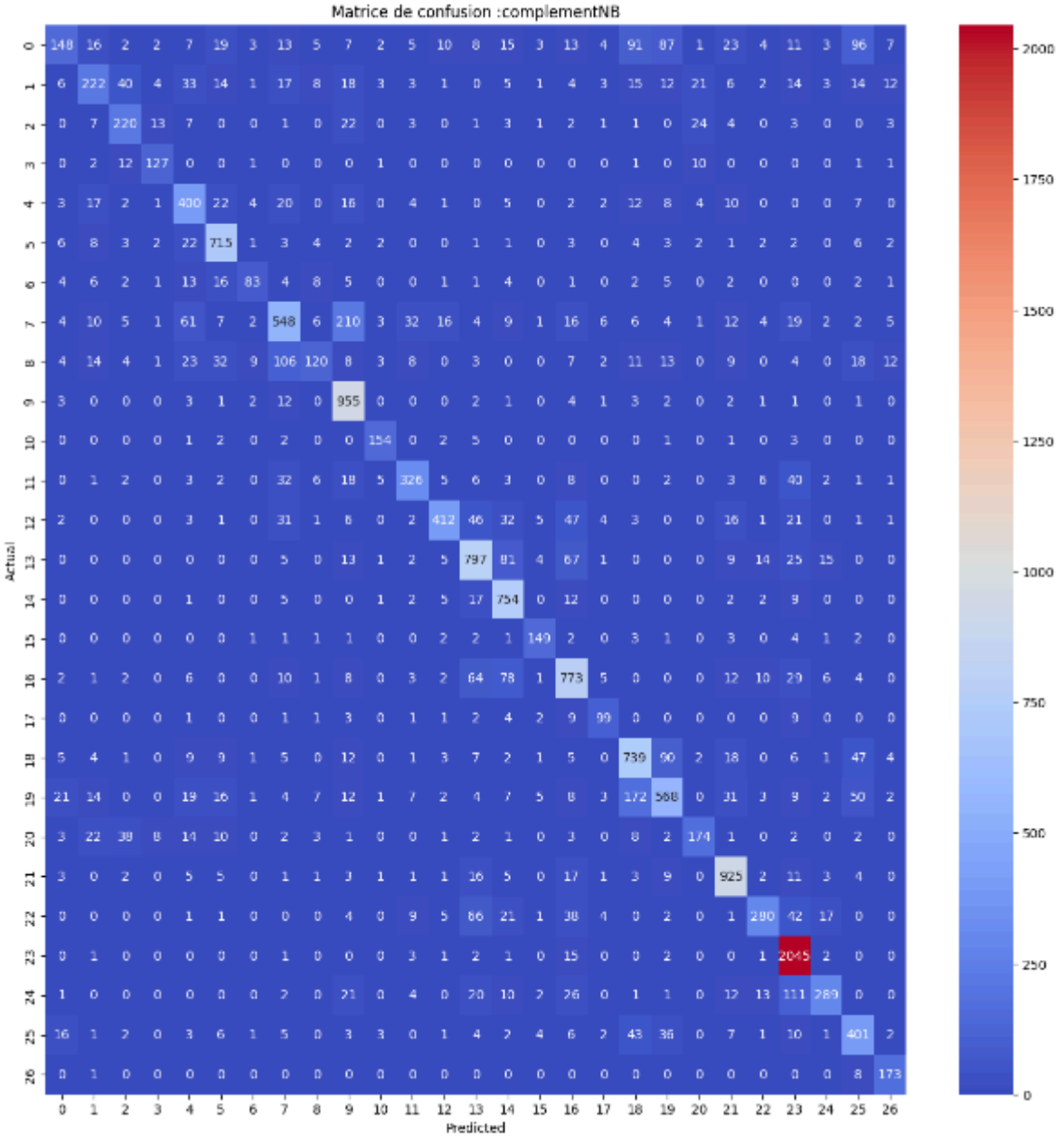
- **Annexe n°6 : Naive Bayes : MultinomialNB (matrice de confusion)**



- **Annexe n°7 : Naive Bayes : ComplementNB (rapport de classification)**

	precision	recall	f1-score	support
10	0.64	0.24	0.35	605
40	0.64	0.46	0.54	482
50	0.65	0.70	0.67	316
60	0.79	0.81	0.80	156
1140	0.63	0.74	0.68	540
1160	0.81	0.90	0.85	795
1180	0.75	0.52	0.61	161
1280	0.66	0.55	0.60	996
1281	0.70	0.29	0.41	411
1300	0.71	0.96	0.82	994
1301	0.86	0.90	0.88	171
1302	0.78	0.69	0.73	472
1320	0.86	0.65	0.74	635
1560	0.74	0.77	0.75	1039
1920	0.72	0.93	0.81	810
1940	0.83	0.86	0.84	174
2060	0.71	0.76	0.73	1017
2220	0.72	0.74	0.73	133
2280	0.66	0.76	0.71	972
2403	0.67	0.59	0.63	968
2462	0.73	0.59	0.65	297
2522	0.83	0.91	0.87	1019
2582	0.81	0.57	0.67	492
2583	0.84	0.99	0.91	2074
2585	0.83	0.56	0.67	513
2705	0.60	0.72	0.65	560
2905	0.77	0.95	0.85	182
accuracy			0.74	16984
macro avg	0.74	0.71	0.71	16984
weighted avg	0.74	0.74	0.73	16984

