UNIVERSITÉ CAEN NORMANDIE

ENSI CAEN

GREYC
Laboratoire de recherche en sciences du numérique

Rapport de stage

Investigation sur les archives numériques d'écrivains contemporains

"Classification automatique de documents"

Étudiante :
Dihia SLIMANA

Encadrants GREYC :
Emmanuel GIGUET
Tanguy GERNOT

Encadrant UNICAEN :
Gaetan RICHARD

Master 1 Cybersécurité

Projet soutenu par le ministère de la Culture

MINISTÈRE DE LA CULTURE
Liberté
Égalité
Fraternité

25 août 2025

Table des matières

Introduction

J'ai réalisé mon stage au sein du laboratoire de recherche **GREYC** (Groupe de Recherche en Informatique, Image, Automatique et Instrumentation de Caen). Le GREYC est composé de six équipes de recherche :

- **AMACC** : Recherche en informatique mathématique et modèles de calcul.
- **CODAG** : Fouille de données, contraintes et graphes.
- **ELECTRONIQUE** : Études des capteurs à haute et faible sensibilité.
- **IMAGE** : Traitement et analyse de signaux, d'images et de vidéos.
- **MAD** : Agents autonomes et systèmes multi-agents.
- **SAFE** : Recherche en sécurité informatique, investigation numérique et biométrie.

Mon stage s'inscrit dans les travaux de l'équipe **SAFE**, spécialisée notamment dans l'investigation numérique. C'est dans ce cadre que j'ai contribué au **projet IANEC**, financé par le ministère de la Culture, visant à fournir un service numérique innovant à l'Institut Mémoires de l'édition contemporaine (**IMEC**). Cet institut conserve des archives privées nativement numériques, contenant une grande quantité de données aux formats variés.

Le projet IANEC a pour objectif d'explorer les archives numériques d'auteurs contemporains, et d'adapter les méthodes d'analyse aux besoins des institutions du patrimoine culturel. Il s'agit notamment de développer un outil d'investigation numérique spécifique, dédié à l'analyse du matériel issu de ce patrimoine.

Ces archives sont souvent issues d'ordinateurs anciens, de disques durs, voire de disquettes, qui sont confiés à l'IMEC. On y retrouve une grande variété de documents, témoignant de leurs vie personnelle et professionnelle : des lettres, des courriels, des notes, des romans ou articles en cours d'écriture, des brouillons, des listes, des carnets, des documents administratifs : factures, contrats, relevés, tableaux budgétaires, etc. Certains sont nativement numériques, d'autres sont des images ou des scans de documents manuscrits ou imprimés, plus ou moins bien cadrés. Il est donc important de distinguer ces contenus potentiellement riches d'un point de vue patrimonial, des fichiers système ou des éléments techniques qui relèvent uniquement du fonctionnement des appareils. Ces supports contiennent souvent l'environnement numérique complet d'un auteur, tel qu'il l'a laissé, avec ses habitudes, ses classements, et ses traces de travail.

Pour pouvoir organiser ces contenus, il est d'abord nécessaire de les identifier et de reconnaître les documents pertinents parmi l'ensemble des fichiers présents. Cette détection implique de croiser plusieurs approches : l'analyse des métadonnées, l'examen du contenu textuel ou visuel.

L'objectif de mon stage est d'explorer la classification automatique de documents et le traitement des informations textuelles, afin de pouvoir organiser les documents selon leur nature et leur type, de manière fiable et automatisée.

1 Classification par type

Dans un premier temps, une classification globale par type a été fondée sur des règles, en s'appuyant sur le type MIME et l'extension des fichiers. Pour cela, la commande `file` a été utilisée afin d'analyser l'extension du fichier, puis complétée avec `magic`, un outil permettant d'analyser le contenu binaire des fichiers issus de logiciels très anciens.

Grâce à cette approche, les fichiers ont été regroupés dans les grandes catégories suivantes :

Images, **Vidéos**, **Audios**, **Présentations (pptx)**, **Documents**, **Archives**, **E-mails systèmes**, **Autres fichiers systèmes** et **Autres**.

Cette première étape visait à faciliter la suite du traitement, en distinguant les documents effectivement créés par les auteurs des fichiers système ou techniques.

Pour extraire le texte et les métadonnées des documents, les modules suivants de `Apache Tika` ont été utilisés :

```
texte = parsed.get("content", "")
metadata = parsed.get("metadata", {})
```

2 OCR et traitement d'images

2.1 Tesseract et OpenCV

Dans cette partie, différentes méthodes d'extraction de texte ont été explorée à partir d'images. L'objectif était de distinguer les images contenant du texte (comme des documents scannés) de celles qui n'en contiennent pas. Pour cela, j'ai supposé que une image contenant une forte proportion de blocs textuels est potentiellement une image de type "scan".

Afin d'extraire le texte présent dans les images, j'ai utilisé Tesseract un moteur de reconnaissance optique de caractères (OCR) basé sur une analyse en plusieurs étapes, via sa bibliothèque Python pytesseract.

Pour évaluer les résultats extraits par `Tesseract OCR` une comparaison a été réalisée à l'aide de deux métriques classiques en reconnaissance de texte : **CER** (Character Error Rate) et **WER** (Word Error Rate).

Ces métriques sont basées sur la *distance de Levenshtein*, qui mesure le nombre minimal de modifications (insertions, suppressions ou substitutions) nécessaires pour transformer le texte reconnu en un texte de référence.

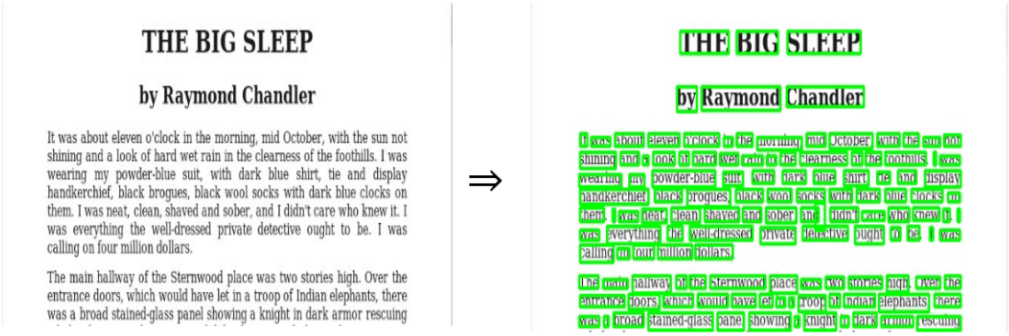
— **Character Error Rate (CER)** : cette mesure indique le pourcentage de caractères mal reconnus par l'OCR. Plus le CER est faible, plus la reconnaissance est précise au

niveau des lettres.

— **Word Error Rate (WER)** : similaire au CER, cette métrique s'applique au niveau des mots. Elle quantifie combien de mots doivent être modifiés pour que le texte OCR corresponde au texte original.

Cependant, pour obtenir des résultats satisfaisants, un prétraitement de l'image en utilisant la bibliothèque OpenCV était nécessaire.

— Conversion en niveaux de gris : `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`
— Réduction du bruit par filtre : `cv2.bilateralFilter`
— Seuillage adaptatif pour améliorer le contraste du texte
— Extraction du texte : `pytesseract.image_to_string(Image_traite)`



Texte extrait :

> *THE BIG SLEEP by Raymond Chandler ok was about eleven o'clock in the morning mid october with the sun net shining and a look of hard wet rain in the clearness of the foothills. I was wearing my powder-blue suit with dark blue shirt, tie and display handkerchief, black brogues, black wool socks with dark blue clocks on them. I was neat, clean, shaved and sober, and I didn't care who knew it. I was everything the well-dressed private detective ought to be. I was calling on four million dollars. The main hallway of the sherwood place was two stories high. Over the entrance doors, which would have let in a troop of Indian elephants, there was a broad stained-glass panel showing a knight in dark armor rescuing...*

Évaluation des erreurs :

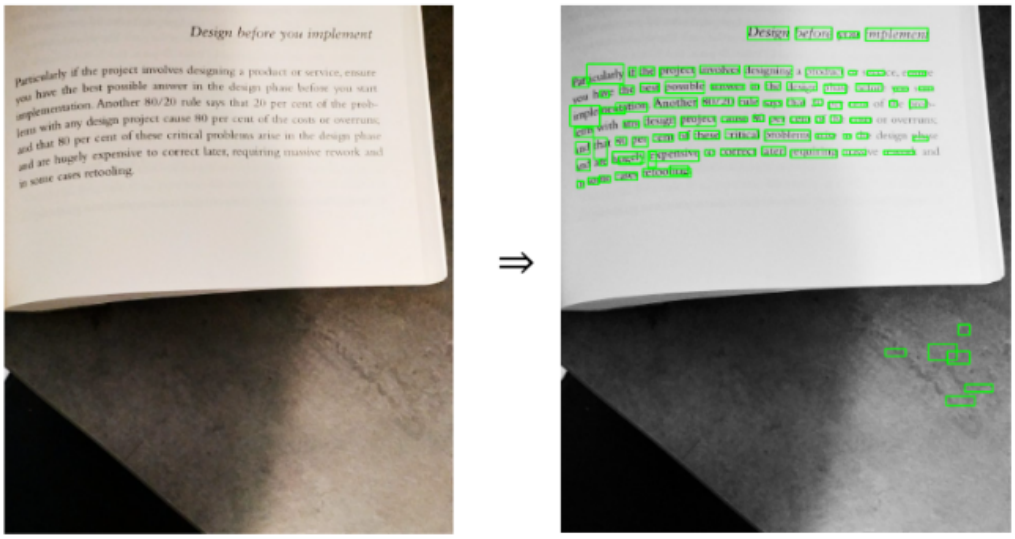
— **CER** = 0.0014 (soit seulement 0,14 % de caractères mal reconnus)
— **WER** = 0.0000 (aucune erreur de mot détectée)

2.2 Limites de Tesseract

Lorsque le texte est bien aligné et de bonne qualité, l'extraction se déroule correctement comme dans l'exemple précédent. En revanche, si le texte est incliné ou courbé, la reconnaissance devient difficile et les résultats sont souvent illisibles. J'ai tenté d'appliquer des techniques de redressement de texte :

— Recherche des plus grands contours externes : `cv2.findContours`
— Approximation du contour dominant sous forme de quadrilatère : `cv2.approxPolyDP`
— Réorganisation des coins des blocs

— Calcul de la largeur et de la hauteur via la norme euclidienne : `np.linalg.norm`
— Construction de la matrice projective : `cv2.getPerspectiveTransform`
— Application de la transformation pour redresser l'image : `cv2.warpPerspective`



Texte extrait :

Design before you implement af the protect wrvolwer dengrang a product on aerie reuse are the best possible answer in the deign phase before you sur Lad and Another 6020 rule see that 20 per cent of the probe ry egg proper aunt be per cent of the ont or onetime pro go per cent of these critical problem arne in the drug phone i lovely expensive to correct later requiring manage reek and we oo cases retooling o a rn 3 .

