المدرسة الوطنية للعلوم التطبيقية بتطوان
ⵜⵉⵍⵍⴰⵏ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵎⵓⵙⵙⵉⵙⵉⵏ ⵜⴰⵎⵓⵔⵜ
Ecole Nationale des Sciences Appliquées de Tétouan

Université Abdelmalek Essaâdi

École Nationale des Sciences Appliquées

Tétouan, Maroc

Data Science, Big Data & Intelligence Artificielle

Rapport de fin de projet

Classification multi-label des articles de presse

Créé par :

Anouar Bouzhar
Hamza Kholti
Zakaryae Bennani
Ahmed Bakkali

Proposé par :

Pr. Belcaid Anas

Module : TALN — Traitement Automatique du Langage Naturel

Année universitaire : 2025–2026

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Ce projet s'inscrit dans le cadre du module de Traitement Automatique du Langage Naturel, dont l'objectif est de fournir une compréhension approfondie des principes fondamentaux du NLP et d'en explorer les applications pratiques. Il porte sur la mise en œuvre des concepts clés du NLP, tels que le prétraitement des textes (e.g., nettoyage, normalisation, tokenisation, stemming et lemmatisation), la représentation vectorielle du texte (sparse embeddings, dense embeddings) ainsi que les les modèles séquentielles et basés sur les Transformers.

Le projet se concentre particulièrement sur la classification de textes, une tâche fondamentale en NLP. La classification de textes consiste à entraîner un modèle sur un ensemble de documents étiquetés, $T = \{X_1, X_2, ..., X_m\}$, chacun associé à une étiquette parmi un ensemble de k classes possibles, afin de prédire la classe de nouveaux documents. Cette tâche peut également inclure la classification multilabel, où un document peut appartenir à plusieurs catégories, ou la classification hiérarchique, où les étiquettes suivent une structure hiérarchique. La classification de textes trouve de nombreuses applications pratiques, telles que le filtrage d'actualités, l'organisation de documents dans des bibliothèques numériques, l'analyse des réseaux sociaux ou le tri automatique des emails.

À travers ce projet, nous cherchons à appliquer les techniques de NLP à une problématique concrète, démontrant comment l'automatisation du traitement de données textuelles peuvent être mises en œuvre efficacement dans les environnements modernes.

2 Problématique et objectifs

2.1 Problématique

Les articles de presse représentent un domaine particulièrement intéressant et complexe pour la classification de textes. La classification thématique automatique permet d'étiqueter les articles sur les plateformes d'actualités en ligne, de regrouper différentes sources par thème (comme le fait Google News) et de servir de base aux systèmes de recommandation. Au-delà des applications techniques, l'influence sociale et politique des médias attire l'attention des spécialistes de la communication et des chercheurs en sciences sociales. L'analyse automatisée des données journalistiques peut mettre en évidence des tendances, des biais et des déséquilibres dans la production, le contenu et la diffusion de l'information.

Cela soulève la problématique centrale : comment concevoir des méthodes de classification de textes efficaces, fiables et interprétables pour les articles de presse, capables d'organiser et d'étiqueter le contenu de manière précise dans les médias ?

2.2 Objectives

Dans ce projet, nous allons :

- *Étudier l'impact de la classification multi-label à travers l'utilisation d'un jeu de données réel.*
- *Expérimenter et comparer différents algorithmes de classification, allant des méthodes classiques aux approches modernes basées sur le transformers.*
- *Développer une application prototype permettant la classification automatique d'articles de presse.*

3 Description du jeu de données utilisée

MultiEURLEX est un jeu de données de grande échelle, multilingue, conçu pour l'évaluation de la *classification de textes multi-label hiérarchique* et du *transfert interlingue*.. Il constitue une extension du jeu de données monolingue EUR-LEX et couvre 23 langues officielles de l'Union européenne.

3.1 Composition du corpus et parallélisme

Le corpus est composé d'environ **65 000 articles** extraits du portail EUR-LEX.
Le caractère multilingue du jeu de données repose sur l'obligation légale de publier les textes dans toutes les langues officielles de l'Union européenne. Toutefois, la couverture linguistique varie en fonction de la date d'adhésion des États membres. Les ensembles de validation et de test contiennent chacun **5 000 documents parfaitement alignés** dans les 23 langues, garantissant un parallélisme complet pour l'évaluation.
L'ensemble d'entraînement comprend environ **55 000 documents** pour les langues fondatrices de l'Union. En revanche, pour les langues introduites plus récemment (par exemple le croate, adhésion en 2013), le nombre de documents disponibles est plus limité, ce qui introduit un scénario réaliste de *faible ressource*.

3.2 Espace d'étiquettes et granularité sémantique (EUROVOC)

L'annotation sémantique des documents repose sur **EUROVOC**, un thésaurus multidisciplinaire maintenu par l'Office des publications de l'Union européenne. L'espace d'étiquettes est de grande cardinalité et structuré sous la forme d'un graphe orienté acyclique (DAG).
Les annotations sont fournies à plusieurs niveaux de granularité hiérarchique :

- **Niveau 1** : 21 catégories sémantiques générales (par exemple *Politics*, *Agri-Foodstuffs*), dont seulement 14 sont effectivement exploitées dans le jeu de données.
- **Niveau 2** : 127 concepts intermédiaires.
- **Niveau 3** : 567 concepts fins.

Le jeu de données est strictement **multi-label**, chaque document étant associé à un nombre variable d'étiquettes actives.

Bien que la hiérarchie EUROVOC définisse initialement 21 concepts de premier niveau, le jeu de données MultiEURLEX n'en exploite effectivement que 14 dans ses ensembles d'entraînement, de validation et de test. Les autres catégories de haut niveau ne sont pas représentées dans les annotations finales ou ont été exclues lors du prétraitement du corpus. Ainsi, dans ce travail, la classification est réalisée sur un espace de 14 classes de haut niveau, conformément à la distribution réelle des labels dans le jeu de données.

3.3 Découpage temporel et dérive conceptuelle

Contrairement aux découpages aléatoires classiques, MultiEURLEX adopte une **stratégie de découpage chronologique**, afin de simuler des conditions de déploiement réalistes et d'évaluer la généralisation temporelle des modèles.

- **Ensemble d'entraînement** : documents publiés entre 1958 et 2010.
- **Ensemble de validation** : documents publiés entre 2010 et 2012.
- **Ensemble de test** : documents publiés entre 2012 et 2016.

Ce découpage introduit un phénomène de **dérive conceptuelle** (*concept drift*), caractérisé par des changements dans la distribution des concepts et du vocabulaire au fil du temps, rendant la tâche de classification plus complexe qu'avec un découpage aléatoire.

3.4 Prétraitement et Préparation des Données

3.4.1 Ingénierie des Labels (Taxonomie EuroVoc)

Le défi principal de ce dataset réside dans la complexité de ses labels originaux (concepts EuroVoc). Une étape de simplification et de regroupement a été réalisée pour passer d'une granularité fine à des catégories thématiques de haut niveau.

- **Récupération des Sémantiques** : Le dataset original ne fournissant que des identifiants numériques (ex : 100163), un script a été utilisé pour interroger l'endpoint

SPARQL d'EuroVoc. Cela a permis d'associer chaque ID à sa description textuelle (ex : 100163 → "political framework").

- **Regroupement Thématique (Mapping)** : Les concepts EuroVoc ont été agrégés manuellement en 14 méta-catégories pour simuler une classification de type "Rubriques Journalistiques".
  - Exemple : Les concepts "political framework", "executive power", et "EU finance" ont été regroupés sous la catégorie "Politics Government".
  - Les 14 catégories finales incluent : Economics Finance, Trade Business, Social Affairs, Technology, Environment, etc.

3.4.2 Encodage des Cibles (Multi-Hot Encoding)

Puisqu'un document peut appartenir à plusieurs catégories simultanément (Classification Multi-Label), les étiquettes ne peuvent pas être représentées par un simple entier. Nous avons transformé les labels en vecteurs Multi-Hot :

- Chaque document est associé à un vecteur de taille 14 (nombre de catégories).
- La valeur 1 est attribuée aux indices correspondant aux catégories présentes, et 0 ailleurs.
- Exemple : Un article parlant de taxes sur le carbone aura des 1 aux indices correspondants à "Economics" et "Environment".

Note technique : Cette représentation est indispensable pour utiliser la fonction de perte BCEWithLogitsLoss (Binary Cross Entropy) lors de l'entraînement du modèle neural.

3.4.3 Nettoyage et Filtrage

Une étape de validation a été effectuée pour garantir la qualité des données d'entraînement :

- Les documents ne correspondant à aucune des 14 méta-catégories définies (labels résiduels ou non mappés) ont été retirés du jeu de données via un filtre conditionnel (if ex["labels"] != -1).
- Cela assure que le modèle n'apprend que sur des données correctement annotées.

3.4.4 Tokenisation et Formatage BERT

Le texte brut des articles ne peut pas être ingéré directement par le modèle. Nous avons utilisé le Tokenizer spécifique au modèle pré-entraîné (bert-base-uncased ou RoBerta). Les opérations effectuées par le NewsMultiLabelDataset sont les suivantes :

- Tokenisation en sous-mots (Subword Tokenization) : Découpage des mots en unités plus petites pour gérer le vocabulaire rare.
- Troncature (Truncation) : Les textes étant souvent très longs, ils ont été tronqués à une longueur maximale de 512 tokens (limite structurelle de l'architecture BERT standard).
- Troncature (Truncation) : Les textes étant souvent très longs, ils ont été tronqués à une longueur maximale de 512 tokens (limite structurelle de l'architecture BERT standard).
- Padding : Les séquences plus courtes que 512 tokens ont été complétées par des tokens de remplissage ([PAD]) pour uniformiser la taille des tenseurs (Batch processing).
- Tenseurs PyTorch : Conversion finale des $\text{input}_i ds, attention_m asketlabelsenobjetstorch.Tensorprt$

3.4.5 Statistiques après Prétraitement

À l'issue de ces transformations, la distribution des données est la suivante :

- Train Set : 55 000 documents
- Validation Set : 5 000 documents
- Test Set : 5 000 documents
- Distribution des classes : L'analyse graphique montre un déséquilibre des classes, avec une prédominance des catégories "Agriculture Food" et "Trade Business", ce qui est typique des corpus législatifs européens.

4 Métriques utilisées

Pour toutes les formules ci-dessous, nous utilisons les notations suivantes :

N : nombre total d'articles.

L : nombre de catégories (14 dans votre cas).

$y_{i,j}$: la valeur réelle (0 ou 1) pour l'article i et la catégorie j.

$\hat{y}_{i,j}$: la valeur prédite (0 ou 1) par le modèle.

4.1 Hamming Loss (Perte de Hamming)

La Hamming Loss mesure la proportion moyenne de labels incorrectement prédits (erreurs de type faux positifs et faux négatifs confondus).

Formule :

$$Hamming\ Loss = \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{j=1}^{L} I(y_{i,j} \neq \hat{y}_{i,j})$$

Où $I(\cdot)$ est une fonction indicatrice qui vaut 1 si la prédiction est différente de la réalité, 0 sinon.

Explication : C'est la métrique la plus souple. Plus elle est proche de 0, plus le modèle est performant. Elle est idéale pour le multi-label car elle ne rejette pas une prédiction si seulement 1 label sur 14 est erroné.