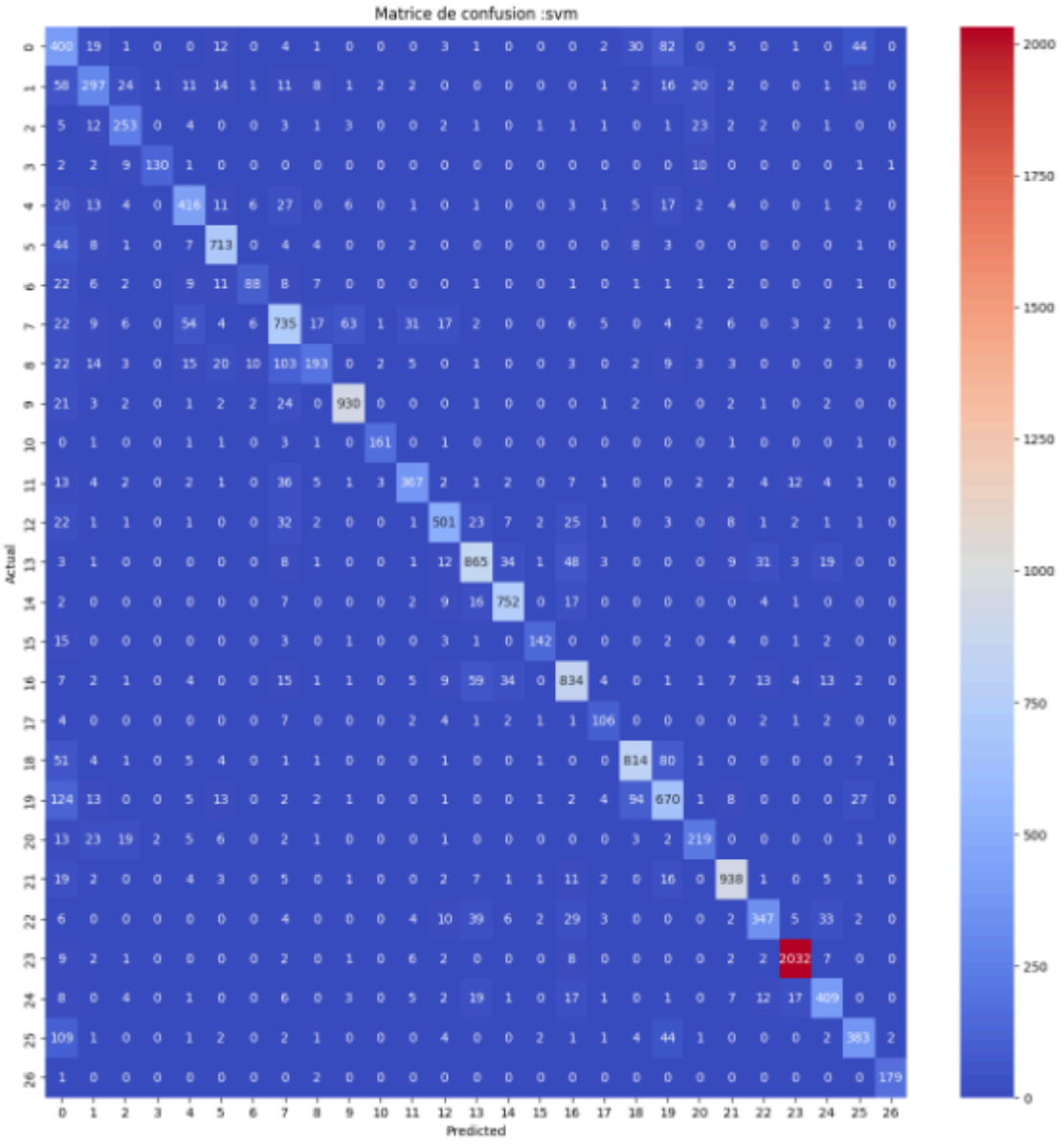
- **Annexe n°8 : Naive Bayes : ComplementNB (matrice de confusion)**