Évaluation des erreurs :

— **CER** = 0.3265 (soit 32,65 % de caractères mal reconnus)
— **WER** = 0.6098 (soit 60,98 % de mots incorrects)

2.3 Mistral-OCR : approche LLM

Face au problème précédent, une solution consiste à aller plus loin que la simple reconnaissance optique. L'idée est d'utiliser un modèle de langage.

Il existe déjà des modèles dans ce domaine, comme Mistral-OCR, un llm (Large Language model) conçu pour extraire du texte à partir d'images de manière plus intelligente. Contrairement aux moteurs OCR classiques comme Tesseract, qui se basent uniquement sur la reconnaissance optique des caractères et éventuellement sur des dictionnaires simples, les modèles de langage de type LLM ont la capacité de corriger automatiquement les erreurs en s'appuyant sur la sémantique contextuelle de la phrase.

Résultat extrait de l'image précédente avec Mistral :

Design before you implement

Particularly if the project involves designing a product or service, ensure you have the best possible answer in the design phase before you start implementation. Another 80/20 rule says that 20 per cent of the problems with any design project cause 80 per cent of the costs or overruns; and that 80 per cent of these critical problems arise in the design phase and are hugely expensive to correct later, requiring massive rework and in some cases retooling.

Évaluation des erreurs :

— **CER** = 0.0000 (aucun caractère mal reconnus)

— **WER** = 0.0000 (aucune erreur de mot détectée)

Pour tester le modèle sur les fonds d'archives locaux, il a été nécessaire d'installer Mistral en local afin de garantir la confidentialité des données.

3 Classification textuelle par machine learning

L'apprentissage automatique est une méthode qui permet à un modèle d'apprendre à partir de données d'entraînement pour effectuer des prédictions ou des classifications sur de nouveaux documents. Dans le cadre de ce stage, il a été utilisé pour classer automatiquement des documents selon leur contenu textuel à travers plusieurs étapes :

3.1 Définition des catégories

En analysant le contenu textuel des documents des auteurs, nous avons définies 5 catégories principales :

— **Administratifs** : factures, documents bancaires, formulaires, attestations, budgets, contrats.

— **Correspondance** : lettres, e-mails.

— **Journaux intimes** : réflexions personnelles.

— **Activité d'écriture** : brouillons d'articles, suites de livres, poèmes.

— **Vie privée** : listes de courses, plannings de voyage.

3.2 Collecte de données

Une fois cette catégorisation établie, la prochaine étape était constituer un jeu de données suffisant pour entraîner un modèle de machine learning. Pour limiter le risque de surapprentissage, il fallait au moins 200 documents par catégorie.

La collecte de données externes s'est avérée chronophage. La recherche a été effectuée sur plusieurs sources, notamment Kaggle et GitHub. Dans certains cas, comme pour les catégories *journal intime* et *vie privée*, des documents artificiels ont été générés à partir d'éléments observés dans les archives.

Certaines données étaient rédigées en anglais. Une traduction automatique a alors été effectuée à l'aide de Google Translate, via la bibliothèque Python `deep_translator`.

D'autres documents du dataset étaient au format image : dans ce cas, *Mistral-OCR* a permis d'extraire le texte afin de construire un jeu de données au format CSV.

3.3 Prétraitement des textes

Dans cette étape, différentes techniques issues du traitement automatique du langage naturel (NLP) ont été appliquées afin de transformer les documents textuels bruts en données exploitables pour l'entraînement du modèle Machine Learning.

3.3.1 Nettoyage du texte

Avant l'entraînement du modèle, un prétraitement des données était nécessaire. Les mêmes méthodes ont été appliquées aussi bien aux données collectées qu'aux données à prédire, afin d'assurer la cohérence du traitement. Dans la fonction de nettoyage,le texte a été limité aux 2 000 premiers mots, ce qui s'est avéré suffisant dans notre cas. Ensuite, seuls les caractères accentués, certains signes de ponctuation (., „ :, /, ?, !), les chiffres et les symboles monétaires ont été conservés. Enfin, les sauts de ligne ainsi que les espaces multiples ont été supprimés, afin d'uniformiser le texte.

3.3.2 Lemmatisation

Parmi les étapes clés du prétraitement, la lemmatisation. Contrairement au stemming, qui se limite à supprimer de manière heuristique les terminaisons des mots pour en extraire la racine (ex. : trouverez → trouv), la lemmatisation repose sur une analyse linguistique approfondie. Elle permet d'isoler la forme canonique du mot (appelée lemme) tout en préservant sa signification sémantique. Par exemple, trouvez devient trouvé.

Cette approche a été privilégiée car elle permet de conserver le sens du texte, ce qui est essentiel pour notre tâche de classification de documents. Elle présente également l'avantage de neutraliser les variations liées au genre et au nombre , ce qui simplifie notamment le traitement lors de l'extraction des entités nommées.

3.3.3 POS-tagging et enrichissement thématique

Une première extraction est réalisée avec le modèle NER de `spaCy`, pour identifier les entités de type personne (PER), organisation (ORG) et lieu (LOC). Une seconde extraction s'appuie sur des listes lexicales organisées par thématique. Ces listes ajoutent des indicateurs linguistiques utiles à la classification.

Les principales thématiques utilisées sont les suivantes :

— `mots_subjectifs` : indicateurs de subjectivité, présents dans les journaux intimes.
— `mots_politesse` : marqueurs de formalisme (lettres, e-mails).
— `mots_facture`, `mots_comptabilité`, `mots_RH`, `mots_administratif`, `mots_médical`, `mots_santé`, `mots_RDV` : vocabulaire typique de documents administratifs.
— `mots_couleurs`, `mots_vêtements`, `mots_maison`, `mots_immeuble`, `mots_vacances`, `liste_documents` : lexique orienté vers la vie privée (listes de courses).

```
vectorizer_text = TfidfVectorizer(max_features=1000)
X_text = vectorizer_text.fit_transform(df[colonne_texte].fillna("")).toarray()

vectorizer_pos = TfidfVectorizer(max_features=17)
X_pos = vectorizer_pos.fit_transform(df[colonne_gramm].fillna("")).toarray()

scaler = StandardScaler()
X_num = scaler.fit_transform(df[colonnes_numeriques].fillna(0))
```

— `verbes_actions` : pour repérer des phrases actives, utiles pour l'activité d'écriture.

Chaque document est analysé selon la présence de ces groupes de mots, ce qui permet d'extraire des *features* thématiques enrichissant la représentation textuelle en entrée du modèle de classification.

Après la lemmatisation du texte et des listes d'entités, une forme de matching lexical a été appliquée pour identifier les entités présentes dans chaque document. Chaque mot détecté dans les listes est alors remplacé par un tag correspondant à sa catégorie ex : @subjectif, @politesse, etc, ce qui a permis d'obtenir un texte enrichi et lemmatisé. Dans une deuxième étape, chaque mot du texte est remplacé par sa catégorie grammaticale *POS-tag*, afin d'intégrer des informations syntaxiques.

Enfin, pour chaque type d'entité ou de tag, le nombre d'occurrences dans le texte est comptabilisé, puis un pourcentage de présence est calculé « nombre de tags rapporté au nombre total de mots ». Chaque pourcentage ainsi obtenu constitue une variable numérique *features* utilisée dans l'apprentissage.

3.4 Vectorisation TF-IDF et normalisation

Afin de pouvoir appliquer les méthodes de Machine Learning aux problèmes relatifs au langage naturel, il est indispensable de transformer les données textuelles en données numériques.

La méthode TF/IDF (Term Frequency-Inverse Document Frequency) a été appliquée : cette méthode consiste à compter le nombre d'occurrences des tokens présents dans le corpus pour chaque texte, que l'on divise ensuite par le nombre d'occurrences total de ces mêmes tokens dans tout le corpus.

La vectorisation TF-IDF a été appliquée sur deux types de texte :
— Pour le texte lemmatisé, les 1 000 mots les plus fréquents ont été conservés.
— Pour la représentation grammaticale, seulement les 17 plus fréquents, car elle est moins représentative que le texte lemmatisé.

Les features numériques étant déjà des valeurs quantitatives, un simple `scaling` avec `StandardScaler` a été appliqué pour les normaliser.