4.2 Subset Accuracy (Exactitude par sous-ensemble)

Aussi appelée "Exact Match Ratio", c'est la métrique la plus exigeante. Elle exige que l'ensemble des étiquettes prédites soit strictement identique à l'ensemble réel.

Formule :

$$Subset\ Accuracy = \frac{1}{N} \sum_{i=1}^{N} I(y_i = \hat{y}_i)$$

Explication : Si un article appartient aux catégories A et B, et que le modèle prédit uniquement A, le score pour cet article est 0. Cette métrique est souvent basse car elle ne tolère aucune erreur partielle.

4.3 F1 Score (Micro-moyenne)

Le score F1 est la moyenne harmonique entre la précision et le rappel. La version "Micro" agrège les contributions de toutes les classes pour calculer une performance globale.

Formule : On calcule d'abord le total des Vrais Positifs (VP), Faux Positifs (FP) et Faux Négatifs (FN) sur l'ensemble des classes :

$$F1_{micro} = \frac{2 \cdot \sum VP_j}{2 \cdot \sum VP_j + \sum FP_j + \sum FN_j}$$

Explication : Cette métrique est dominée par les classes les plus fréquentes. Elle reflète l'efficacité du modèle sur la masse totale des données.

4.4 F1 Score (Macro-moyenne)

La version "Macro" calcule le score F1 indépendamment pour chaque catégorie, puis en fait la moyenne arithmétique.

Formule :

$$F1_{macro} = \frac{1}{L} \sum_{j=1}^{L} F1_j$$

Où $F1_j$ est le score F1 calculé pour la classe j uniquement.

Explication : C'est la métrique la plus importante pour détecter si votre modèle est équitable. Un F1-Macro élevé signifie que votre modèle réussit aussi bien sur les catégories rares que sur les catégories fréquentes du dataset EURLEX.

5 Modèles utilisées

5.1 Classifier Chain avec Support Vector Machine

5.1.1 Composants du modèle

TF–IDF

TF–IDF (*Term Frequency – Inverse Document Frequency*) est une méthode statistique largement utilisée en traitement automatique du langage naturel (NLP) et en recherche

d'information. Elle permet d'évaluer l'importance d'un terme dans un document relativement à un ensemble de documents appelé *corpus*. Cette méthode repose sur la combinaison de deux mesures complémentaires : la fréquence de terme (TF) et la fréquence inverse de document (IDF).

Fréquence de terme (TF)

La fréquence de terme mesure le nombre de fois qu'un terme t apparaît dans un document d. Plus un terme apparaît fréquemment dans un document, plus il est susceptible d'être pertinent pour le contenu de ce document. Elle est définie par :

$$\mathrm{TF}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

où $f_{t,d}$ représente le nombre d'occurrences du terme t dans le document d.

Fréquence inverse de document (IDF)

La fréquence inverse de document permet de réduire l'importance des termes très fréquents dans l'ensemble du corpus (comme les mots courants) et d'augmenter celle des termes rares, souvent plus informatifs. Elle est définie comme suit :

$$\mathrm{IDF}(t) = \log\left(\frac{N}{\mathrm{DF}(t)}\right)$$

où N désigne le nombre total de documents dans le corpus et $\mathrm{DF}(t)$ le nombre de documents contenant le terme t.

La formule de TF–IDF

La formule de TF–IDF d'un terme t dans un document d est obtenu par la multiplication des deux mesures précédentes :

$$\text{TF-IDF}(t,d) = \mathrm{TF}(t,d) \times \mathrm{IDF}(t)$$

Cette pondération permet de mettre en évidence les termes qui sont à la fois fréquents dans un document donné et rares dans le corpus global.

Représentation vectorielle des documents

Chaque document est représenté sous forme d'un vecteur de poids TF–IDF :

$$\vec{d} = \left[\text{TF-IDF}(t_1, d), \text{TF-IDF}(t_2, d), \ldots, \text{TF-IDF}(t_n, d)\right]$$

Singular Value Decomposition

La réduction de dimensionnalité vise à simplifier la représentation de données complexes et fortement multidimensionnelles, telles que les données textuelles ou les images, tout en préservant l'essentiel de l'information qu'elles contiennent. Elle consiste à projeter des observations initialement définies dans un espace de grande dimension vers un sous-espace de dimension plus réduite, facilitant ainsi leur analyse, leur comparaison et leur visualisation.

Dans le cadre de ce projet, nous appliquons la Décomposition en Valeurs Singulières (SVD) après la phase de vectorisation par TF-IDF afin de traiter efficacement la forte dimensionalité et la sparsité induites par cette représentation. En effet, la matrice TF-IDF est généralement de grande taille et contient de nombreuses dimensions redondantes ou faiblement informatives. L'utilisation de la SVD permet alors de compresser cette matrice tout en conservant les structures sémantiques essentielles.

Définition mathématique

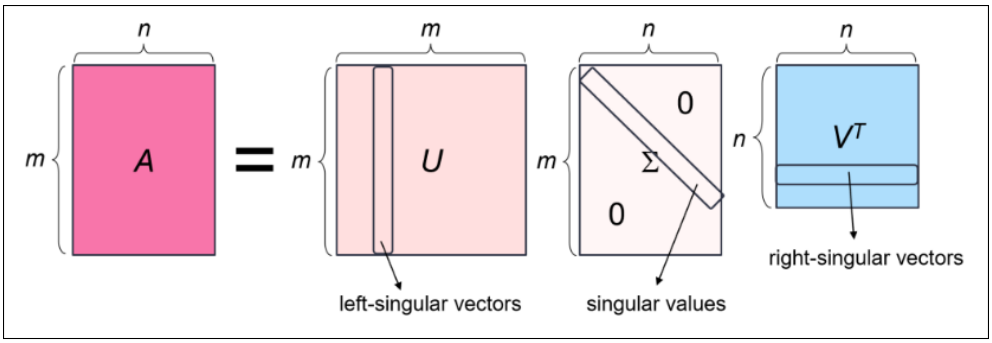


FIGURE 1 : SVD d'une matrice

La décomposition en valeurs singulières (SVD) d'une matrice réelle $A \in \mathbb{R}^{m \times n}$ est une factorisation de la forme :

$$A = U \Sigma V^{\top},$$

où :

- U est une matrice orthogonale de dimension $m \times m$ (c'est-à-dire que ses colonnes et ses lignes sont des vecteurs orthonormés). Les colonnes de U sont appelées les *vecteurs singuliers à gauche* de A.
- Σ est une matrice diagonale rectangulaire de dimension $m \times n$ contenant des nombres réels non négatifs sur la diagonale. Les éléments diagonaux $\sigma_i = \Sigma_{ii}$ sont appelés les *valeurs singulières* de A et sont généralement ordonnés par ordre décroissant :

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

  Le nombre de valeurs singulières non nulles est égal au rang de A.
- V est une matrice orthogonale de dimension $n \times n$. Les colonnes de V sont appelées les *vecteurs singuliers à droite* de A.

Support Vector Machine

L'algorithme SVM nécessite la résolution du problème d'optimisation suivant :

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.c.} \quad & y_i(w^\top x_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \ldots, n, \end{aligned} \tag{1}$$

où C est un paramètre de régularisation.
L'objectif du SVM est de trouver un hyperplan linéairement séparateur qui maximise la marge dans l'espace de dimension supérieure vers lequel x_i est projeté .
Dans le cadre de notre projet, nous avons exploité la bibliothèque `cuml` de Python afin de mettre en œuvre un classifieur à vecteurs de support (SVC et LinearSVM). Cette bibliothèque est optimisée pour le calcul sur GPU, et donc va nous permettre d'effectuer l'entraînement et la prédiction de manière efficace sur de grands ensembles de données.

Classifier Chain

La méthode des Classifier Chains (CC) consiste à connecter plusieurs classificateurs binaires en une « chaîne », de sorte que la prédiction d'un classificateur est ajoutée comme

caractéristique supplémentaire à l'entrée des classificateurs suivants. Cette approche permet de modéliser les corrélations entre labels et améliore les performances par rapport à la méthode de référence Binary Relevance (BM), qui traite chaque label indépendamment.

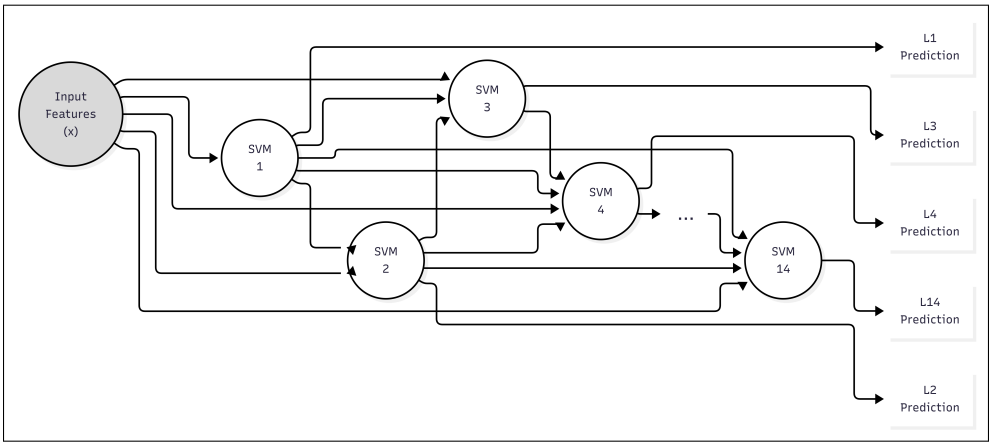


FIGURE 2 : Structure de classifier chain

Dans le modèle Classifier Chain, un ensemble de $|L|$ classificateurs binaires est utilisé, un pour chaque label $l_j \in L$. Chaque classificateur C_j résout le problème de pertinence binaire pour son label correspondant. L'espace de caractéristiques de chaque classificateur est étendu avec les prédictions binaires de tous les classificateurs précédents.
Ainsi, une chaîne $C_1, \ldots, C_{|L|}$ de classificateurs binaires est formée. Chaque classificateur C_j prédit l'association binaire de son label l_j en tenant compte de l'espace de caractéristiques initial, enrichi par les prédictions précédentes $l_1, \ldots, l_{j-1}$. Cette méthode permet de capturer les corrélations entre labels, tout en conservant les avantages de BM.

Les hyperparamètres du classificateur de base ont été ajustés à l'aide de la méthode Grid Search, permettant d'identifier la configuration optimale à partir d'un ensemble prédéfini de valeurs de C et gamma.

5.1.2 Prétraitement des donnees et Pipeline dentraînement

Prétraitement des données

Avant l'extraction des caractéristiques et la phase de classification, les données textuelles brutes subissent une série d'étapes de prétraitement visant à réduire le bruit, normaliser le texte et améliorer la qualité des caractéristiques extraites.