- **Annexe n°9 : SVM (rapport de classification)**

	precision	recall	f1-score	support
10	0.39	0.66	0.49	605
40	0.68	0.62	0.65	482
50	0.76	0.80	0.78	316
60	0.98	0.83	0.90	156
1140	0.76	0.77	0.77	540
1160	0.87	0.90	0.88	795
1180	0.78	0.55	0.64	161
1280	0.70	0.74	0.72	996
1281	0.78	0.47	0.59	411
1300	0.92	0.94	0.93	994
1301	0.95	0.94	0.95	171
1302	0.85	0.78	0.81	472
1320	0.85	0.79	0.82	635
1560	0.83	0.83	0.83	1039
1920	0.90	0.93	0.91	810
1940	0.92	0.82	0.87	174
2060	0.82	0.82	0.82	1017
2220	0.77	0.80	0.79	133
2280	0.84	0.84	0.84	972
2403	0.70	0.69	0.70	968
2462	0.77	0.74	0.75	297
2522	0.93	0.92	0.92	1019
2582	0.83	0.71	0.76	492
2583	0.98	0.98	0.98	2074
2585	0.81	0.80	0.80	513
2705	0.78	0.68	0.73	560
2905	0.98	0.98	0.98	182
accuracy			0.82	16984
macro avg	0.82	0.79	0.80	16984
weighted avg	0.83	0.82	0.82	16984

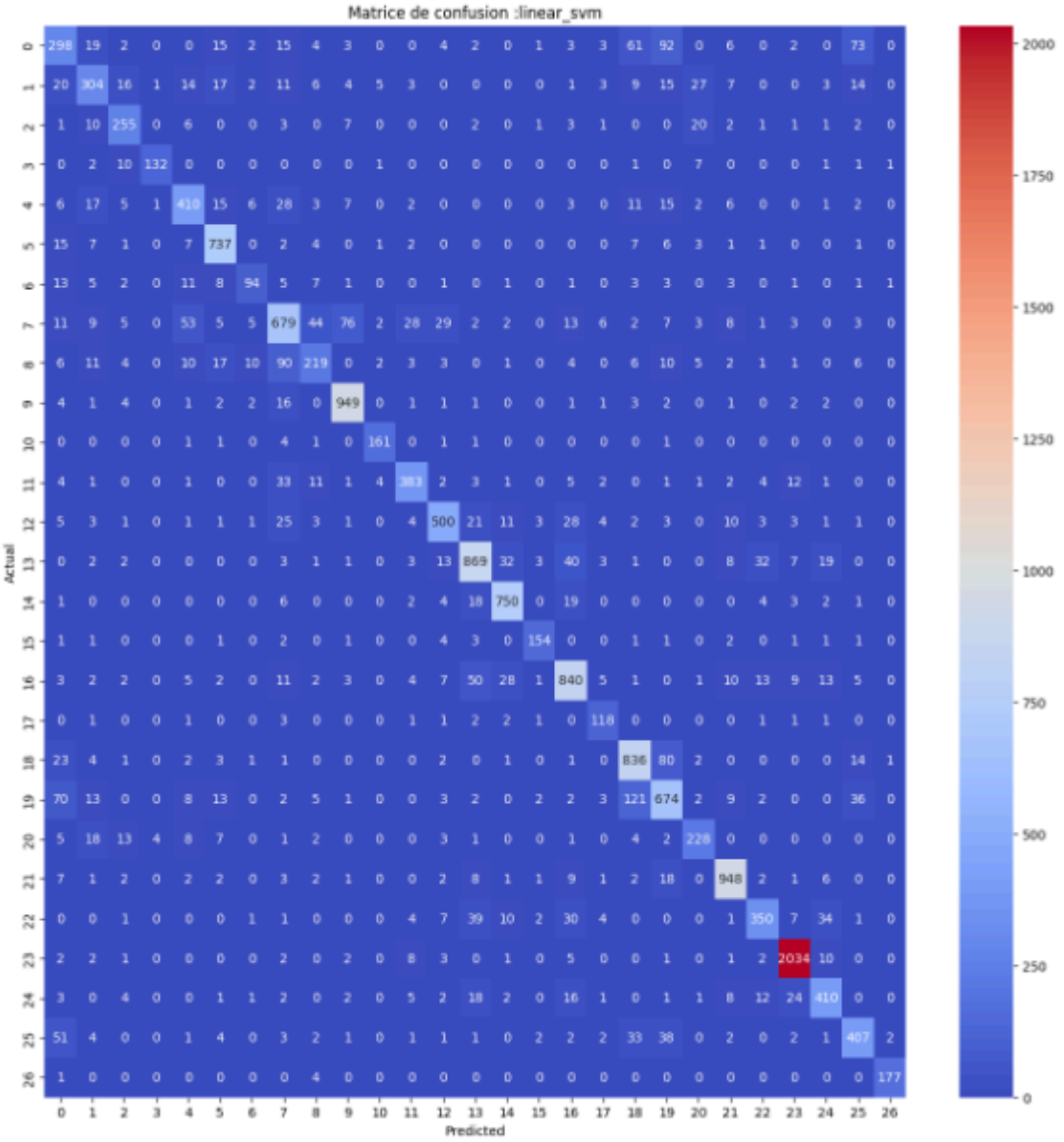
- **Annexe n°10 : SVM (matrice de confusion)**