3.5 Évaluation des modèles

Pour la classification automatique des documents textuels, le choix s'est porté sur le modèle SVM (Support Vector Machine), reconnu pour sa robustesse dans les tâches de classification textuelle. Avant d'évaluer les performances de notre modèle, il est essentiel de comprendre les différents cas possibles lors de la comparaison entre les prédictions du modèle et les valeurs réelles. quatre cas sont généralement distingués :

Exactitude (Accuracy)

Elle représente le pourcentage de prédictions correctes (positives ou négatives) parmi l'ensemble des prédictions réalisées par le modèle.Pas fiables si les classes sont déséquilibrées.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Précision

permet de mesurer la fiabilité des prédictions positives du modèle. En d'autres termes, elle indique le pourcentage de documents correctement classés parmi ceux que le modèle a prédits comme appartenant à une certaine catégorie.

$$\text{Précision} = \frac{TP}{TP + FP}$$

On privilégie la précision lorsque les faux positifs sont plus problématiques que les faux négatifs. *Exemple :* si le modèle prédit que 100 documents sont des lettres, mais que seulement 80 le sont vraiment, alors la précision est de 80 %.

Rappel

Mesure la capacité du modèle à retrouver tous les documents réellement appartenant à une catégorie donnée. Il correspond au taux de vrais positifs détectés parmi tous les vrais positifs existants.

$$\text{Rappel} = \frac{TP}{TP + FN}$$

On privilégie le rappel lorsqu'il est important de ne pas passer à côté d'un cas positif. *Exemple :* si 100 documents sont réellement des lettres, mais que le modèle n'en détecte que 70, alors le rappel est de 70 %.

F1-score

La moyenne harmonique entre Précision et Rappel. Il équilibre les faux positifs et les faux négatifs.

$$F1 = 2 \cdot \frac{\text{Précision} \cdot \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

Il est utile pour classifier des documents avec peu d'erreurs, notamment lorsqu'on ne souhaite

ni rater une catégorie importante, ni surclasser des documents.

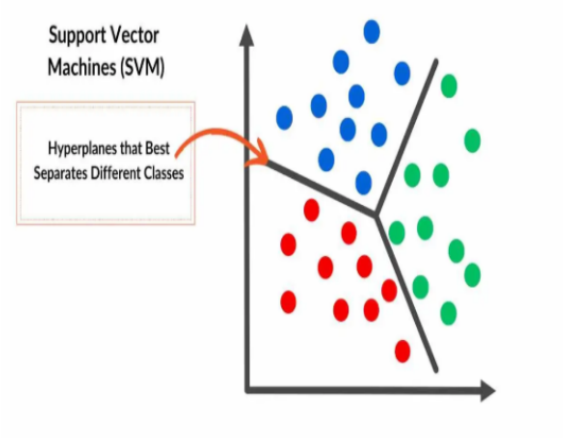
Exemple : le rappel est de 70 % si 100 documents sont réellement des journaux intimes, mais que le modèle n'en détecte que 70.

Évaluation comparative des modèles

Quatre modèles ont été choisis pour effectuer une évaluation.

SVM (Support Vector Machine) :

Pour la classification, le modèle SVC (Support Vector Classification) a été utilisé. Il repose sur le principe de séparation des classes à l'aide d'un hyperplan maximalement séparateur. SVC permet l'utilisation de différents noyaux, et dans ce projet, un noyau linéaire a été retenu pour sa simplicité et son efficacité sur des données textuelles.



Régression logistique :

La régression logistique est un modèle de classification qui estime la probabilité qu'un échantillon appartienne à une classe. Elle repose sur une fonction sigmoïde pour transformer une combinaison linéaire des variables en une probabilité comprise entre 0 et 1.

Random Forest :

Le modèle Random Forest repose sur un ensemble d'arbres de décision construits à partir d'échantillons aléatoires du jeu de données. Chaque arbre donne une prédiction, et la décision finale est obtenue par vote majoritaire.

XGBoost :

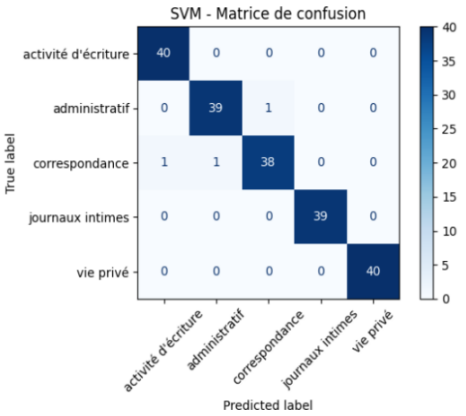
XGBoost (Extreme Gradient Boosting) est un algorithme d'ensemble basé sur des arbres de décision. Il construit les arbres de manière séquentielle, chaque nouvel arbre corrigeant les erreurs des précédents.

Évaluation des performances des modèles :

=== SVM ===

Classification report :

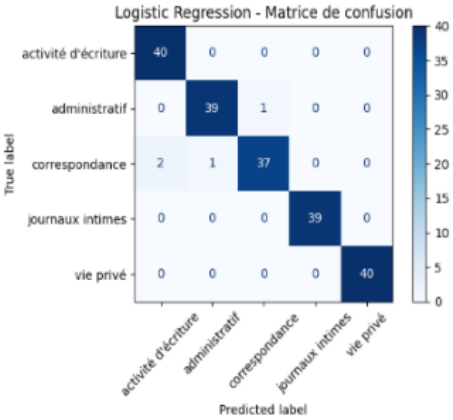
	precision	recall	f1-score	support
activité d'écriture	0.98	1.00	0.99	40
administratif	0.97	0.97	0.97	40
correspondance	0.97	0.95	0.96	40
journaux intimes	1.00	1.00	1.00	39
vie privée	1.00	1.00	1.00	40
accuracy			**0.98**	199
macro avg	0.98	0.98	0.98	199
weighted avg	0.98	0.98	0.98	199



=== Régression Logistique ===

Classification report :

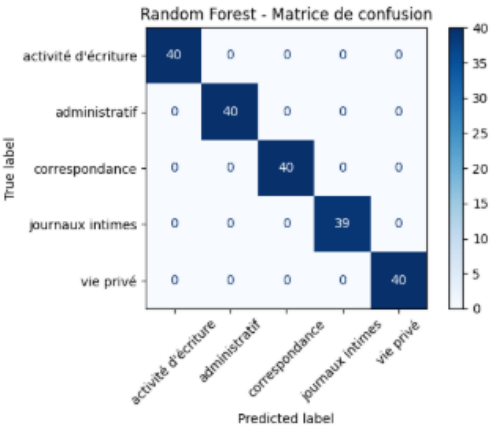
	precision	recall	f1-score	support
activité d'écriture	0.95	1.00	0.98	40
administratif	0.97	0.97	0.97	40
correspondance	0.97	0.93	0.95	40
journaux intimes	1.00	1.00	1.00	39
vie privée	1.00	1.00	1.00	40
accuracy			**0.98**	199
macro avg	0.98	0.98	0.98	199
weighted avg	0.98	0.98	0.98	199



=== Random Forest ===

Classification report :

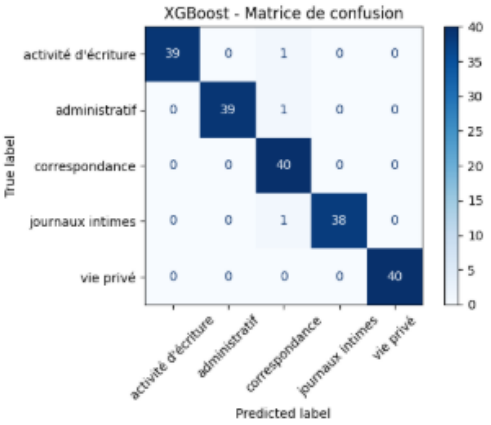
	precision	recall	f1-score	support
activité d'écriture	1.00	1.00	1.00	40
administratif	1.00	1.00	1.00	40
correspondance	1.00	1.00	1.00	40
journaux intimes	1.00	1.00	1.00	39
vie privée	1.00	1.00	1.00	40
accuracy			**1.00**	199
macro avg	1.00	1.00	1.00	199
weighted avg	1.00	1.00	1.00	199



=== **XGBoost** ===

Classification report :

	precision	recall	f1-score	support
activité d'écriture	1.00	0.97	0.99	40
administratif	1.00	0.97	0.99	40
correspondance	0.93	1.00	0.96	40
journaux intimes	1.00	0.97	0.99	39
vie privée	1.00	1.00	1.00	40
accuracy			**0.98**	199
macro avg	0.99	0.98	0.99	199
weighted avg	0.99	0.98	0.99	199



Résultats des prédictions :

Afin de tester le modèle sur des données réelles issues des archives d'auteurs, un ensemble de fichiers a été trié et organisé dans des dossiers, chacun correspondant à une catégorie réelle attribuée manuellement. Le modèle a ensuite été appliqué sur cet ensemble pour évaluer sa capacité de prédiction. Les résultats obtenus sont présentés dans le tableau ci-dessous.

Classe réelle	Classe prédite	Fichiers	Pourcentage
Administratif	Administratif	4	66.7%
Administratif	Activité d'écriture	1	16.7%
Administratif	Correspondance	1	16.7%
Activité d'écriture	Activité d'écriture	45	91.8%
Activité d'écriture	Administratif	3	6.1%
Activité d'écriture	Correspondance	1	2.0%
Correspondance	Correspondance	22	88.0%
Correspondance	Activité d'écriture	3	12.0%
Journaux intimes	Activité d'écriture	5	83.3%
Journaux intimes	Correspondance	1	16.7%
Vie privée	Vie privée	9	17.6%
Vie privée	Administratif	27	52.9%
Vie privée	Correspondance	15	29.4%

Analyse des résultats

Les performances initiales de la classification sur cinq catégories se révèlent satisfaisantes, avec des scores de précision, rappel et F1 atteignant ou dépassant les 95 % pour l'ensemble des modèles testés.