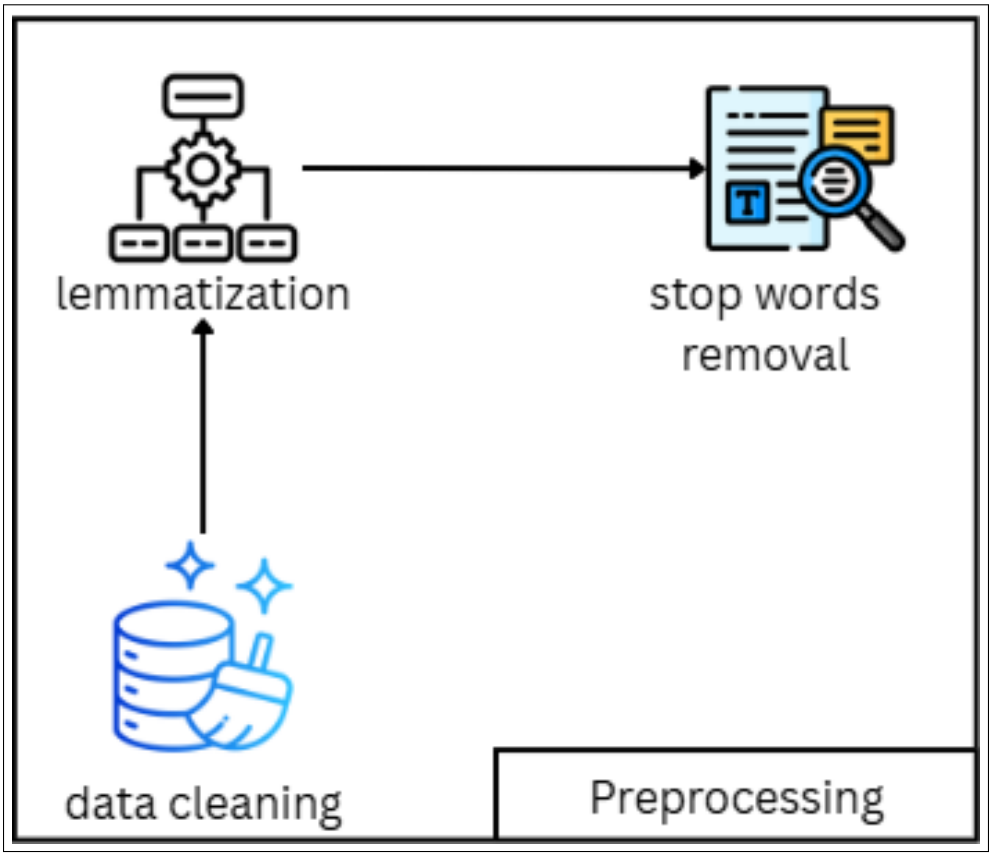


FIGURE 3 : vue d'ensemble du Pipeline du prétraitement

Dans un premier temps, le pipeline débute par une phase de normalisation, au cours de laquelle l'ensemble du texte est converti en minuscules afin d'assurer l'insensibilité à la casse et de réduire la redondance du vocabulaire. Les artefacts d'encodage et les caractères accentués sont normalisés, les retours à la ligne sont remplacés par des espaces, et les traits d'union et les apostrophes sont traités de manière cohérente afin d'éviter la fragmentation artificielle des tokens.

Ensuite, une étape de suppression du bruit est réalisée à l'aide d'expressions régulières. Cette étape élimine les éléments qui n'apportent pas de signification sémantique à la classification, tels que les URL, les adresses IP, les valeurs numériques, la ponctuation et

d'autres caractères spéciaux. La suppression de ces éléments permet de réduire le bruit et d'éviter l'introduction de dimensions non pertinentes dans l'espace des caractéristiques.
Après le nettoyage, le texte est segmenté au niveau des phrases puis des mots. Les tokens de très petite taille sont éliminés, car ils correspondent généralement à des mots-outils ou à des fragments non informatifs. Un nettoyage supplémentaire est effectué au niveau des tokens afin de supprimer tout symbole ou chiffre résiduel.
Une étape de lemmatisation est ensuite appliquée à l'aide du lemmatiseur WordNet. Cette opération ramène les formes fléchies des mots à leur forme canonique tout en préservant leur sens linguistique. La lemmatisation permet de représenter différentes variantes grammaticales d'un même mot par une seule unité lexicale, réduisant ainsi la taille du vocabulaire et améliorant la capacité de généralisation du classificateur.
Enfin, les stopwords sont supprimés à l'aide d'une combinaison de stopwords standards en anglais et d'une liste personnalisée de termes spécifiques au domaine. Ces mots sont très fréquents mais possèdent une faible valeur discriminante dans les corpus journalistiques et juridiques. Leur suppression permet de mettre en avant les termes sémantiquement informatifs et d'améliorer la qualité des caractéristiques extraites.

pipeline de modélisation

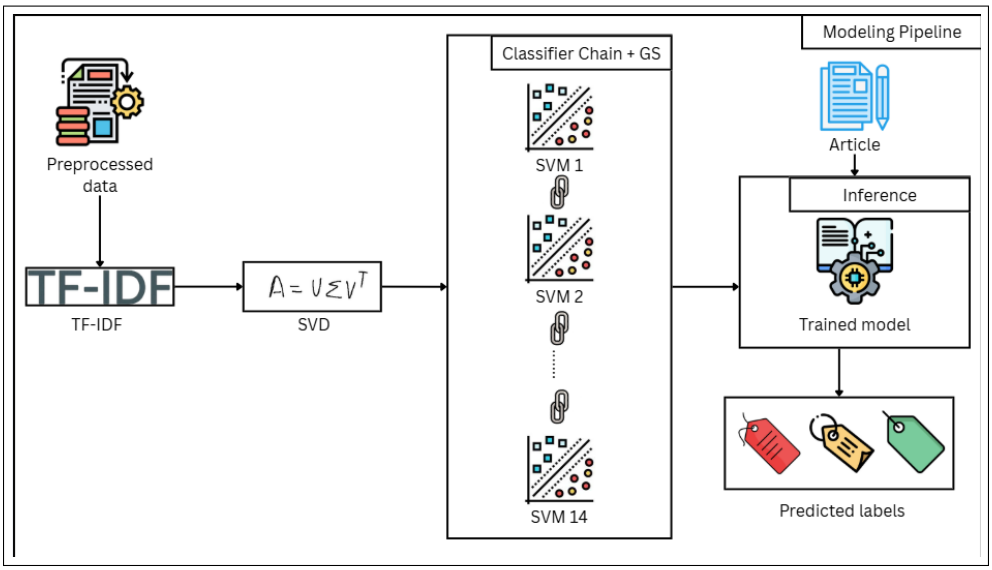


FIGURE 4 : Pipeline de modélisation

Le pipeline de modélisation proposé, illustré à la Figure 4, présente une approche complète pour la classification de texte multi-label. Le processus débute avec des données

textuelles prétraitées, qui sont ensuite transformées en vecteurs numériques à l'aide de la technique **TF-IDF**. Afin de réduire la dimensionnalité, une **SVD** est appliquée aux données vectorisées.
L'architecture de classification repose sur une stratégie de **Classifier Chain** (Chaîne de Classifieurs), conçue pour modéliser les corrélations entre les étiquettes. Une recherche par grille (**Grid Search**) est intégrée pour l'ajustement des hyperparamètres.

Enfin, l'étape d'inférence démontre la capacité du système à traiter de nouveaux articles et à générer des étiquettes prédites précises grâce au modèle entraîné.

5.1.3 Interprétation des résultats

Après avoir mené l'ensemble des expérimentations sur le jeu de données, nous présentons dans les tableaux suivants les performances obtenues par les différentes configurations de SVM.

TABLE 1 : Performances en F1-score (macro et micro) des SVM linéaire et RBF dans un cadre commun de Classifier Chains avec optimisation par GridSearch.

	SVM	**TF-IDF**	**SVD**	**F1-score (Macro)**			**F1-score (Micro)**		
				Train	Val	Test	Train	Val	Test
CC + Grid Search	Linear	4096	256	0.6499	0.6459	0.6145	0.8082	0.8037	0.7687
			512	0.6860	0.7033	0.6679	0.8238	0.8208	0.7871
			1024	0.6999	0.7232	0.6898	0.8302	0.8376	0.8132
		8192	256	0.6136	0.6040	0.5625	0.7991	0.7887	0.7512
			512	0.6888	0.6974	0.6714	0.8227	0.8184	0.7886
			1024	0.7111	0.7194	0.6875	0.8323	0.8272	0.7945
	RBF	4096	256	0.8356	0.8345	0.7412	0.9058	0.9012	0.8392
			512	0.9014	0.8927	0.7630	0.9380	0.9340	0.8464
			1024	0.9353	0.9262	0.7710	0.9551	0.9520	0.8470
		8192	256	0.8240	0.8034	0.7371	0.9005	0.8932	0.8378
			512	0.8540	0.8460	0.7480	0.9160	0.9110	0.8425
			1024	0.8773	0.8695	0.7561	0.9256	0.9212	0.8447

TABLE 2 : Subset accuracy et Hamming score des SVM linéaire et RBF dans un cadre commun de Classifier Chains avec optimisation par GridSearch.

	SVM	**TF-IDF**	**SVD**	**Subset Accuracy**			**Hamming Score**		
				Train	Val	Test	Train	Val	Test
CC + Grid Search	Linear	4096	256	0.3899	0.3564	0.2500	0.0763	0.0841	0.0966
			512	0.4132	0.3724	0.2702	0.0704	0.0780	0.0908
			1024	0.4242	0.4340	0.3752	0.0679	0.0717	0.0812
		8192	256	0.3768	0.3278	0.2434	0.0791	0.0894	0.1024
			512	0.4079	0.3668	0.2784	0.0710	0.0790	0.0902
			1024	0.4254	0.3794	0.2840	0.0675	0.0759	0.0885
	RBF	4096	256	0.6601	0.6364	0.4578	0.0387	0.0443	0.0699
			512	0.7651	0.7424	0.4704	0.0258	0.0299	0.0671
			1024	0.8201	0.8014	0.4648	0.0188	0.0220	0.0676
		8192	256	0.6451	0.6210	0.4562	0.0408	0.0471	0.0704
			512	0.6820	0.6573	0.4629	0.0358	0.0413	0.0689
			1024	0.7189	0.6936	0.4696	0.0308	0.0355	0.0674

SVM linéaire : Le SVM linéaire constitue une base de référence stable et robuste. Les résultats montrent des valeurs de micro-F1 relativement élevées, comprises entre 0.77 et 0.82 sur l'ensemble de test, indiquant une bonne capacité à prédire correctement les étiquettes fréquentes. En revanche, le macro-F1 reste sensiblement plus faible, ce qui traduit une difficulté à modéliser les classes rares, phénomène classique dans les contextes de classification multi-étiquettes déséquilibrée.

L'augmentation de la dimension de la réduction SVD améliore progressivement les performances, la configuration à 1024 dimensions offrant les meilleurs compromis. Néanmoins, la subset accuracy demeure faible, confirmant que le modèle linéaire peine à prédire correctement l'ensemble complet des étiquettes. Le Hamming score, relativement stable, suggère un comportement conservateur et une bonne généralisation, avec des écarts limités entre les phases d'entraînement et de test.

SVM RBF : Le SVM à noyau RBF, combiné aux classifier chains et optimisé par GridSearch, présente une capacité de modélisation nettement supérieure. Les scores F1, tant macro que micro, atteignent des valeurs très élevées en entraînement et validation, et surpassent clairement le SVM linéaire sur l'ensemble de test. Cette amélioration est particulièrement notable pour le macro-F1, ce qui indique une meilleure prise en compte

des classes rares.

La subset accuracy progresse significativement, montrant que le modèle RBF est plus apte à prédire des ensembles d'étiquettes complets. Toutefois, l'écart entre les performances d'entraînement et celles de test révèle un risque de surapprentissage, malgré l'utilisation d'une recherche d'hyperparamètres. Par ailleurs, le Hamming score légèrement inférieur suggère une augmentation du nombre moyen d'erreurs par étiquette, en contrepartie d'une meilleure cohérence globale des prédictions.