- **Annexe n°11 : LinearSVM (rapport de classification)**

	precision	recall	f1-score	support
10	0.54	0.49	0.52	605
40	0.70	0.63	0.66	482
50	0.77	0.81	0.79	316
60	0.96	0.85	0.90	156
1140	0.76	0.76	0.76	540
1160	0.87	0.93	0.90	795
1180	0.75	0.58	0.66	161
1280	0.71	0.68	0.70	996
1281	0.68	0.53	0.60	411
1300	0.89	0.95	0.92	994
1301	0.91	0.94	0.93	171
1302	0.84	0.81	0.83	472
1320	0.84	0.79	0.81	635
1560	0.83	0.84	0.83	1039
1920	0.89	0.93	0.91	810
1940	0.90	0.89	0.89	174
2060	0.82	0.83	0.82	1017
2220	0.75	0.89	0.81	133
2280	0.76	0.86	0.81	972
2403	0.69	0.70	0.70	968
2462	0.75	0.77	0.76	297
2522	0.91	0.93	0.92	1019
2582	0.82	0.71	0.76	492
2583	0.96	0.98	0.97	2074
2585	0.81	0.80	0.80	513
2705	0.72	0.73	0.72	560
2905	0.97	0.97	0.97	182
accuracy			0.82	16984
macro avg	0.81	0.80	0.80	16984
weighted avg	0.82	0.82	0.82	16984

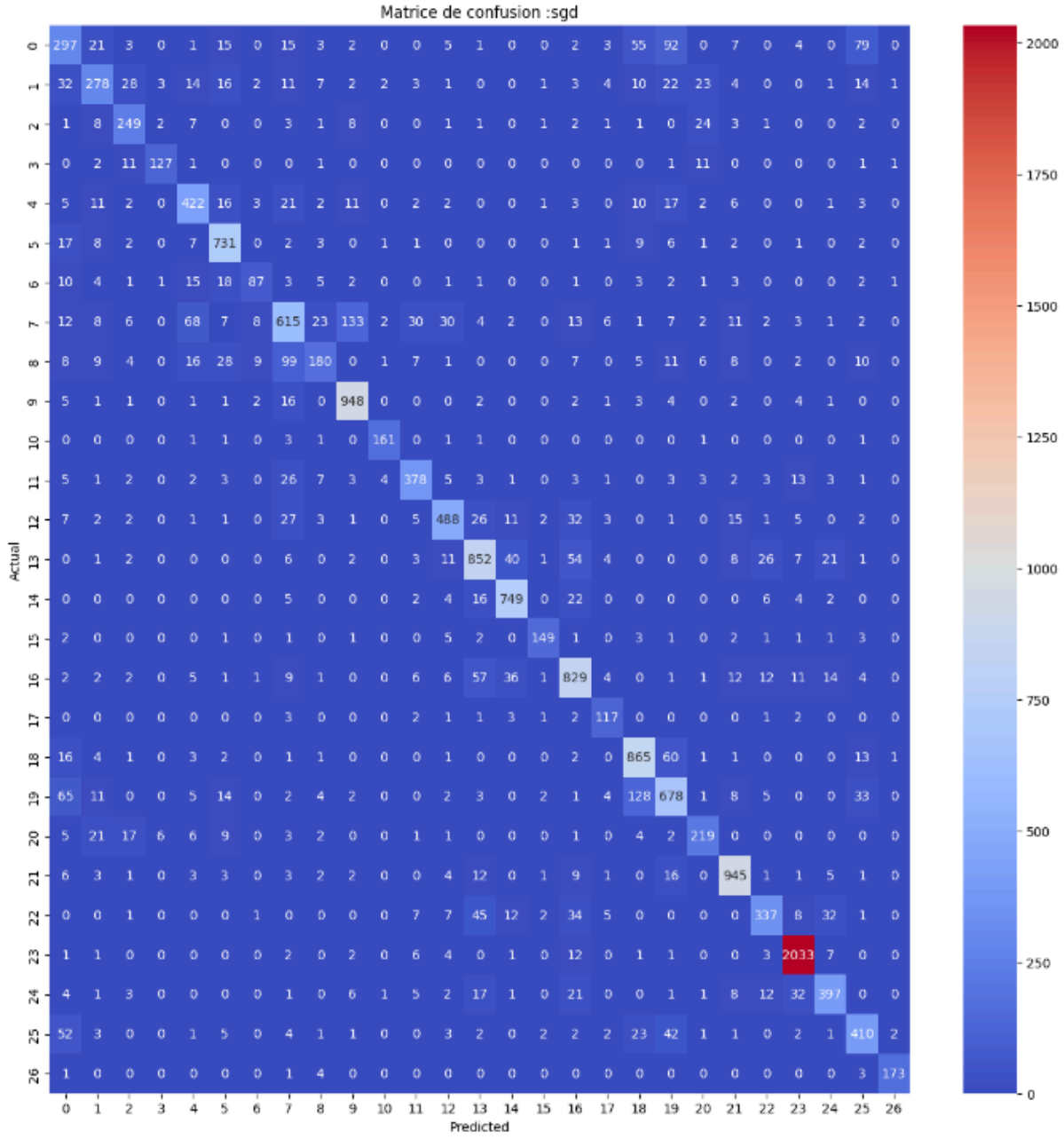
- **Annexe n°12 : LinearSVM (matrice de confusion)**