Cependant, une analyse qualitative des erreurs a révélé des confusions récurrentes entre certaines classes sémantiquement proches. Ce **chevauchement sémantique** se manifeste notamment dans trois situations :

— Entre *Activité d'écriture* et *Journaux intimes* : certains documents contiennent des

brouillons et des suites de romans dans lesquels l'auteur évoque des épisodes personnels marquants. Cette écriture introspective, parfois très proche du journal intime, brouille la frontière entre les deux classes. Une autrice en particulier avait intitulé un livre "MOI", plusieurs autres fichiers mêlaient récit et souvenirs personnels au sein d'un même texte.

— Entre *Vie privée* et *Administratif* : des documents comme des listes de courses ou des plannings de voyage incluent souvent des chiffres (dates, quantités, budgets). Ces occurrences numériques élevées ont parfois été interprétées par le modèle comme indicateurs de la classe "Administratif", où la densité numérique est également forte (factures, contrats, etc.).

— De manière plus générale, la nature des archives, très personnelle et hétérogène selon les auteurs, induit une **variabilité linguistique importante**.

Ces observations soulignent que, malgré de bons scores globaux, certaines ambiguïtés inhérentes aux contenus textuels empêchent une séparation nette entre les classes. Elles justifient la décision de réduire le nombre de catégories et d'adapter le modèle à une typologie plus robuste et adaptée aux archives étudiées.

Réajustements du modèle

Une reconfiguration du jeu de données a été réalisée, comprenant :

— Une **réduction du nombre de classes à trois**, en fusionnant ou supprimant les catégories les moins discriminantes.
— Un **filtrage des documents trop courts**, afin d'éliminer les bruits.
— Un **enrichissement ciblé** des classes conservées, à l'aide de documents propres aux auteurs, complété par des données externes.

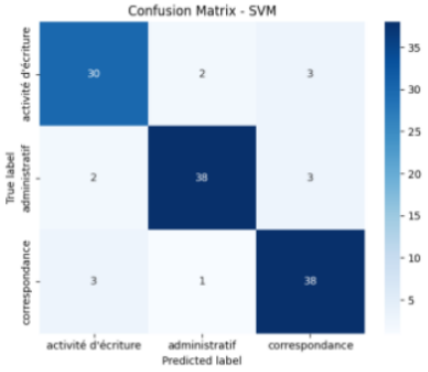
Enfin, à la suite de tests comparatifs, le modèle SVM a été remplacé par **XGBoost**, un algorithme de gradient boosting plus flexible. Il élabore une suite d'arbres de décision, et chacun de ces arbres s'évertue à corriger les inexactitudes ou imperfections du précédent. Ainsi, à chaque étape, les prédictions deviennent de plus en plus précises, particulièrement efficaces pour combiner des données textuelles vectorisées avec des variables numériques.

Ce changement a permis d'obtenir des résultats plus robustes, notamment sur les documents complexes et les textes aux structures moins conventionnelles.

=== **SVM** ===

Classification report :

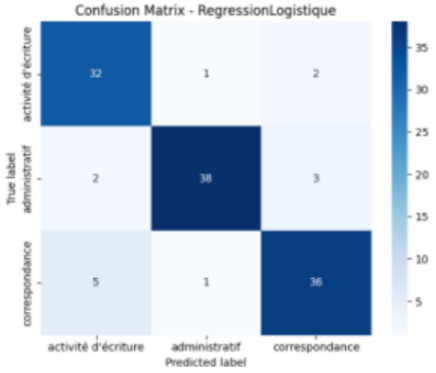
	precision	recall	f1-score	support
activité d'écriture	0.86	0.86	0.86	35
administratif	0.93	0.88	0.90	43
correspondance	0.86	0.90	0.88	42
accuracy			**0.88**	120
macro avg	0.88	0.88	0.88	120
weighted avg	0.88	0.88	0.88	120



=== **Régression Logistique** ===

Classification report :

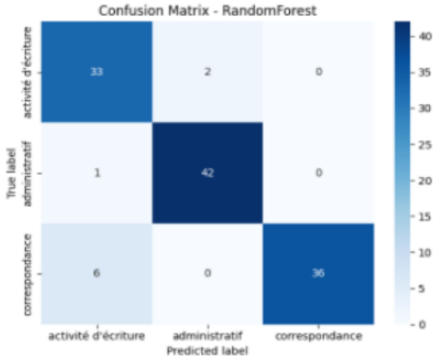
	precision	recall	f1-score	support
activité d'écriture	0.82	0.91	0.86	35
administratif	0.95	0.88	0.92	43
correspondance	0.88	0.86	0.87	42
accuracy			**0.88**	120
macro avg	0.88	0.89	0.88	120
weighted avg	0.89	0.88	0.88	120



=== **Random Forest** ===

Classification report :

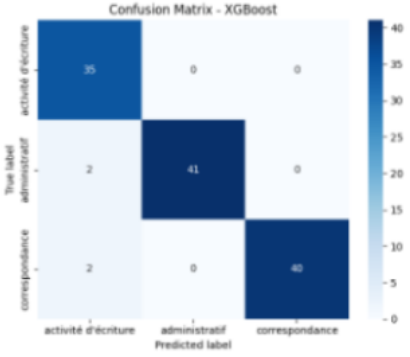
	precision	recall	f1-score	support
activité d'écriture	0.82	0.94	0.88	35
administratif	0.95	0.98	0.97	43
correspondance	1.00	0.86	0.92	42
accuracy			**0.93**	120
macro avg	0.93	0.93	0.92	120
weighted avg	0.93	0.93	0.93	120



=== **XGBoost** ===

Classification report :

	precision	recall	f1-score	support
activité d'écriture	0.90	1.00	0.95	35
administratif	1.00	0.95	0.98	43
correspondance	1.00	0.95	0.98	42
accuracy			**0.97**	120
macro avg	0.97	0.97	0.97	120
weighted avg	0.97	0.97	0.97	120



Résultats des prédictions :

Sur un échantillon de 20 documents correctement définis dans chaque catégorie, une prédiction a été effectuée à l'aide du modèle. Les résultats sont présentés dans le tableau ci-dessous.

Classe réelle	Classe prédite	Fichiers	Pourcentage
Administratif	Administratif	20	100.0%
Activité d'écriture	Activité d'écriture	20	100.0%
Correspondance	Activité d'écriture	3	15.0%
Correspondance	Correspondance	17	85.0%

4 Fine-tuning léger Mistral pour une classification textuelle

Dans le cadre de l'exploration des modèles de langage (LLM), une technique de spécialisation, le *fine-tuning*, a été testée. Elle permet d'adapter un modèle pré-entraîné à une tâche spécifique, comme la classification de documents. L'objectif du fine-tuning est de réentraîner partiellement le modèle sur un jeu de données annoté, tout en préservant ses connaissances générales. Cela permet au modèle de mieux répondre à des cas particuliers, sans repartir de zéro.

Approche utilisée Pour des raisons de performances, un fine-tuning léger avec LoRA a été utilisé sur le modèle `Mistral-7B-Instruct-v0.3`, car le réentraînement complet est extrêmement coûteux en termes de temps : il durerait plusieurs semaines, voire mois, car le modèle possède 7 milliards de paramètres à réentraîner.

4.1 La méthode LoRA (Low-Rank Adaptation) :

Cette technique consiste à injecter de petites matrices entraînables dans les couches internes du modèle (ici les projections `q_proj` et `v_proj`) sans modifier les poids principaux. Cela permet : une réduction significative du nombre de paramètres ajustés, une exécution plus rapide et une meilleure compatibilité avec des ressources matérielles. Le modèle choisi est chargé en mode quantifié 4 bits (`nf4`), entraîné à l'aide du module `SFTTrainer` de `trl`.

Le dataset utilisé est un fichier `.jsonl` contenant des paires texte / catégorie. Chaque exemple a été formaté selon un schéma instructionnel de type :

```
### Instruction:
Classifie le texte suivant selon sa catégorie.

### Input:
[Texte du document]
```

```
### Response:
[Catégorie cible]
```

Trois catégories ont été définies : administratif, activité d'écriture, correspondance.

```
Taille du batch : 1 (avec accumulation de gradient pour simuler un
   batch plus large)
Nombre d'epoques : 5
Optimiseur : paged_adamw_32bit
Strategie de sauvegarde : par epoque
Tokenizer : AutoTokenizer avec padding sur eos_token
Separation des jeux : 80 % entrainement, 10 % validation, 10 % test
```

L'entraînement a été réalisé sur GPU avec la configuration suivante :

```
lora_config = LoraConfig(
r=64,
lora_alpha=128,
target_modules=["q_proj", "v_proj"],
lora_dropout=0.05,
bias="none",
task_type=TaskType.CAUSAL_LM )
```

4.2 Analyse des performances

	precision	recall	f1-score	support
administratif	0.99	0.83	0.90	170
activité d'écriture	0.89	0.98	0.93	196
correspondance	0.95	0.99	0.97	198
accuracy			0.94	568
macro avg	0.94	0.93	0.94	568
weighted avg	0.94	0.94	0.94	568

Le modèle Mistral, après un fine-tuning léger avec la méthode LoRA, atteint une précision globale de 94,% sur les données de test. Ce résultat est encourageant, compte tenu de la légèreté de l'entraînement, la faible taille des matrices ajustées et l'absence de modification directe des poids du modèle d'origine.