Comparaison

La comparaison met en évidence deux comportements complémentaires. Le SVM linéaire se distingue par sa simplicité, sa rapidité et sa stabilité, mais reste limité en termes de performance globale et de prédiction exacte des ensembles d'étiquettes. À l'inverse, le SVM RBF avec classifier chains offre des performances supérieures en F1-score et en subset accuracy, au prix d'une complexité accrue et d'un risque plus marqué de surapprentissage.

Dans un contexte applicatif où la qualité globale des prédictions multi-étiquettes est prioritaire, le SVM RBF constitue le choix le plus pertinent. Néanmoins, une validation plus stricte, un réglage fin de la régularisation et éventuellement une combinaison avec un modèle linéaire pourraient permettre d'améliorer encore la robustesse et la généralisation du système.

5.2 XGBoost (eXtreme Gradient Boosting)

Après avoir évalué les SVM, le deuxième modèle examiné est XGBoost. Il s'agit d'un algorithme appartenant à la famille des méthodes d'ensemble (Ensemble Learning), spécifiquement basé sur le Gradient Boosting. Nous avons choisi ce modèle pour sa capacité reconnue à traiter efficacement des données structurées, sa robustesse face au bruit et au déséquilibre des classes, ainsi que pour servir de "Baseline" robuste face aux modèles de Deep Learning présentés ultérieurement. XGBoost s'est imposé comme une référence incontournable dans les compétitions de Data Science (notamment Kaggle) et dans l'industrie grâce à son excellent compromis entre performance prédictive et efficacité computationnelle.

5.2.1 Architecture de XGBoost

XGBoost (*eXtreme Gradient Boosting*) est une implémentation hautement performante et évolutive de l'algorithme de *Gradient Boosting*. Appartenant à la famille des méthodes d'ensemble, il repose sur le principe du *Boosting*, qui consiste à combiner séquentiellement un ensemble de modèles faibles (généralement des arbres de décision *CART - Classification and Regression Trees*) pour former un modèle prédictif robuste et puissant.

Contrairement aux forêts aléatoires (*Random Forest*) qui construisent des arbres indépendants en parallèle (paradigme *Bagging*), XGBoost construit les arbres l'un après l'autre dans une approche séquentielle et adaptative. Chaque nouvel arbre f_t a pour objectif spécifique de corriger les erreurs résiduelles commises par l'ensemble des arbres précédents $f_1, ..., f_{t-1}$. Cette approche permet de focaliser progressivement l'apprentissage sur les cas difficiles mal classés par les itérations précédentes, améliorant ainsi la capacité de généralisation du modèle.

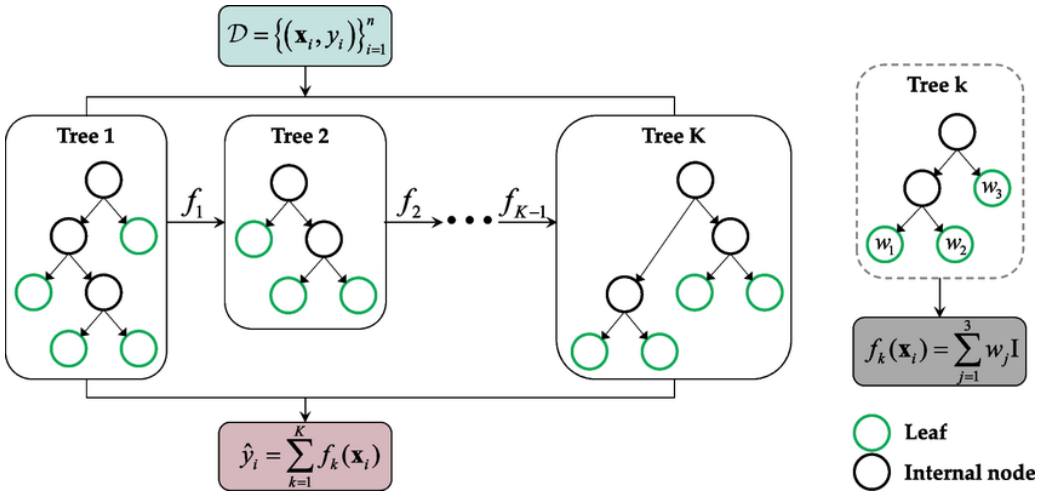


FIGURE 5 : Architecture séquentielle de XGBoost

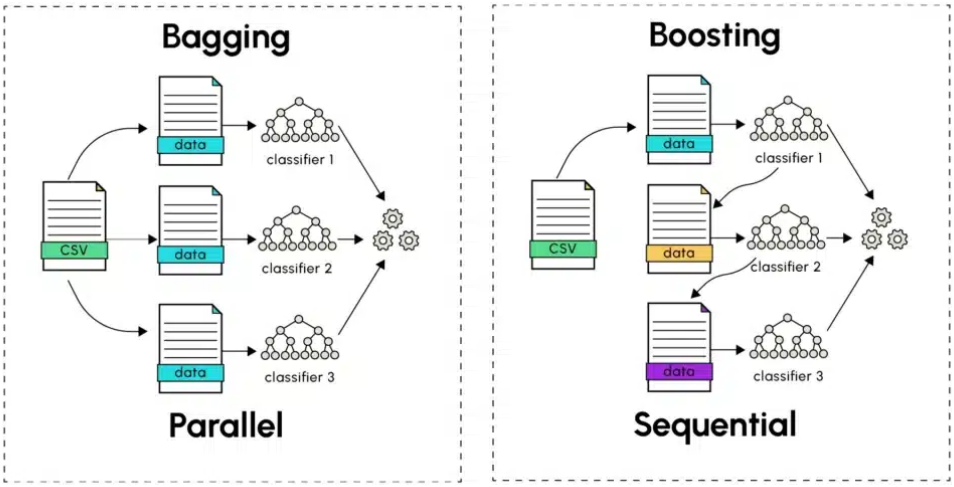


FIGURE 6 : Comparaison Conceptuelle Entre Boosting et Bagging.

5.2.2 Fondements Mathématiques et Fonction Objectif

Le modèle final après K itérations pour une donnée x_i est exprimé par la somme additive suivante :

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F} \tag{2}$$

où $\mathcal{F}$ est l'espace des fonctions contenant tous les arbres de régression possibles (CART). Cette formulation additive permet d'interpréter chaque arbre comme une correction incrémentale du modèle global.

L'innovation majeure de XGBoost réside dans sa fonction objectif régularisée, qui équilibre précision et complexité du modèle. Pour optimiser le modèle à l'étape t, nous cherchons à minimiser :

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{3}$$

Cette fonction comporte deux termes distincts et complémentaires :

- **Le terme de perte l :** Une fonction différentiable convexe qui mesure l'écart entre la prédiction $\hat{y}_i$ et la valeur réelle y_i. Pour la classification binaire, on utilise typiquement la Log-Loss (entropie croisée) :

$$l(y_i, \hat{y}_i) = -[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{4}$$

- **Le terme de régularisation Ω :** Il pénalise la complexité du modèle pour prévenir le surapprentissage (*overfitting*), défini par :

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{5}$$

où T est le nombre de feuilles terminales de l'arbre, w_j représente le poids (score) associé à la feuille j, γ (gamma) pénalise le nombre de feuilles (favorisant des arbres plus simples), et λ (lambda) applique une régularisation L2 sur les poids des feuilles (évitant des prédictions extrêmes).

5.2.3 Optimisation par Développement de Taylor

Contrairement au Gradient Boosting standard qui utilise uniquement le gradient (dérivée première), XGBoost exploite une approximation de second ordre de la fonction de perte via un développement de Taylor. Cette innovation mathématique permet une convergence plus rapide, plus stable et plus précise vers l'optimum.
L'objectif à l'étape t est approximé par :

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{6}$$

où g_i et h_i sont respectivement le gradient (dérivée première) et le hessien (dérivée seconde) de la fonction de perte par rapport à la prédiction actuelle :

$$g_i = \frac{\partial l(y_i, \hat{y}^{(t-1)})}{\partial \hat{y}^{(t-1)}} \quad \text{et} \quad h_i = \frac{\partial^2 l(y_i, \hat{y}^{(t-1)})}{\partial (\hat{y}^{(t-1)})^2} \tag{7}$$

Avantages de l'approximation de second ordre :

- Le gradient g_i indique la direction de descente (où aller)
- Le hessien h_i capture la courbure de la fonction de perte (à quelle vitesse y aller)
- Cette information de courbure permet des pas d'optimisation mieux calibrés
- Convergence plus rapide avec moins d'itérations nécessaires

Cette utilisation de l'information de second ordre (courbure de la fonction de perte) constitue l'une des clés de la supériorité de XGBoost sur les implémentations classiques

de Gradient Boosting.

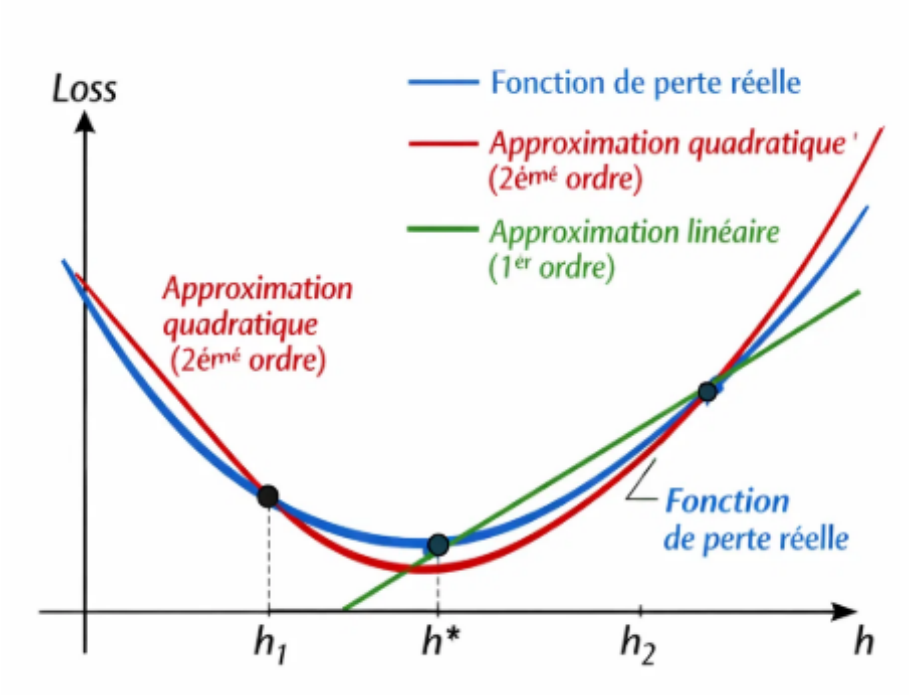


FIGURE 7 : Illustration du Développement de Taylor du Second Ordre.

5.2.4 Calcul du Score Structurel et Algorithme de Split

Une fois la structure de l'arbre fixée (c'est-à-dire l'affectation des données x_i aux feuilles j), le poids optimal w_j^* pour chaque feuille peut être calculé analytiquement en annulant la dérivée de la fonction objectif simplifiée :

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda} \tag{8}$$

où $I_j = \{i \mid x_i \in \text{feuille } j\}$ représente l'ensemble des indices des instances tombant dans la feuille j. Cette formule analytique garantit que chaque feuille produit la meilleure prédiction possible étant donnée la structure de l'arbre.