- **Annexe n°13 : SGDClassifier (rapport de classification)**

	precision	recall	f1-score	support
10	0.54	0.49	0.51	605
40	0.69	0.58	0.63	482
50	0.74	0.79	0.76	316
60	0.91	0.81	0.86	156
1140	0.73	0.78	0.75	540
1160	0.84	0.92	0.88	795
1180	0.77	0.54	0.64	161
1280	0.70	0.62	0.65	996
1281	0.72	0.44	0.54	411
1300	0.84	0.95	0.89	994
1301	0.94	0.94	0.94	171
1302	0.83	0.80	0.81	472
1320	0.83	0.77	0.80	635
1560	0.81	0.82	0.82	1039
1920	0.88	0.92	0.90	810
1940	0.91	0.86	0.88	174
2060	0.78	0.82	0.80	1017
2220	0.75	0.88	0.81	133
2280	0.77	0.89	0.83	972
2403	0.70	0.70	0.70	968
2462	0.73	0.74	0.74	297
2522	0.90	0.93	0.91	1019
2582	0.82	0.68	0.75	492
2583	0.95	0.98	0.97	2074
2585	0.82	0.77	0.79	513
2705	0.70	0.73	0.71	560
2905	0.97	0.95	0.96	182
accuracy			0.81	16984
macro avg	0.80	0.78	0.79	16984
weighted avg	0.80	0.81	0.80	16984

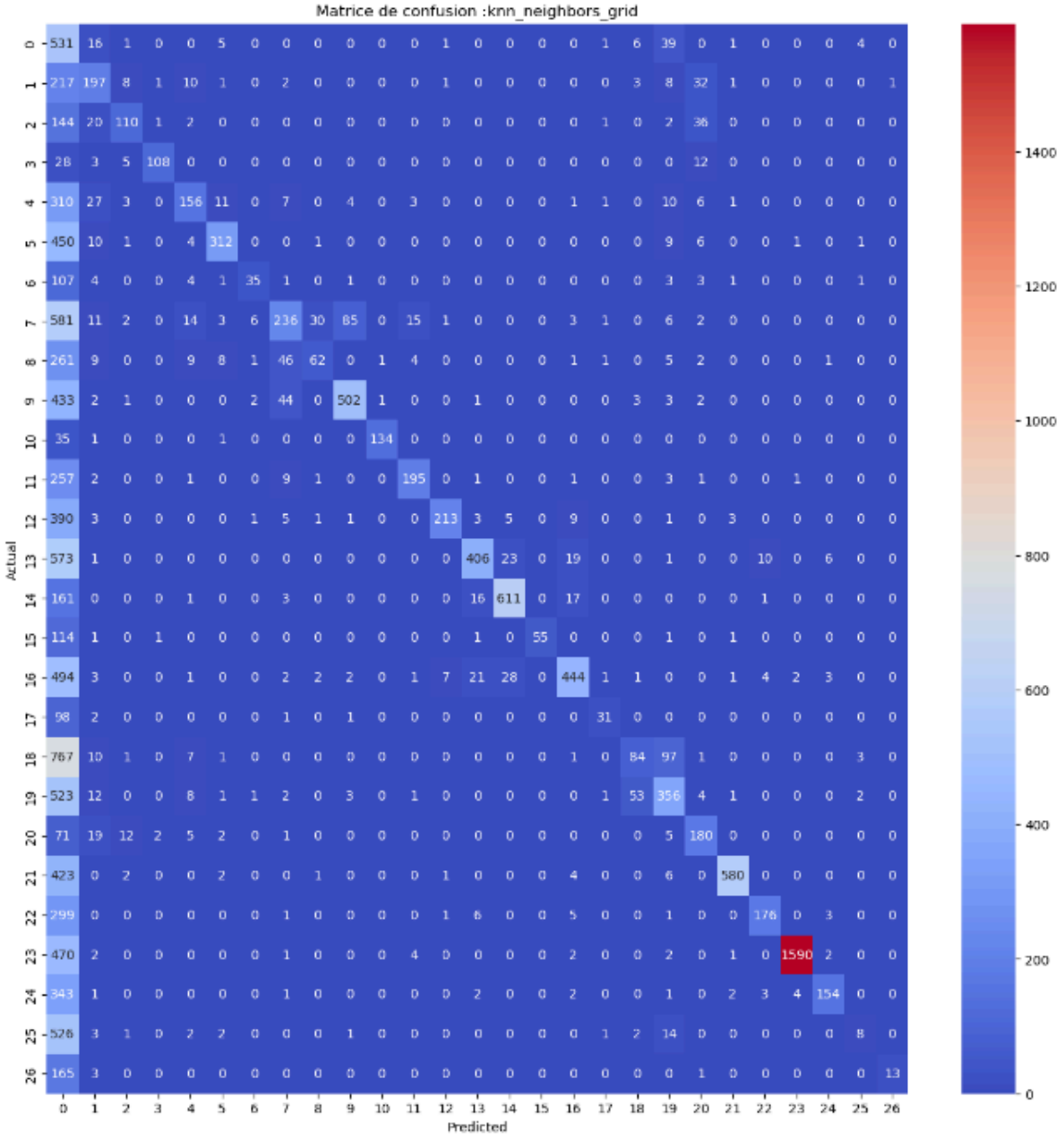
- **Annexe n°14 : SGDClassifier (matrice de confusion)**