La classe "administratif" montre un rappel plus bas (83,%), ce qui signifie que plusieurs documents de cette catégorie ont été mal classés.

Une analyse qualitative montre que cette confusion vient en grande partie de la proximité linguistique entre certains documents administratifs (factures, attestations, déclarations, contrats) et la correspondance professionnelle. Ces documents partagent souvent des formules standards. Le modèle de base, qui a été exposé à un vaste corpus généraliste, tend naturellement à interpréter ce type de structure comme de la "correspondance" – et ce biais n'a pas été entièrement corrigé par le fine-tuning léger.

Cela s'explique par la nature même de LoRA : en injectant uniquement de petites matrices d'ajustement dans certaines couches spécifiques, on ne modifie pas la majeure partie des 7 milliards de paramètres du modèle. Ainsi, lors de l'inférence, les prédictions du modèle s'appuie encore majoritairement sur ses connaissances d'origine. Les poids ajustés par LoRA n'ont qu'une influence marginale – surtout dans une tâche aussi fine que la classification textuelle.

Malgré cela, les scores obtenus sur l'ensemble des classes sont bons, avec un équilibre satisfaisant entre précision, rappel et F1-score. Ce résultat démontre que même une adaptation légère d'un LLM comme Mistral peut donner lieu à de bonnes performances.

5 Machine Learning VS Fine-tuning léger pour une classification textuelle

Les modèles classiques comme SVM ou XGBoost se sont montrés particulièrement performants et fiables. Une fois entraînés, ils adoptent un comportement déterministe : un document donné produira toujours la même prédiction. Cette stabilité est un atout précieux dans un contexte structuré, où la cohérence et la reproductibilité des résultats sont essentielles. Les scores obtenus " $F1 > 98\ \%$ " témoignent de leur efficacité sur notre jeu de données enrichi et bien prétraité.

À l'inverse, le modèle fine-tuné Mistral conserve des caractéristiques propres aux LLM : il reste un modèle instructionnel, sensible à la formulation des prompts et aux variations contextuelles. Il peut générer des réponses différentes à partir d'un même texte, et est donc plus exposé aux hallucinations ou erreurs d'interprétation. Cela en fait un outil puissant mais plus difficile à contrôler finement et dépend de la précision et la qualité du prompt.

Ainsi, même si le modèle Mistral fine-tuné est prometteur pour des cas plus ouverts ou créatifs, les méthodes classiques de machine learning se montrent plus efficaces et fiables dans un contexte structuré et orienté vers la précision.

6 Classification d'images avec Pixtral

En complément de l'analyse textuelle. L'analyse a été étendue aux documents en format image. Cette exploration a conduit à découvrir PixTral-12B, un modèle de type LLM multimodal capable de traiter à la fois des images et du texte.

Dans les usages classiques des LLM, le modèle est généralement chargé directement dans un script Python. Toutefois, PixTral-12B contient environ 400 milliards de paramètres, ce qui rend son chargement extrêmement lourd. Il nécessite au minimum deux GPU pour être entièrement chargé en mémoire.

Pour contourner cette contrainte technique, j'ai opté pour l'utilisation de vLLM (Virtual Large Language Model), une architecture optimisée pour le déploiement de modèles LLM à grande échelle.

6.1 Déploiement avec vLLM

vLLM repose sur une **architecture client/serveur** qui permet de charger le modèle une seule fois, côté serveur, puis de traiter plusieurs requêtes de manière **asynchrone et continue**. Ce système est particulièrement efficace pour gérer de gros volumes de données (dans notre cas, un lot important d'images) tout en assurant des temps de réponse rapides.

Le fonctionnement est le suivant :

— Le **serveur** charge une fois pour toutes le modèle **PixTral-12B** en mémoire GPU.
— Les **clients** envoient des requêtes via une **API locale HTTP**.
— Le serveur traite ces requêtes **par lots** et en parallèle, en utilisant des techniques de **cache clé-valeur** qui réduisent considérablement le temps de traitement.

Cette approche est particulièrement efficace pour :

— Servir un très grand nombre d'images ou de prompts.
— Gérer des traitements longs en minimisant le coût de rechargement du modèle.
— Obtenir des réponses rapides sans redémarrer le modèle à chaque appel.

6.2 Prompt engineering pour la classification

Le prompt engineering consiste à rédiger des instructions claires pour un modèle, afin qu'il génère systématiquement des résultats précis. Comme le contenu produit par le modèle est non déterministe, il faut trouver un équilibre entre technique et créativité pour que les instructions conduisent à des réponses conformes aux attentes.

Classification globale en trois catégories

Un ***prompt* de type instruction** a été rédigé pour demander au modèle de classer chaque image selon trois catégories principales :

— **document** : contenant un texte structuré (ex : formulaire, page de livre),
— **photo** : image de scène, objet ou personne, même avec du texte non structuré,
— **icône** : image stylisée représentant une fonction ou un logo.

Le modèle devait répondre par un **mot unique**, sans justification.

Affinage pour les documents

Lorsque l'image était classée comme **document**, un second *prompt* était envoyé afin de préciser le type de contenu textuel :

— **manuscrit** : texte écrit à la main,
— **imprimé** : texte typographié.

Ce processus en deux étapes permet d'éviter une classification directe en quatre classes, qui pourrait induire des hallucinations du modèle. En effet, la diversité des contenus et des formes textuelles rend la tâche plus complexe. L'usage de *prompts* simplifiés et successifs permet de réduire ce risque, tout en respectant les contraintes du modèle, notamment en termes de longueur maximale des instructions.

6.3 Préparation des images pour PixTral

Avant d'envoyer une image au modèle Pixtral, quelques étapes simples sont nécessaires. L'image est d'abord **redimensionnée** pour rester légère et rapide à traiter. Elle est ensuite **convertie en base64**, un format texte qui permet de l'intégrer facilement dans une requête HTTP. Cette requête est envoyée au serveur local via l'API de vLLM. En retour, **Pixtral génère une réponse au format JSON**, contenant la catégorie de l'image ou une description en faisant l'analyse visuel de l'image ou le résultat de l'ocr pour extraire et sauvegarder le texte extrait de cette image, selon la fonction appelée.

7 Exploration Complémentaire

Dans cette partie, je présente quelques pistes explorées durant mon stage, mais qui n'ont finalement pas été retenues dans l'approche finale pour diverses raisons, notamment leur complexité, leur manque d'efficacité ou leur pertinence limitée au regard des résultats obtenus.

7.1 Extract Thinker :

ExtractThinker est un outil permettant d'extraire des caractéristiques (features) à partir de documents, en utilisant des modèles de langage. Utile pour extraire des features spécifiques des textes dans le dataset, afin de faciliter une classification plus précise en fonction du vocabulaire des documents.

Cependant, cet outil n'a pas été intégré en raison de deux contraintes majeures. D'une part, l'API d'ExtractThinker est compatible uniquement avec des modèles OpenAI, comme GPT-4, et non avec Mistral, le modèle utilisé localement. Les tentatives d'adaptation à Mistral ont rencontré des problèmes de compatibilité. D'autre part, le dataset contenait des informa-

tions sensibles issues des archives d'auteurs, ce qui a soulevé des préoccupations concernant la confidentialité des données car les Modèles GPT ne sont pas open source.

7.2 Embedding des Features :

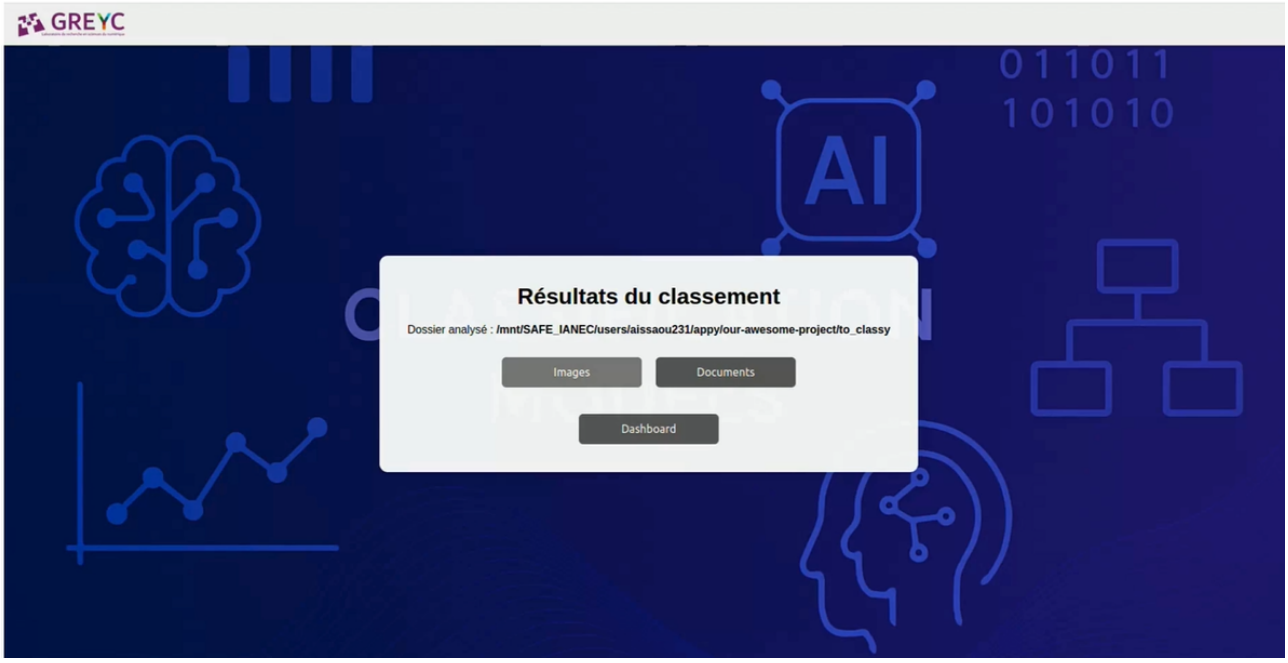
Une autre approche de vectorisation a été explorée, consistant à utiliser l'embedding des features en calculant le pourcentage de similarité sémantique entre les mots. Cette méthode permet de représenter les mots sous forme de vecteurs dans un espace continu, facilitant ainsi l'évaluation de la similarité sémantique entre les termes et l'extraction de relations plus fines entre les données.