Cependant, trouver la structure d'arbre optimale globale est un problème NP-difficile (complexité exponentielle). XGBoost utilise donc un algorithme glouton (*Greedy Algorithm*) qui construit l'arbre de manière itérative en ajoutant des branches une par une. Pour évaluer la qualité d'une division (*split*) d'un nœud parent en deux nœuds enfants gauche (L) et droit (R), on utilise la formule de Gain suivante :

$$\text{Gain} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{9}$$

Interprétation détaillée du Gain :

- Les deux premiers termes représentent les scores des nœuds enfants (gauche et droit)
- Le troisième terme (négatif) représente le score du nœud parent non divisé
- Leur différence mesure la réduction de l'erreur apportée par la division
- Le terme $-\gamma$ agit comme un seuil de complexité (coût de complexification). Si le gain net apporté par le split est inférieur à γ, la division est rejetée (mécanisme de *pruning* ou élagage pré-emptif)
- Un γ élevé favorise des arbres plus simples et peu profonds

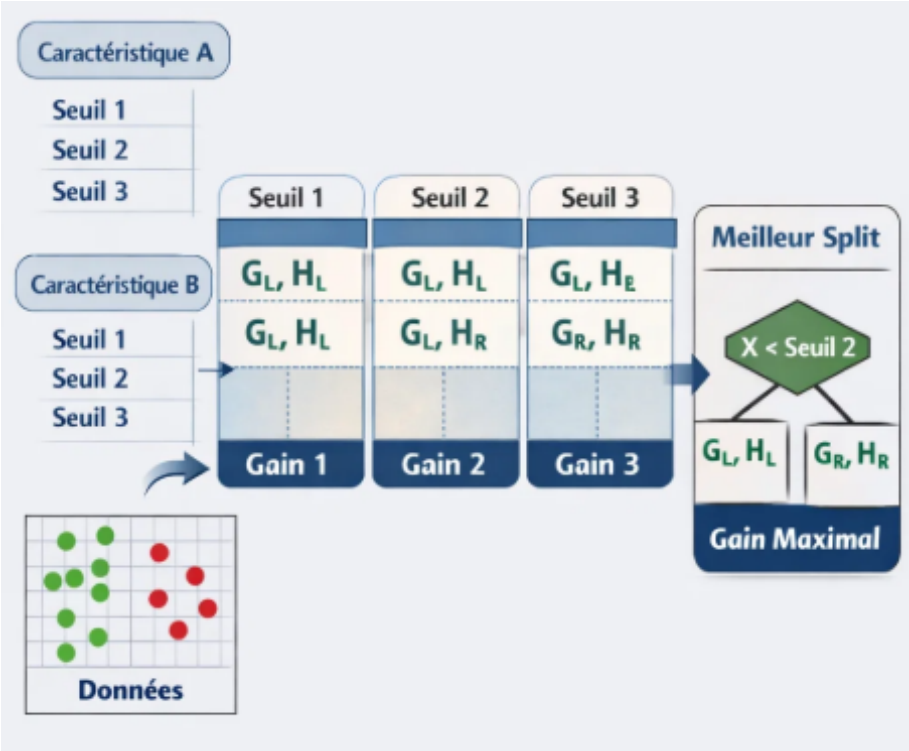


FIGURE 8 : Processus de recherche exhaustive du meilleur split

5.2.5 Déroulement de l'Algorithme (Algorithm Steps)

L'entraînement du modèle XGBoost suit les étapes itératives suivantes, formant un cycle d'amélioration continue :

1. **Initialisation :** Le modèle commence avec une prédiction constante, typiquement :
   - Pour la classification binaire : $\hat{y}_i^{(0)} = \log\left(\frac{p}{1-p}\right)$ où p est la proportion de la classe positive
   - Pour la régression : $\hat{y}_i^{(0)} = \bar{y}$ (moyenne des valeurs cibles)
2. **Calcul des Statistiques de Gradient :** Pour chaque itération $t = 1, 2, ..., K$:
   - Calculer le gradient $g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$ pour chaque instance i
   - Calculer le hessien $h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial (\hat{y}_i^{(t-1)})^2}$ pour chaque instance i
   - Ces statistiques encodent l'information sur l'erreur résiduelle du modèle actuel
3. **Construction de l'Arbre f_t :**
   - Partir d'un nœud racine contenant toutes les instances d'entraînement
   - Pour chaque nœud courant non terminal :
     (a) Énumérer tous les splits candidats (toutes les features et tous les seuils)
     (b) Calculer le Gain pour chaque split candidat selon l'équation (3.8)
     (c) Sélectionner le split avec le Gain maximal
     (d) Si Gain > 0, créer deux nœuds enfants, sinon transformer en feuille
   - Continuer jusqu'à atteindre la profondeur maximale (*max_ depth*) ou jusqu'à ce qu'aucun split ne soit bénéfique
   - Calculer les poids optimaux w_j^* pour toutes les feuilles terminales
4. **Mise à jour des Prédictions :**
   - Pour chaque instance i, mettre à jour la prédiction :

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta \cdot f_t(x_i) \tag{10}$$

     où $\eta \in (0, 1]$ est le taux d'apprentissage (*learning rate* ou *shrinkage*) qui contrôle la contribution de chaque arbre
   - Un η faible (ex : 0.1) rend l'apprentissage plus lent mais plus robuste
   - Un η élevé (ex : 0.3) accélère la convergence mais risque le surapprentissage

5. **Évaluation et Arrêt Anticipé :**
    - Évaluer la performance du modèle sur un ensemble de validation
    - Si la performance ne s'améliore pas pendant N itérations consécutives (*early stopping rounds*), arrêter l'entraînement prématurément
    - Sinon, répéter les étapes 2 à 4 jusqu'à atteindre le nombre maximal d'estimateurs K

5.2.6 Fonctionnalités Système Avancées

Outre son architecture mathématique sophistiquée, l'efficacité pratique de XGBoost repose sur des optimisations système cruciales pour notre projet, notamment :

- **Sparsity Awareness (Gestion intelligente des données creuses) :**
    - Nos données textuelles vectorisées (TF-IDF, Bag-of-Words) contiennent naturellement de nombreux zéros (matrices creuses)
    - XGBoost gère nativement ces valeurs manquantes sans besoin d'imputation
    - L'algorithme apprend automatiquement une "direction par défaut" optimale pour chaque nœud (envoyer les instances avec valeurs manquantes vers le nœud gauche ou droit)
    - Cette fonctionnalité réduit considérablement le temps de calcul et l'utilisation mémoire
    - Particulièrement pertinent pour notre corpus où la plupart des documents ne contiennent qu'une fraction du vocabulaire total
- **Parallelization (Calcul Parallèle Multi-Cœurs) :**
    - Bien que le boosting soit intrinsèquement séquentiel (un arbre après l'autre), la construction de chaque arbre individuel est hautement parallélisable
    - La recherche des meilleurs splits est parallélisée au niveau des caractéristiques (*features*)
    - Chaque thread traite un sous-ensemble de features simultanément
    - Utilisation optimale de tous les cœurs CPU disponibles (ou GPU avec l'extension appropriée)

  - Accélération significative sur les datasets avec de nombreuses features (notre cas avec TF-IDF)

- **Column Block Structure (Structure en Blocs de Colonnes) :**
  - Les données sont stockées en mémoire dans un format colonne compressé (CSC - Compressed Sparse Column)
  - Chaque feature est stockée dans un bloc séparé, trié par valeur
  - Cette organisation optimise les accès mémoire séquentiels et exploite le cache processeur
  - Réduction drastique du temps de recherche des meilleurs splits
  - Les blocs peuvent être pré-calculés et réutilisés à chaque itération

- **Cache-Aware Access Pattern :**
  - Optimisation des patterns d'accès mémoire pour maximiser l'utilisation du cache L1/L2/L3
  - Minimisation des défauts de cache (*cache misses*)
  - Impact majeur sur la vitesse d'entraînement pour des datasets de taille moyenne à grande

- **Out-of-Core Computing :**
  - Capacité à traiter des datasets qui ne tiennent pas en mémoire RAM
  - Division des données en blocs multiples stockés sur disque
  - Chargement progressif des blocs pendant l'entraînement
  - Utilisation de compression pour réduire les opérations I/O disque

Ces optimisations font de XGBoost un choix particulièrement adapté pour notre tâche de classification de textes, où nous manipulons des représentations vectorielles de haute dimension et creuses (TF-IDF), sur un dataset de taille conséquente nécessitant une efficacité computationnelle maximale.

5.2.7 Pipeline de Classification Multi-Label avec XGBoost

Pour appliquer XGBoost à notre tâche de classification multi-label d'articles de presse, nous avons mis en place un pipeline complet qui transforme les textes bruts en prédictions de catégories. Ce pipeline comprend plusieurs étapes de transformation des données et de configuration du modèle, chacune jouant un rôle dans l'adaptation de l'algorithme à notre problème spécifique.

1. **Simplification de l'Espace des Labels :**

   Notre dataset EuroLex contient 100 classes de concepts EuroVoc. Pour améliorer la stabilité du modèle et faciliter l'interprétation, nous avons effectué une agrégation hiérarchique de ces concepts en 14 catégories de haut niveau basées sur la structure taxonomique d'EuroVoc. Par exemple, les concepts "0406 political framework", "0431 politics and public safety" et "0436 executive power" sont regroupés sous la catégorie "Politics & Government". Cette transformation réduit la complexité du problème tout en préservant la nature multi-label, puisqu'un document peut toujours appartenir à plusieurs catégories simultanément.

   Les labels sont ensuite encodés en vecteurs binaires de dimension fixe (14) à l'aide d'un MultiLabelBinarizer, où chaque position correspond à une catégorie. Une valeur de 1 indique la présence de la catégorie, 0 son absence.

2. **Représentation Textuelle :**

   La transformation du texte en représentation numérique constitue l'étape centrale du pipeline. Nous avons adopté la vectorisation TF-IDF (Term Frequency-Inverse Document Frequency) qui quantifie l'importance relative de chaque terme dans un document par rapport à l'ensemble du corpus.

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log\left(\frac{N}{\text{DF}(t)}\right) \tag{11}$$

   Le vocabulaire est limité aux 50 000 termes les plus fréquents, incluant les unigrammes et bigrammes (ngram_range=(1,2)) pour capturer des expressions juridiques composées comme "executive power". Les mots-outils anglais et les termes apparaissant dans moins de 2 documents sont filtrés. La matrice TF-IDF résultante est extrêmement creuse, propriété qui sera exploitée par les optimisations de

XGBoost.