- **Annexe n°15 : KNeighborsClassifier (rapport de classification)**

	precision	recall	f1-score	support
10	0.06	0.88	0.11	605
40	0.54	0.41	0.47	482
50	0.75	0.35	0.48	316
60	0.96	0.69	0.80	156
1140	0.70	0.29	0.41	540
1160	0.89	0.39	0.54	795
1180	0.76	0.22	0.34	161
1280	0.65	0.24	0.35	996
1281	0.63	0.15	0.24	411
1300	0.84	0.51	0.63	994
1301	0.99	0.78	0.87	171
1302	0.87	0.41	0.56	472
1320	0.95	0.34	0.50	635
1560	0.89	0.39	0.54	1039
1920	0.92	0.75	0.83	810
1940	1.00	0.32	0.48	174
2060	0.87	0.44	0.58	1017
2220	0.79	0.23	0.36	133
2280	0.55	0.09	0.15	972
2403	0.62	0.37	0.46	968
2462	0.62	0.61	0.62	297
2522	0.98	0.57	0.72	1019
2582	0.91	0.36	0.51	492
2583	0.99	0.77	0.87	2074
2585	0.91	0.30	0.45	513
2705	0.42	0.01	0.03	560
2905	0.93	0.07	0.13	182
accuracy			0.44	16984
macro avg	0.78	0.40	0.48	16984
weighted avg	0.79	0.44	0.52	16984