7.3 Extraction des Features avec Mistral :

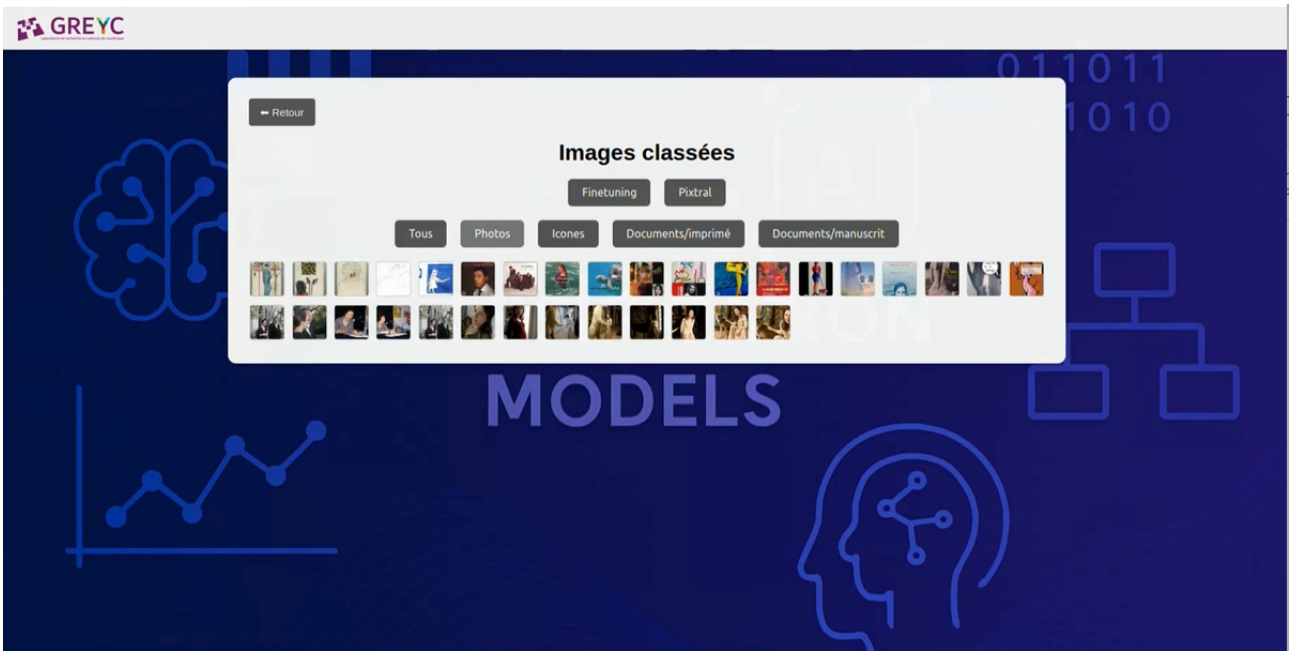
Dans cette approche, une tentative a été réalisée pour extraire les features en utilisant un prompt avec le modèle Mistral. Les résultats n'ont pas pleinement répondu aux attentes en raison du nombre élevé de features à extraire. Le modèle a tendance à générer des réponses plus longues que prévu, incluant parfois des explications supplémentaires non nécessaires. Cette caractéristique est liée à la nature du modèle Mistral, qui, en tant que modèle instructif, produit des réponses détaillées, mais qui, dans ce cas, n'était pas optimisée pour la tâche d'extraction de features.

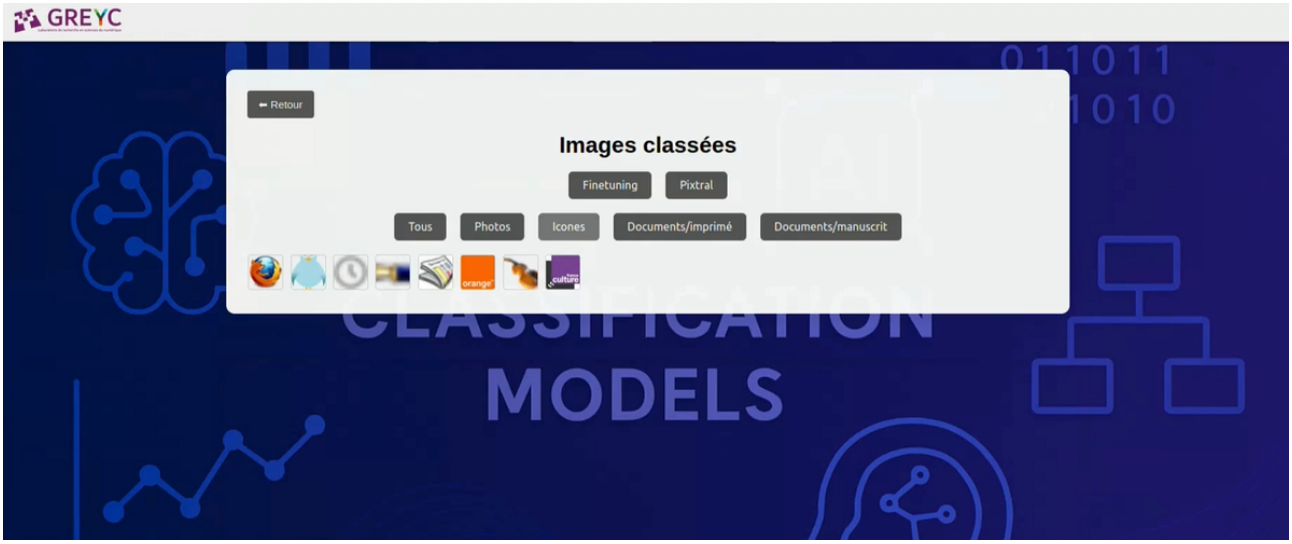
8 Plateforme de classification

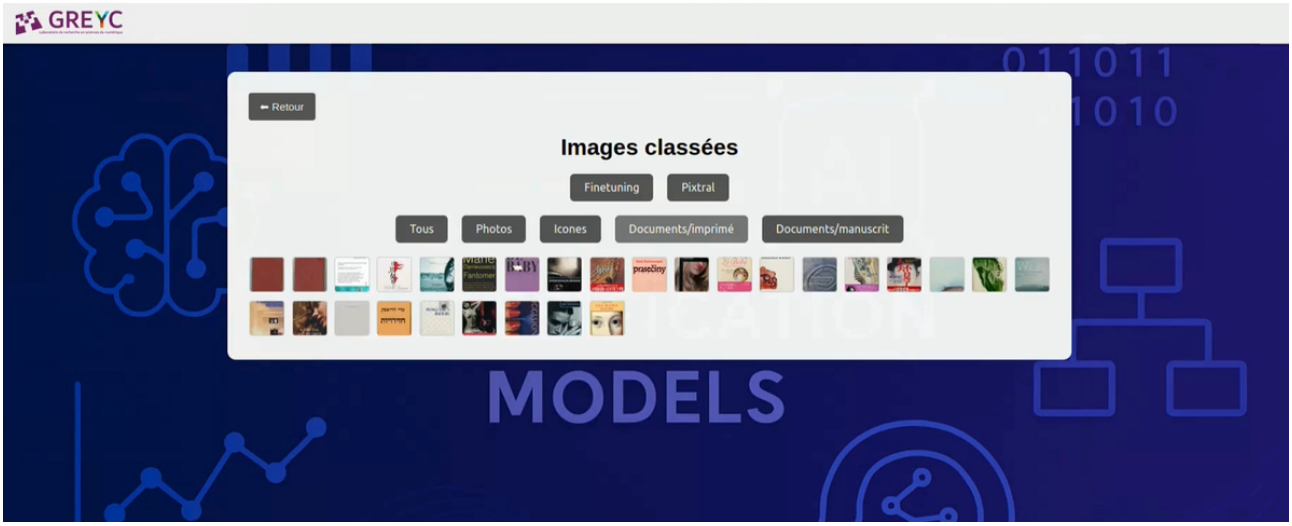
Vers la fin du stage, une plateforme de classification a été développée pour tester les modèles sur des fonds d'archives d'auteurs, comprenant deux types de classification : textuelle et visuelle. Après avoir effectué la classification globale et extrait les documents et images, une classification spécifique a été appliquée. La solution Pixtral a été intégrée pour le traitement OCR, la génération de descriptions et la classification des images. Parallèlement, un modèle de machine learning a été intégré pour la classification textuelle, permettant de prédire la catégorie des fichiers fournis en entrée.