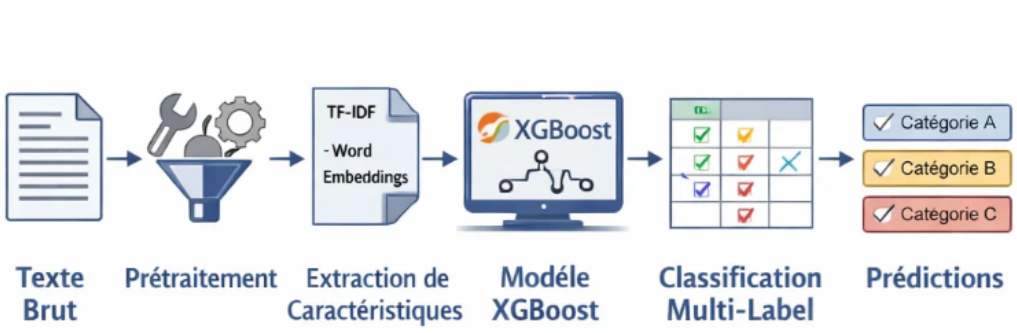


FIGURE 9 : Vue d'ensemble du pipeline XGBoost : du texte brut aux prédictions multi-label

Pour réduire la dimensionnalité et accélérer l'entraînement, nous appliquons une décomposition en valeurs singulières tronquée (Truncated SVD) qui projette les vecteurs TF-IDF de 50 000 dimensions dans un espace de 300 dimensions. Cette transformation préserve environ 80-85% de la variance tout en produisant une représentation dense plus efficace pour XGBoost.

3. **Gestion du Déséquilibre des Classes :**

   L'analyse de la distribution révèle un déséquilibre important : certaines catégories comme "Trade & Business" apparaissent dans plus de 60% des documents tandis que d'autres comme "Law & Justice" ne concernent que 2-3% des échantillons. Pour atténuer ce problème, nous avons exploré deux approches complémentaires.

   La première utilise des poids d'échantillons inversement proportionnels à la fréquence des labels. Pour chaque document, le poids est déterminé par la classe la plus rare qu'il contient, forçant le modèle à accorder plus d'importance aux exemples sous-représentés. La seconde approche applique un sur-échantillonnage des classes minoritaires : pour chaque catégorie ayant moins de 2000 échantillons, nous dupliquons aléatoirement des documents existants jusqu'à atteindre ce seuil.

4. **Configuration du Modèle :**

   Le modèle XGBoost est configuré avec 500 arbres de profondeur maximale 15, permettant de capturer des interactions complexes entre features. Le taux d'apprentissage de 0.1 assure une convergence stable, tandis que les paramètres de sous-

échantillonnage (subsample=0.8, colsample_bytree=0.8) introduisent de la stochasticité pour améliorer la robustesse.

Les termes de régularisation sont essentiels pour contrôler la complexité. Les paramètres reg_alpha=10 et reg_lambda=10 appliquent respectivement des pénalités L1 et L2 sur les poids des feuilles. Le paramètre gamma=0.2 définit le gain minimal requis pour une division, tandis que min_child_weight=2 exige un nombre minimal d'exemples dans chaque feuille. L'accélération GPU est activée via tree_method="hist" et device="cuda" pour réduire le temps d'entraînement sur les 55 000 documents.

TABLE 3 : Configuration des hyperparamètres principaux du modèle XGBoost

Paramètre	**Valeur**	**Rôle**
n_estimators	500	Nombre d'arbres dans l'ensemble
max_depth	15	Profondeur maximale de chaque arbre
learning_rate	0.1	Taux d'apprentissage
subsample	0.8	Fraction des données par arbre
reg_alpha, reg_lambda	10	Régularisation L1 et L2
gamma	0.2	Gain minimal pour une division

5. **Stratégie Multi-Label** Pour la classification multi-label, XGBoost est appliqué de manière indépendante à chaque label selon l'approche Binary Relevance, traitant le problème comme 14 tâches de classification binaire distinctes. Cette stratégie présente l'avantage de la simplicité mais ignore les corrélations entre labels. Nous avons également expérimenté avec des Classifier Chains, où les prédictions d'un label servent de features pour prédire le suivant, capturant ainsi les dépendances entre catégories.

6. **Approche Alternative : Embeddings GloVe**

   En complément de TF-IDF, nous avons exploré l'utilisation d'embeddings préentraînés GloVe qui capturent des relations sémantiques entre mots. Chaque mot est représenté par un vecteur dense de 300 dimensions, et un document est obtenu par la moyenne de ses vecteurs de mots :

$$\mathbf{v}_{\text{doc}} = \frac{1}{|D|} \sum_{w \in D} \mathbf{v}_w \tag{12}$$

   Cette représentation encode des similarités sémantiques (les mots "legislation" et

"regulation" ont des vecteurs proches) et offre une dimensionnalité plus compacte que TF-IDF. En revanche, elle ne tient pas compte de la spécificité des termes au corpus juridique et la simple moyenne peut perdre certaines nuances structurelles du texte.

5.2.8 Interprétation des Résultats

L'évaluation du modèle XGBoost sur notre tâche de classification multi-label révèle des performances contrastées selon les métriques et les catégories. Cette section analyse en détail les résultats obtenus sur l'ensemble de test, identifie les forces et faiblesses du modèle, et discute les différentes stratégies explorées pour améliorer les performances.

Performances Globales avec TF-IDF + SVD Le modèle baseline utilisant la représentation TF-IDF réduite par SVD à 300 dimensions atteint un score Micro F1 de 0.7623 et un Macro F1 de 0.6500 sur l'ensemble de test. La Hamming Loss, qui mesure la proportion d'étiquettes incorrectement prédites, se situe à 0.1004, indiquant qu'environ 10% des prédictions individuelles de labels sont erronées. L'accuracy stricte (Subset Accuracy), qui exige une correspondance exacte de tous les labels d'un document, n'atteint que 0.2634, reflétant la difficulté intrinsèque de prédire correctement toutes les catégories simultanément.
L'écart significatif entre le Micro F1 (0.9981) sur l'ensemble d'entraînement et celui sur le test (0.7623) révèle un surapprentissage prononcé. Le modèle mémorise efficacement les patterns de l'ensemble d'entraînement mais peine à généraliser à de nouveaux documents. Cet écart de 0.2358 points suggère que malgré les mécanismes de régularisation intégrés (gamma, reg_alpha, reg_lambda), la complexité du modèle reste excessive pour la taille et la diversité du corpus.

Analyse par Catégorie Les performances varient considérablement selon les catégories, révélant une structure hiérarchique dans la difficulté de classification. Trois catégories se distinguent par d'excellents résultats : "Agriculture & Food" (F1 = 0.9613), "Geography & Regional" (F1 = 0.8624) et "Trade & Business" (F1 = 0.8245). Ces catégories bénéficient d'une forte représentation dans l'ensemble d'entraînement (respectivement 67.7%, 42.4% et 60.9% des documents) et possèdent probablement un vocabulaire spécifique bien délimité facilitant leur identification.

À l'opposé, trois catégories affichent des performances préoccupantes. "Law & Justice" obtient le score le plus faible avec un F1 de 0.3103, malgré un score d'entraînement de 0.9893. Cette chute drastique de 0.6789 points entre entraînement et test indique un surapprentissage sévère sur cette catégorie rare (2.4% des documents d'entraînement). "Employment & Labor" (F1 = 0.4397) et "Energy & Resources" (F1 = 0.4958) souffrent également de performances insuffisantes, corrélées à leur faible fréquence dans le corpus.

TABLE 4 : Performances par catégorie et corrélation avec la fréquence

Catégorie	**Fréquence (%)**	**F1 Test**	**Surapprentissage**
Agriculture & Food	67.7	0.9613	Faible (0.04)
Geography & Regional	42.4	0.8624	Faible (0.14)
Trade & Business	60.9	0.8245	Modéré (0.17)
Law & Justice	2.4	0.3103	Sévère (0.68)
Employment & Labor	3.4	0.4397	Sévère (0.55)
Energy & Resources	2.0	0.4958	Sévère (0.48)

L'analyse de corrélation confirme un biais fort du modèle envers les classes fréquentes. Le coefficient de corrélation entre la fréquence d'une catégorie et son score F1 sur le test atteint 0.8434, indiquant que 84% de la variance des performances s'explique par la simple fréquence des classes. Ce phénomène est caractéristique des problèmes de classification déséquilibrée où le modèle optimise la métrique globale en favorisant les classes majoritaires.

Impact de la Pondération des Classes L'introduction de poids d'échantillons inversement proportionnels à la fréquence des labels constitue notre première stratégie pour atténuer le déséquilibre. Chaque document reçoit un poids correspondant à sa classe la plus rare, forçant le modèle à accorder plus d'attention aux exemples minoritaires pendant l'entraînement.

Cette approche améliore légèrement le Macro F1 de 0.6500 à 0.6698 (+0.0198), indiquant une meilleure prise en compte des classes rares. Certaines catégories bénéficient particulièrement de cette stratégie : "Law & Justice" passe de 0.3103 à 0.4000, et "Energy & Resources" de 0.4958 à 0.5339. Cependant, ces gains restent modestes et ne résolvent pas fondamentalement le problème de surapprentissage. L'écart train-test diminue légèrement à 0.1994 mais demeure significatif.

Le compromis principal de cette méthode réside dans la diminution de l'accuracy stricte (de 0.2634 à 0.2498), suggérant que le modèle prédit désormais plus de labels par do-

cument, augmentant les faux positifs sur les classes majoritaires pour mieux couvrir les minoritaires.

Augmentation de Données par Sur-échantillonnage La seconde stratégie consiste à augmenter artificiellement le dataset en dupliquant aléatoirement les documents des classes sous-représentées. Les trois catégories ayant moins de 2000 échantillons ("Law & Justice", "Employment & Labor", "Energy & Resources") sont sur-échantillonnées jusqu'à ce seuil, portant le dataset de 55 000 à 56 675 documents.
Cette augmentation produit une légère amélioration du Micro F1 à 0.7841 (+0.0093 par rapport au baseline), mais dégrade le Macro F1 à 0.6614. Ce résultat paradoxal s'explique par un phénomène connu : la simple duplication d'exemples ne crée pas de diversité réelle et peut encourager le modèle à mémoriser les instances rares plutôt qu'à apprendre leurs patterns généralisables. "Law & Justice" régresse même à 0.2876, pire que le baseline.
L'échec relatif de cette approche confirme que le problème ne réside pas uniquement dans le déséquilibre quantitatif mais aussi dans la qualité et la diversité des représentations textuelles des classes rares.

Classifier Chains et Optimisation des Seuils L'approche Binary Relevance standard ignore les corrélations potentielles entre labels. Les Classifier Chains modélisent ces dépendances en utilisant les prédictions de labels précédents comme features supplémentaires pour prédire les suivants. De plus, nous adaptons les seuils de décision par classe : 0.35 au lieu de 0.5 pour "Law & Justice" et "Employment & Labor".
Cette combinaison maintient un Micro F1 de 0.7695, légèrement inférieur au baseline, avec un Macro F1 de 0.6556. Les seuils abaissés améliorent marginalement les classes ciblées ("Law & Justice" : 0.3736, "Employment & Labor" : 0.4211) mais introduisent davantage de faux positifs. L'accuracy stricte reste stable à 0.2558.
Les Classifier Chains n'apportent donc pas de gain significatif dans notre contexte, possiblement parce que les corrélations entre catégories juridiques sont complexes et variables d'un document à l'autre, rendant difficile l'apprentissage de dépendances généralisables.