- **Annexe n°16 : KNeighborsClassifier (matrice de confusion)**

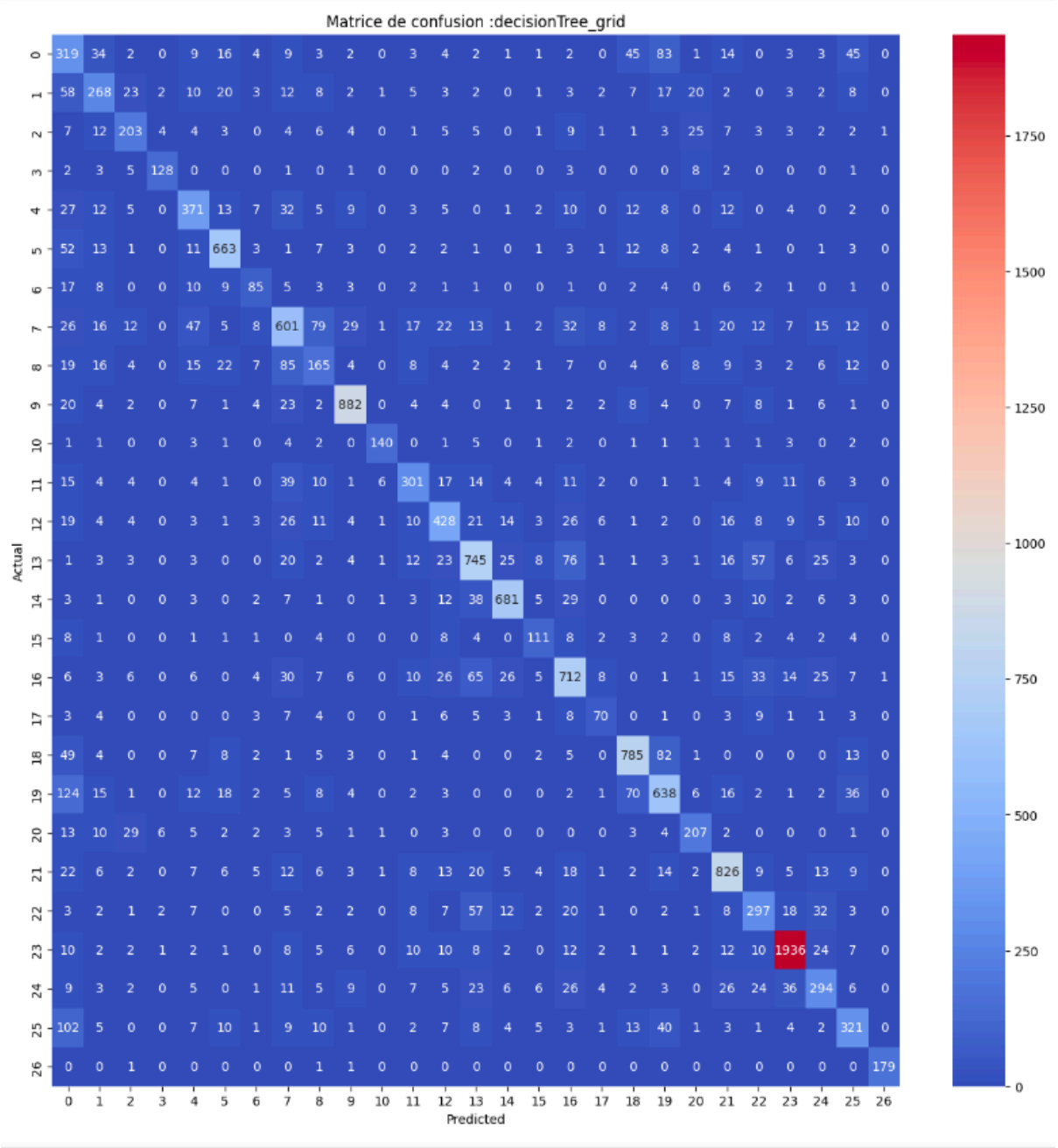


- **Annexe n°17 : DecisionTreeClassifier (rapport de classification)**

	precision	recall	f1-score	support
10	0.34	0.53	0.41	605
40	0.59	0.56	0.57	482
50	0.65	0.64	0.65	316
60	0.90	0.82	0.86	156
1140	0.66	0.69	0.68	540
1160	0.83	0.83	0.83	795
1180	0.58	0.53	0.55	161
1280	0.63	0.60	0.61	996
1281	0.45	0.40	0.42	411
1300	0.90	0.89	0.89	994
1301	0.92	0.82	0.86	171
1302	0.72	0.64	0.67	472
1320	0.69	0.67	0.68	635
1560	0.72	0.72	0.72	1039
1920	0.86	0.84	0.85	810
1940	0.66	0.64	0.65	174
2060	0.69	0.70	0.70	1017
2220	0.62	0.53	0.57	133
2280	0.81	0.81	0.81	972
2403	0.68	0.66	0.67	968
2462	0.72	0.70	0.71	297
2522	0.79	0.81	0.80	1019
2582	0.59	0.60	0.60	492
2583	0.93	0.93	0.93	2074
2585	0.62	0.57	0.60	513
2705	0.62	0.57	0.60	560
2905	0.99	0.98	0.99	182
accuracy			0.73	16984
macro avg	0.71	0.69	0.70	16984
weighted avg	0.73	0.73	0.73	16984