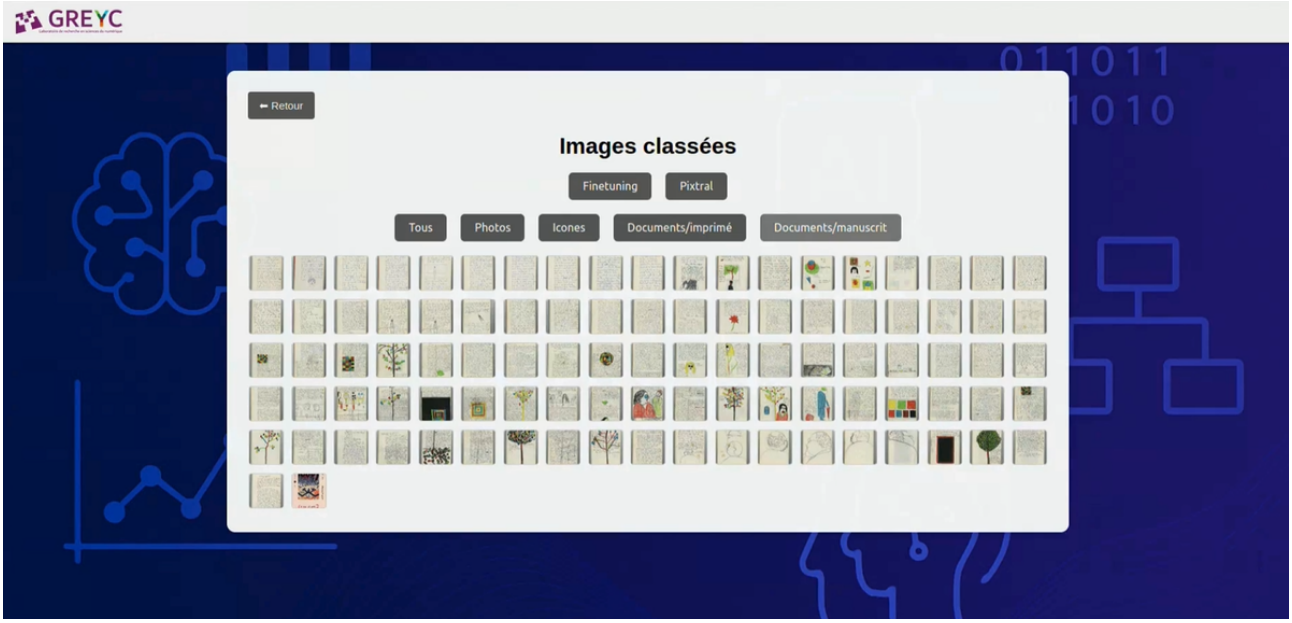


8.1 Classification Visuelle avec Pixtral

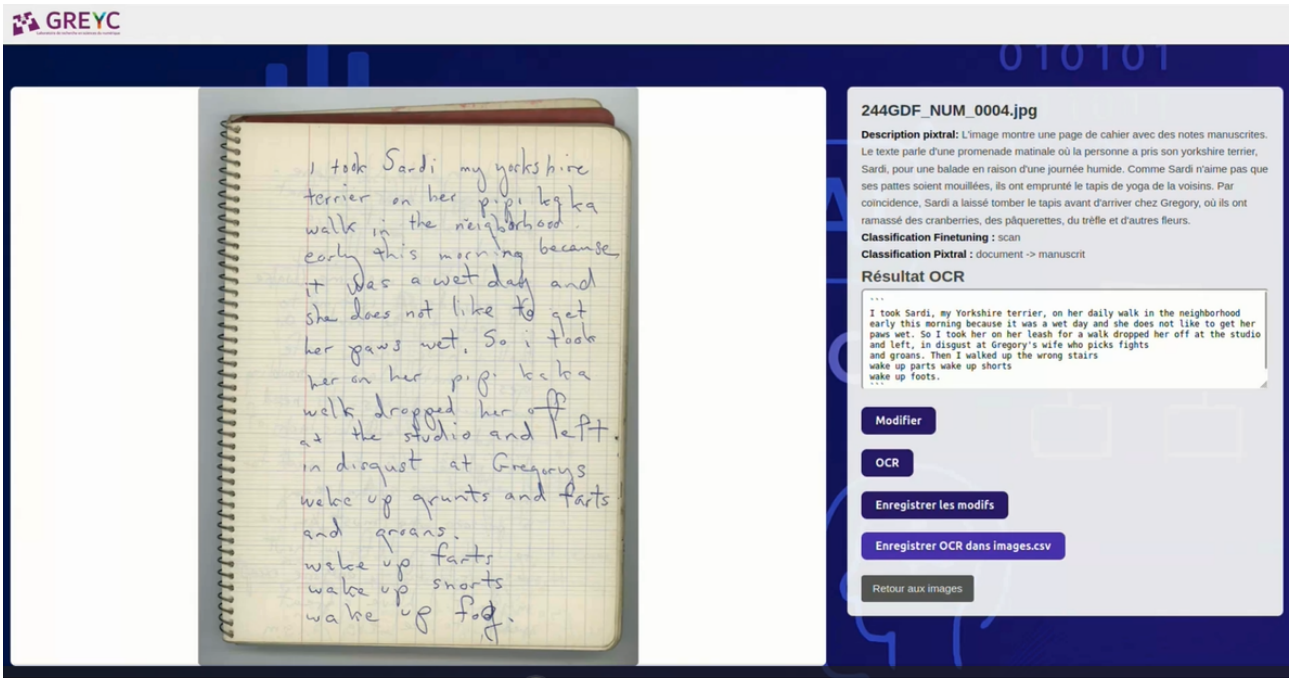



GREYC
Retour
Images classées
Finetuning
Pixtral
Tous
Photos
Icones
Documents/imprimé
Documents/manuscrit
MODELS

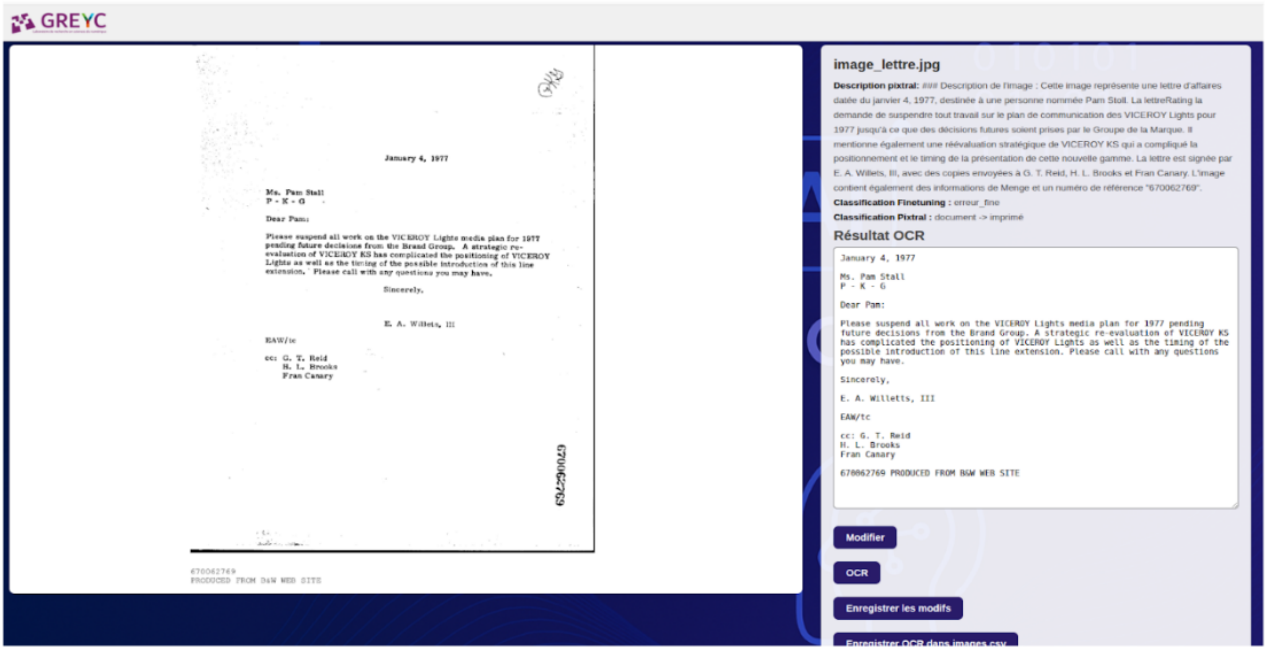

GREYC
Retour
Images classées
Finetuning
Pixtral
Tous
Photos
Icones
Documents/imprimé
Documents/manuscrit
MODELS


GREYC
Retour
Images classées
Finetuning
Pixtral
Tous
Photos
Icones
Documents/imprimé
Documents/manuscrit