Représentation Alternative : Embeddings GloVe L'utilisation d'embeddings GloVe pré-entraînés remplace la représentation TF-IDF par des vecteurs denses de 300 dimensions capturant des similarités sémantiques. Contrairement à TF-IDF qui encode la spé-

cificité des termes au corpus, GloVe repose sur des connaissances linguistiques générales apprises sur Wikipedia.
Cette représentation alternative dégrade les performances : Micro F1 de 0.7909 et Macro F1 de 0.6457. Plus préoccupant encore, l'écart train-test explose à 0.2624, indiquant un surapprentissage encore plus sévère. Ce résultat s'explique par deux facteurs. Premièrement, le vocabulaire juridique spécialisé d'EuroLex n'est pas bien représenté dans GloVe entraîné sur des textes généraux. Deuxièmement, la simple moyenne des embeddings de mots perd l'information de pondération TF-IDF qui accentue les termes discriminants.
Les catégories rares souffrent particulièrement : "Law & Justice" chute à 0.2597 et "Employment & Labor" à 0.2917, confirmant que les embeddings génériques ne compensent pas le manque de données d'entraînement pour ces classes.

Limites et Perspectives Les résultats de XGBoost révèlent plusieurs limites fondamentales pour notre tâche. Le surapprentissage systématique, même avec régularisation forte, suggère que les arbres de décision captent des patterns spécifiques aux documents d'entraînement plutôt que des règles linguistiques générales. Le biais envers les classes fréquentes persiste malgré les techniques de rééquilibrage, indiquant que le problème nécessite des représentations plus riches que les simples statistiques de mots.
La représentation TF-IDF, bien qu'efficace pour les classes majoritaires, ne capte pas les nuances sémantiques et contextuelles essentielles pour distinguer des concepts juridiques proches. Par exemple, "Law & Justice" et "Politics & Government" partagent un vocabulaire substantiel, rendant leur séparation difficile sans compréhension du contexte.
Ces observations motivent l'exploration de modèles de Deep Learning dans la suite de ce travail. Les architectures neuronales, notamment les Transformers, peuvent apprendre des représentations contextualisées des mots et modéliser des dépendances à longue distance dans les documents. Leur capacité à bénéficier de pré-entraînement sur de vastes corpus juridiques pourrait également atténuer le problème de rareté de certaines catégories.

5.3 Bidirectional Encoder Representations from Transformers (BERT)

5.3.1 Processus d'entraînement des encodeurs

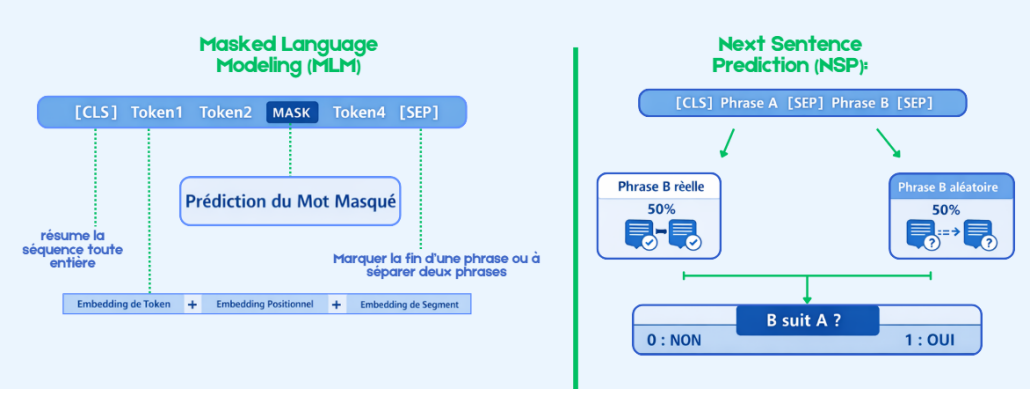


FIGURE 10 : Entraînement des encodeurs

Les encodeurs de type BERT reposent sur une phase de pré-entraînement réalisée à partir de grandes quantités de texte brut, sans recours à des annotations manuelles. Cette approche s'inscrit dans un cadre d'apprentissage auto-supervisé, où le modèle apprend des structures linguistiques à partir du contexte même des données. Le pré-entraînement de BERT repose sur deux objectifs principaux qui sont optimisés simultanément, permettant au modèle d'acquérir une compréhension profonde et bidirectionnelle du langage naturel. Le premier objectif est le *Masked Language Modeling* (MLM). Dans cette approche, une phrase est sélectionnée et environ quinze pour cent de ses mots sont masqués de manière aléatoire. Le rôle du modèle consiste alors à prédire les mots masqués en s'appuyant sur le contexte global de la phrase, aussi bien à gauche qu'à droite du mot manquant. Cette caractéristique permet à BERT de développer une compréhension bidirectionnelle du langage, contrairement aux modèles séquentiels traditionnels qui ne considèrent qu'un seul sens de lecture.

Les phrases d'entrée sont transformées en une séquence de tokens, incluant des tokens spéciaux. La structure typique d'une entrée est la suivante : `[CLS] token_1 token_2 [MASK] token_4 [SEP]`. Chaque token est ensuite converti en une représentation vectorielle obtenue par la somme de trois embeddings distincts. Le premier est le *token embedding*, qui représente l'identité du mot. Le deuxième est le *position embedding*, qui encode la position du mot dans la séquence afin de conserver l'ordre des mots. Le troisième est le *segment embedding*, utilisé pour distinguer les phrases lorsqu'une entrée contient plu-

sieurs segments. Les tokens `[MASK]` permettent de masquer les mots à prédire, tandis que le token `[SEP]` marque la fin d'une phrase ou la séparation entre deux phrases. Le token spécial `[CLS]`, placé au début de chaque séquence, joue un rôle central puisqu'il sert à résumer l'information globale de la séquence.

Le second objectif du pré-entraînement est le *Next Sentence Prediction* (NSP), qui vise à apprendre au modèle la relation logique entre deux phrases. Durant l'entraînement, le modèle reçoit une paire de phrases A et B. Dans cinquante pour cent des cas, la phrase B suit réellement la phrase A dans le texte original. Dans les cinquante pour cent restants, la phrase B est choisie de manière aléatoire et ne suit pas la phrase A. Le modèle doit alors prédire si la phrase B est bien la continuation logique de la phrase A. Les deux phrases sont concaténées sous la forme `[CLS] Phrase A [SEP] Phrase B [SEP]`. Le vecteur associé au token `[CLS]` est utilisé par une couche de classification finale afin de produire une sortie binaire indiquant si la phrase B suit ou non la phrase A.

5.3.2 Fine-tuning des encodeurs

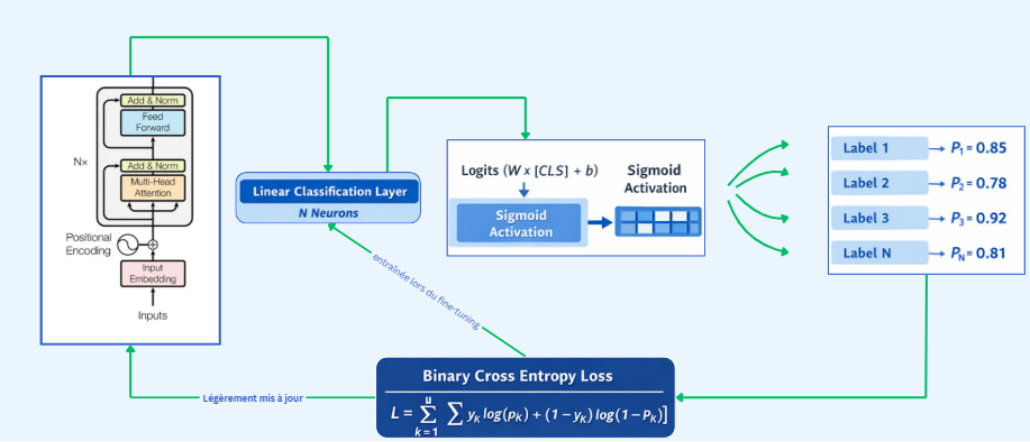


FIGURE 11 : Fine-tuning des encodeurs

À l'issue de la phase de pré-entraînement, le modèle BERT possède une connaissance générale du langage, incluant la syntaxe, la sémantique et les relations contextuelles entre les mots et les phrases. Cette connaissance est ensuite exploitée dans une phase de fine-tuning, où le modèle est adapté à une tâche spécifique à l'aide d'un jeu de données annoté. Dans le cas de la classification multi-label, une couche de classification est ajoutée à la sortie du modèle pré-entraîné. Cette couche est généralement une couche linéaire contenant un nombre de neurones égal au nombre total de labels à prédire. Le vecteur de sortie

associé au token `[CLS]` est utilisé comme représentation globale du texte d'entrée. Les logits sont calculés selon la formule suivante : logits $= W \times \text{embedding}_{[CLS]} + b$, où le vecteur obtenu est de taille N, correspondant au nombre de labels.
Une fonction sigmoïde est ensuite appliquée indépendamment à chaque logit, ce qui permet d'obtenir une probabilité pour chaque label. Contrairement à la classification multi-classe, plusieurs labels peuvent ainsi être activés simultanément avec des probabilités élevées. La fonction de perte utilisée est la *Binary Cross Entropy*, où chaque label est traité comme un problème de classification binaire indépendant.
La couche de classification multi-label est entraînée à partir de zéro, tandis que les paramètres du modèle BERT sont ajustés de manière légère durant le fine-tuning. Cette stratégie permet de conserver les connaissances linguistiques générales acquises lors du pré-entraînement tout en spécialisant le modèle pour la tâche cible, ce qui conduit à de meilleures performances même avec des jeux de données annotés de taille limitée.

5.3.3 Architecture du Modèle : BERT-Base

L'architecture choisie est BERT (Bidirectional Encoder Representations from Transformers), spécifiquement la version bert-base-uncased. Contrairement aux modèles de langage traditionnels qui lisent le texte de gauche à droite ou de droite à gauche, BERT analyse le texte dans les deux sens simultanément grâce à son mécanisme d'attention.