Score : 0.7275082430522845
F1-score : 0.7294943948246022

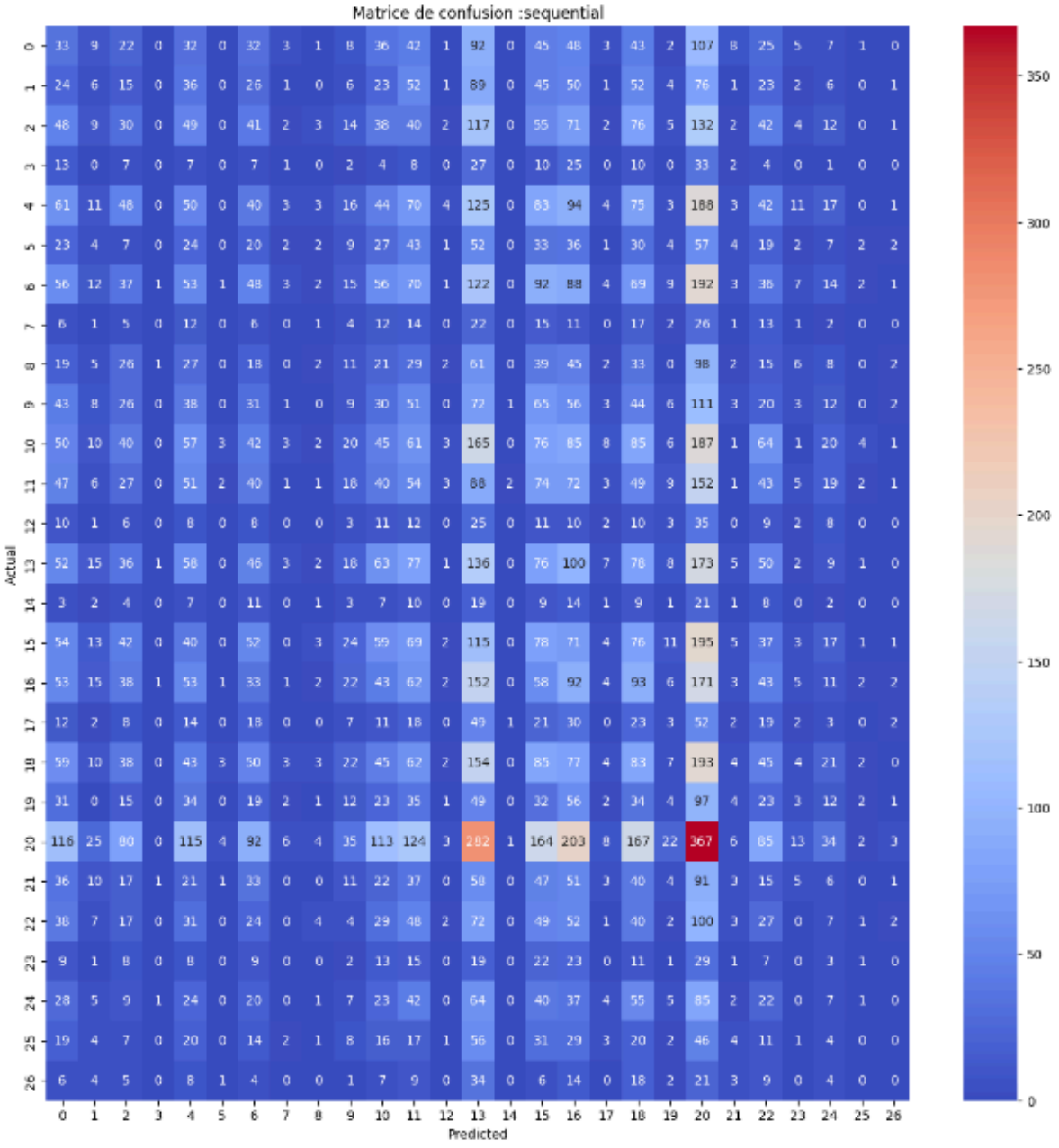
- **Annexe n°18 : DecisionTreeClassifier (matrice de confusion)**



- **Annexe n°19 : Sequential (rapport de classification)**

	precision	recall	f1-score	support
10	0.04	0.05	0.04	605
1140	0.02	0.01	0.02	540
1160	0.04	0.04	0.04	795
1180	0.00	0.00	0.00	161
1280	0.07	0.06	0.07	996
1281	0.00	0.00	0.00	411
1300	0.06	0.08	0.07	994
1301	0.03	0.01	0.01	171
1302	0.02	0.00	0.01	472
1320	0.03	0.01	0.02	635
1560	0.06	0.05	0.06	1039
1920	0.05	0.05	0.05	810
1940	0.00	0.00	0.00	174
2060	0.06	0.07	0.06	1017
2220	0.00	0.00	0.00	133
2280	0.06	0.10	0.07	972
2403	0.05	0.07	0.06	968
2462	0.00	0.00	0.00	297
2522	0.06	0.05	0.05	1019
2582	0.02	0.01	0.01	492
2583	0.12	0.24	0.16	2074
2585	0.04	0.01	0.01	513
2705	0.04	0.04	0.04	560
2905	0.01	0.01	0.01	182
40	0.03	0.01	0.02	482
50	0.00	0.00	0.00	316
60	0.00	0.00	0.00	156
accuracy			0.07	16984
macro avg	0.03	0.04	0.03	16984
weighted avg	0.05	0.07	0.06	16984

- **Annexe n°20 : Sequential (matrice de confusion)**



- **Annexe n°21 : ResNet50 (rapport de classification)**

	precision	recall	f1-score	support
10	0.02	0.02	0.02	605
1140	0.03	0.02	0.02	540
1160	0.07	0.06	0.06	795
1180	0.00	0.00	0.00	161
1280	0.07	0.07	0.07	996
1281	0.03	0.02	0.03	411
1300	0.06	0.09	0.07	994
1301	0.00	0.00	0.00	171
1302	0.02	0.02	0.02	472
1320	0.04	0.03	0.03	635
1560	0.06	0.08	0.07	1039
1920	0.04	0.04	0.04	810
1940	0.00	0.00	0.00	174
2060	0.04	0.06	0.05	1017
2220	0.00	0.00	0.00	133
2280	0.07	0.08	0.08	972
2403	0.05	0.04	0.05	968
2462	0.04	0.02	0.03	297
2522	0.06	0.05	0.06	1019
2582	0.04	0.02	0.02	492
2583	0.12	0.15	0.14	2074
2585	0.03	0.02	0.02	513
2705	0.03	0.04	0.04	560
2905	0.01	0.01	0.01	182
40	0.01	0.02	0.02	482
50	0.03	0.01	0.01	316
60	0.01	0.01	0.01	156
accuracy			0.06	16984
macro avg	0.04	0.04	0.04	16984
weighted avg	0.05	0.06	0.06	16984

- **Annexe n°22 : ResNet50 (matrice de confusion)**