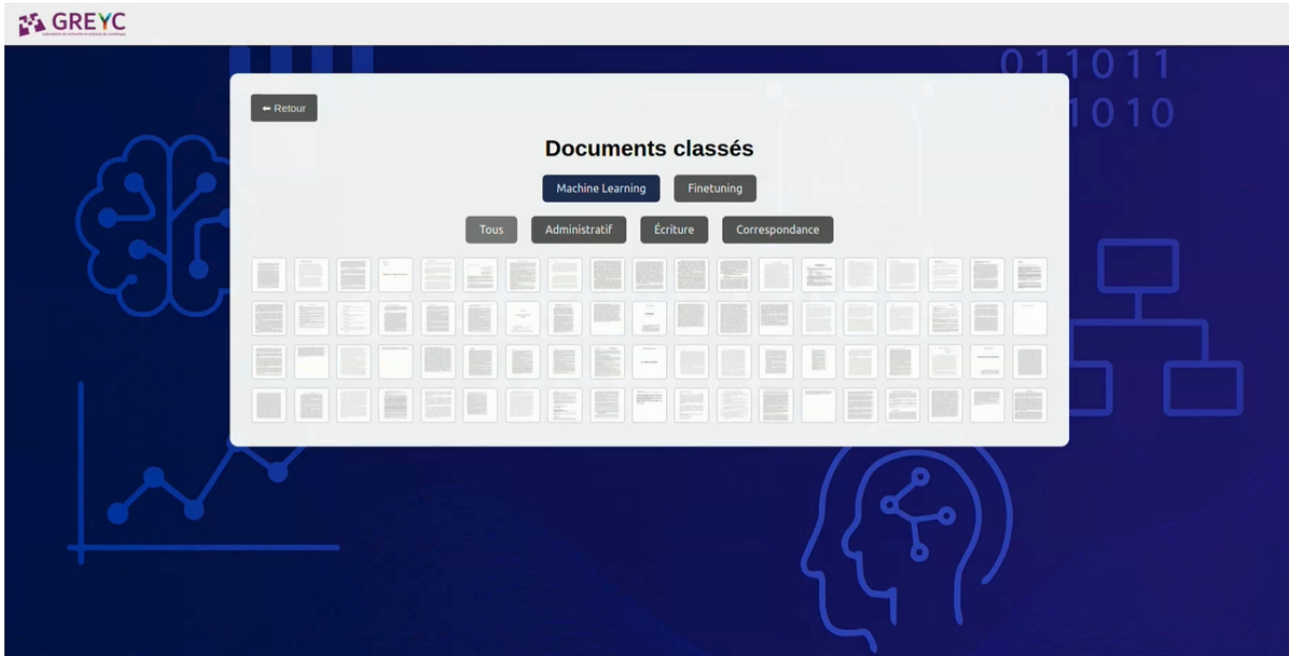
8.2 Résultat OCR document Manuscrit

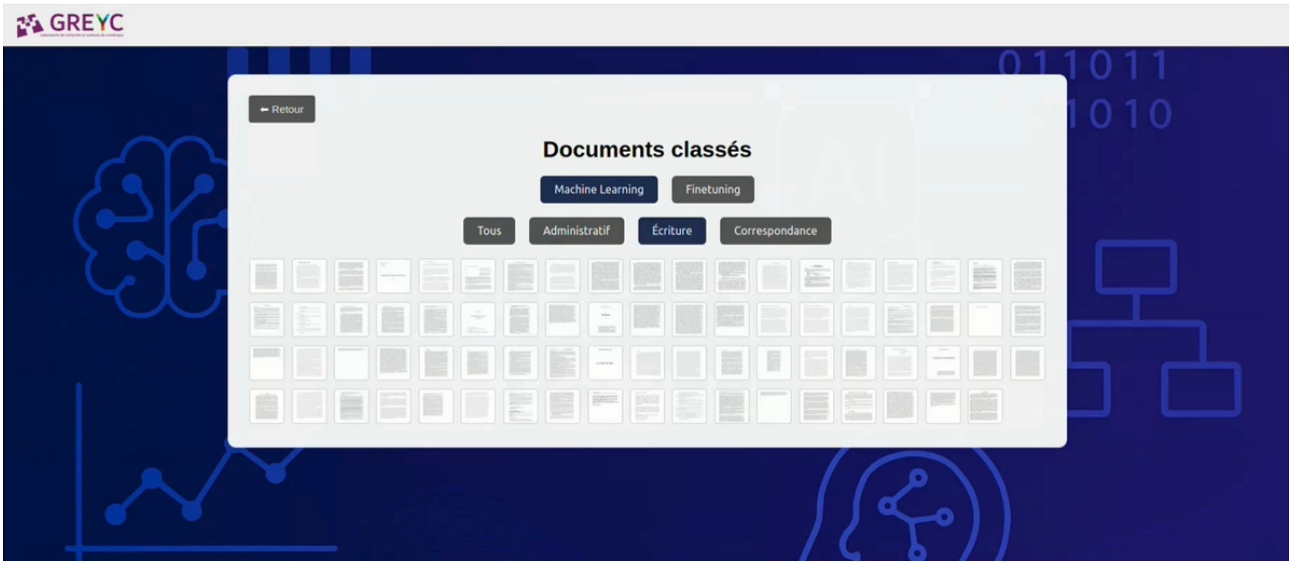


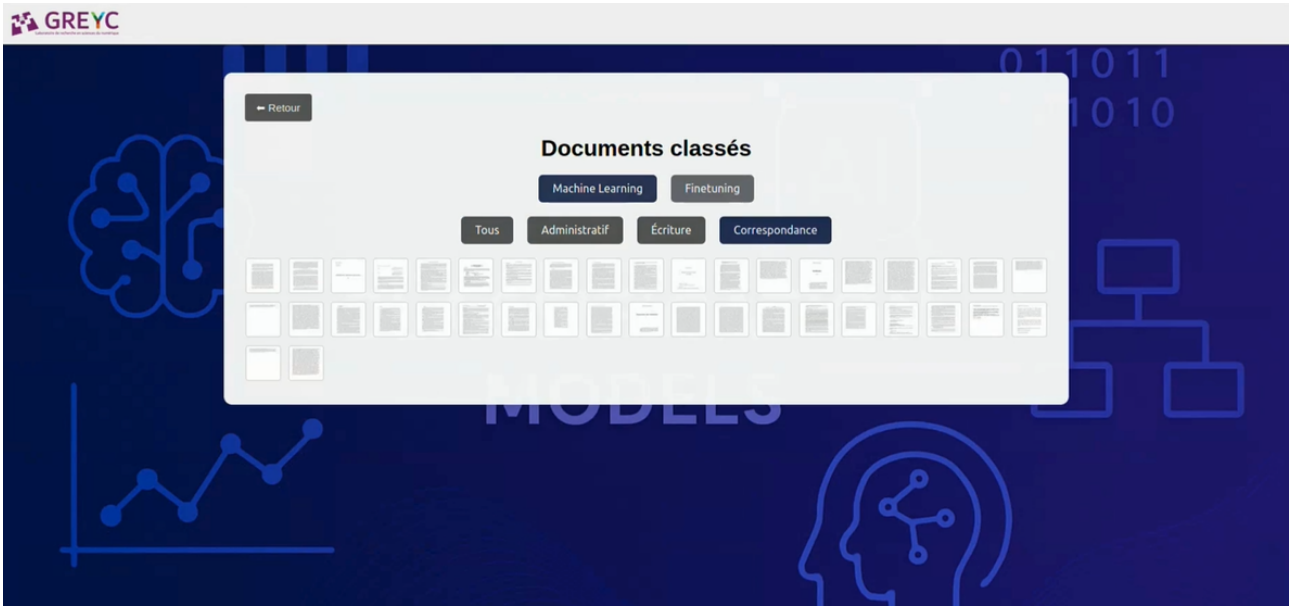
8.3 Résultat OCR document Imprimé



8.4 Classification Textuelle avec Machine Learning







Conclusion

Ce stage m'a permis de plonger au cœur d'une problématique passionnante : comment rendre les archives numériques des écrivains contemporains accessibles et exploitables grâce aux outils de l'intelligence artificielle. En alliant des approches de classification automatique, des techniques de traitement du langage naturel, et l'utilisation de modèles de type LLM, j'ai pu expérimenter différentes méthodes pour analyser, structurer et catégoriser des documents textuels et visuels issus de corpus complexes.

Au fil des semaines, j'ai acquis des compétences concrètes en extraction d'information, en prétraitement de données textuelles, en entraînement et évaluation de modèles de machine learning, mais aussi en fine-tuning de modèles de langage comme Mistral. J'ai également abordé la dimension multimodale à travers l'utilisation de Pixtral-OCR pour la classification et l'analyse d'images, en explorant des stratégies de prompt engineering adaptées à ces usages.

Ce travail m'a aussi confrontée à des défis concrets : des données sensibles, parfois difficiles à traiter, un vocabulaire très varié selon les auteurs, des catégories parfois proches et donc difficiles à distinguer. J'ai appris à faire preuve d'adaptation, de méthode, mais aussi de créativité pour surmonter ces difficultés et ajuster les choix techniques.

ces travaux m'ont permis de metriser l'usage de l'intelligence artificielle. J'y ai découvert comment les techniques de classification et d'apprentissage automatique peuvent renforcer l'investigation numérique, un domaine où IA et cybersécurité se rejoignent. Cette expérience m'a inspiré des idées innovantes dans le domaine de la sécurité, afin de contribuer à relever les défis actuels et futurs.

Références

[1] Tamanna, T. *Deciphering Accuracy Evaluation Metrics in NLP and OCR...*, Medium. https://medium.com/@tam.tamanna18/deciphering-accuracy-evaluation-metrics-in-nlp-and-ocr-a-comparison-of-character-err

[2] Code & Cortex. *spaCy – Natural Langage – Lemmatisation*. https://www.codeandcortex.fr/spacy-natural-langage-process-lemmatisation/

[3] Mistral AI. *Prompting Capabilities (Classification)*. https://docs.mistral.ai/guides/prompting_capabilities/#classification

[4] Wikipédia. *Précision et rappel*. https://fr.wikipedia.org/wiki/Pr%C3%A9cision_et_rappel

[5] GeeksforGeeks. *Machine Learning — Introduction*. https://www.geeksforgeeks.org/machine-learning

[6] Code & Cortex. *spaCy — Présentation générale*. https://www.codeandcortex.fr/spacy

[7] Amit, H. *Fine-tuning Mistral 7B : A Practical Guide*, Medium. https://medium.com/@heyamit10/fine-tuning-mistral-7b-a-practical-guide-b18f63f27e56

[8] Mistral AI. *Self-deployment with vLLM*. https://docs.mistral.ai/deployment/self-deployment/vllm/

[9] Klippa. *Qu'est-ce que Tesseract OCR ?*. https://www.klippa.com/fr/blog/informations/quest-ce-que-tesseract-ocr/