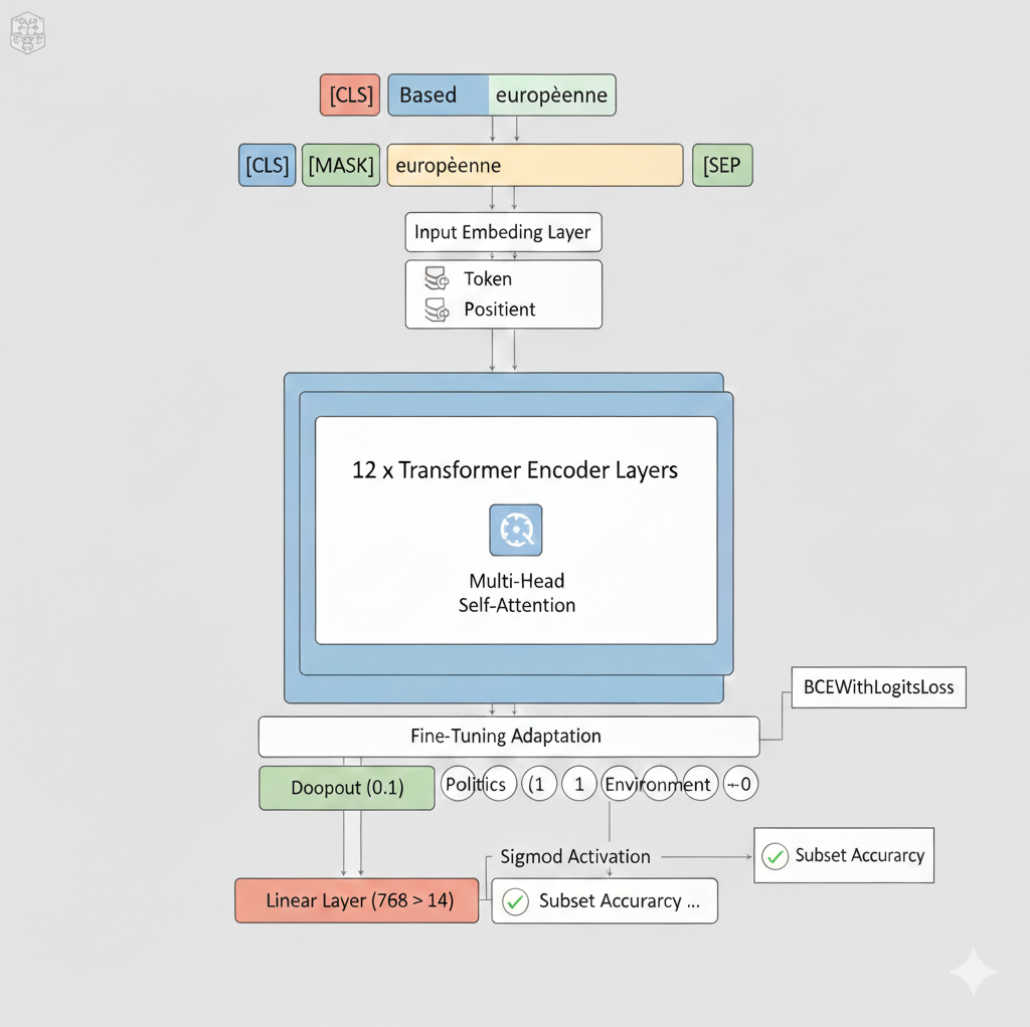


FIGURE 12 : Architecture du modèle BERT Base

Structure interne : L'architecture BERT-Base se compose de trois blocs principaux :

- **Couche d'Embedding (Entrée) :** Elle transforme les tokens en vecteurs de dimension 768. Elle combine trois types d'informations : les vecteurs de mots (Token Embeddings), les vecteurs de position (Position Embeddings) et les types de segments.

- **Encodeurs Transformers (Le cœur) :** BERT-Base contient 12 couches d'encodeurs empilées. Chaque couche utilise un mécanisme de Multi-Head Self-Attention (12 têtes) qui permet au modèle de comprendre les relations de dépendance entre tous les mots d'une phrase, quelle que soit leur distance.

- **Le Pooler (Sortie) :** Pour la classification, BERT utilise un token spécial inséré au début de chaque séquence : le token [CLS]. La représentation finale de ce token

sert de résumé sémantique de toute la phrase.

5.3.4 Adaptation Multi-Label (BertMultiLabelClassifier)

- **Dropout** : Une couche de désactivation aléatoire (10%) est ajoutée pour éviter le surapprentissage (overfitting).
- **Couche Linéaire :** Une couche finale projette la sortie de dimension 768 vers les 14 unités correspondant à vos catégories.
- **Fonction d'activation Sigmoid :** Contrairement à la classification classique (Softmax), on utilise ici une Sigmoid sur chaque sortie. Cela permet à chaque classe d'être prédite indépendamment avec une probabilité entre 0 et 1.

5.3.5 Pipeline de l'Entraînement

Le pipeline d'entraînement a été conçu pour maximiser l'efficacité sur les GPU T4 de Kaggle en utilisant des techniques d'optimisation modernes.

- **Optimiseur : AdamW.** C'est une variante de l'algorithme Adam qui gère mieux la régularisation (Weight Decay), essentielle pour éviter que les poids du modèle ne deviennent trop importants.
- **Scheduler : Cosine Annealing.** Le taux d'apprentissage (Learning Rate) ne reste pas fixe. Il suit une courbe cosinus : il commence fort pour apprendre vite, puis diminue progressivement vers zéro pour affiner les poids en fin d'entraînement.
- **Loss Function : BCEWithLogitsLoss.** Cette fonction combine une couche Sigmoid et une perte d'entropie croisée binaire. Elle est mathématiquement plus stable que l'application manuelle d'une Sigmoid suivie d'une BCE.

5.3.6 Résultats

Métriques	
F1 Score (micro)	**0.8572**
Precision	**0.8797**
Recall	**0.8358**
Hamming Loss	**0.0644**
Subset Accuracy	**0.4318**
F1 Score (macro)	**0.7864**

TABLE 5 : Métriques de Validation

Métriques	
F1 Score (micro)	0.7800
Precision	0.8005
Recall	0.7606
Hamming Loss	0.0703
Subset Accuracy	0.3929

TABLE 6 : Métriques de Test

5.4 RoBERTa : Robustly Optimized BERT Approach

RoBERTa, acronyme de *Robustly Optimized BERT Approach*, est un modèle proposé par Facebook AI Research. Contrairement à ce que son nom pourrait suggérer, RoBERTa ne constitue pas une nouvelle architecture de réseau de neurones. Il repose exactement sur la même architecture que BERT, à savoir un Transformer utilisant uniquement la partie encodeur. Le nombre de couches, la taille des embeddings, le nombre de têtes d'attention ainsi que le nombre total de paramètres sont identiques à ceux de BERT. L'attention reste bidirectionnelle, permettant au modèle de capturer le contexte global d'une séquence de texte.

L'amélioration apportée par RoBERTa réside principalement dans la manière dont le modèle est entraîné. Les auteurs ont montré que certaines hypothèses de conception de BERT, notamment l'utilisation de l'objectif Next Sentence Prediction et certaines contraintes liées aux données et à la stratégie d'entraînement, limitaient les performances finales du modèle. RoBERTa a donc été entraîné selon une stratégie optimisée, mettant l'accent sur un apprentissage plus robuste et une meilleure exploitation des données disponibles.

L'une des différences majeures entre BERT et RoBERTa concerne les objectifs d'entraînement. Alors que BERT combine le Masked Language Modeling et le Next Sentence Prediction, RoBERTa supprime complètement l'objectif NSP et se concentre uniquement sur le Masked Language Modeling. Cette suppression a permis d'améliorer la stabilité de l'entraînement et d'obtenir de meilleures performances sur de nombreuses tâches en aval, suggérant que l'objectif NSP n'était pas indispensable à l'apprentissage de représentations linguistiques efficaces.

Une autre amélioration importante concerne les données d'entraînement. BERT a été entraîné sur un corpus relativement limité d'environ seize gigaoctets de texte, tandis que RoBERTa exploite un ensemble de données beaucoup plus vaste, atteignant environ cent soixante gigaoctets. Cette augmentation significative du volume de données permet au modèle d'apprendre des structures linguistiques plus variées et plus riches, ce qui améliore sa capacité de généralisation.

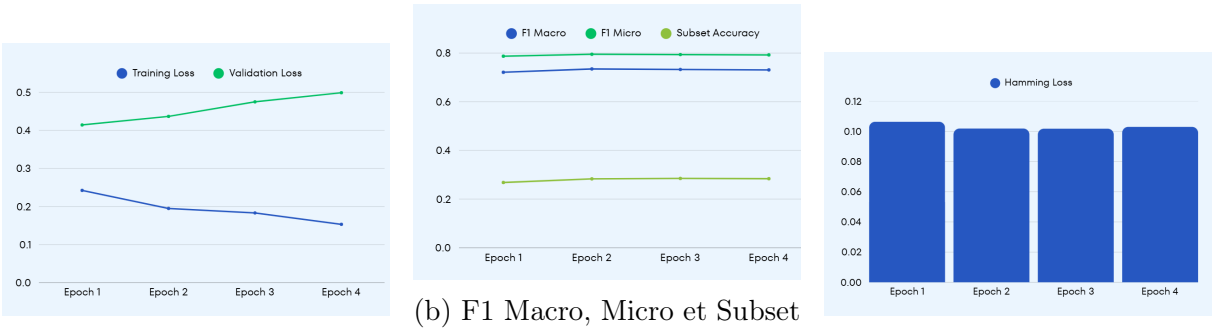
RoBERTa adopte également une stratégie de masquage dynamique. Contrairement à BERT, où le masquage des tokens est effectué une seule fois avant l'entraînement et reste fixe durant toutes les époques, RoBERTa applique un masquage différent à chaque passage des données. Cette approche empêche le modèle de mémoriser les positions masquées et favorise un apprentissage plus robuste des dépendances contextuelles.

En outre, RoBERTa est entraîné avec des tailles de batch beaucoup plus grandes et sur une durée plus longue que BERT. Ces choix contribuent à une meilleure convergence du modèle, à une stabilité accrue de l'apprentissage et à des performances supérieures sur la majorité des tâches de traitement automatique du langage naturel, au prix toutefois d'un coût de calcul plus élevé.

Critère	**BERT**	**RoBERTa**
Type de modèle	Transformer avec encodeur uniquement	Transformer avec encodeur uniquement
Architecture	Architecture originale de BERT	Identique à BERT
Mécanisme d'attention	Attention bidirectionnelle	Attention bidirectionnelle

Critère	**BERT**	**RoBERTa**
Objectifs d'entraînement	Masked Language Modeling et Next Sentence Prediction	Masked Language Modeling uniquement
Next Sentence Prediction	Utilisée durant le pré-entraînement	Supprimée
Données d'entraînement	Environ 16 Go de texte	Environ 160 Go de texte
Stratégie de masquage	Masquage statique effectué une seule fois	Masquage dynamique à chaque itération
Taille des batchs	Relativement petite	Beaucoup plus grande
Durée d'entraînement	Plus courte	Plus longue
Stabilité de l'apprentissage	Bonne	Meilleure
Performances globales	Bonnes	Supérieures sur la plupart des tâches
Robustesse	Moyenne	Plus robuste
Capacité de généralisation	Correcte	Améliorée
Coût de calcul	Plus faible	Plus élevé

5.4.1 Model Evaluation



(a) Training vs Validation Loss

(b) F1 Macro, Micro et Subset Accuracy

(c) Hamming Loss

FIGURE 13 : Évolution des métriques d'évaluation au cours des époques

TABLE 8 : Évolution des métriques d'entraînement et de validation au fil des epochs

Epoch	Train Loss	Val Loss	F1 Macro	F1 Micro	Hamming Loss	Subset Acc.	Accuracy
1	0.2424	0.4144	0.7209	0.7870	0.1064	0.2682	0.8936
2	0.1949	0.4369	**0.7346**	**0.7949**	**0.1019**	0.2831	0.8981
3	0.1832	0.4751	0.7326	0.7937	0.1018	**0.2850**	**0.8982**
4	0.1532	0.4991	0.7307	0.7923	0.1030	0.2838	0.8970

L'analyse des métriques d'entraînement et de validation permet d'évaluer le comportement du modèle au cours des différentes époques et d'identifier d'éventuels phénomènes de sur-apprentissage.

Tout d'abord, la perte d'entraînement diminue de manière régulière tout au long des epochs. Elle passe d'environ 0,24 à l'epoch 1 à 0,15 à l'epoch 4. Cette diminution continue indique que le modèle apprend efficacement à ajuster ses paramètres afin de mieux prédire les labels sur le jeu d'entraînement.

Concernant la Hamming Loss, on observe également une amélioration progressive au début de l'entraînement. Elle diminue de 0,106 à l'epoch 1 pour atteindre environ 0,101 à l'epoch 3, ce qui signifie que la proportion moyenne de labels mal prédits diminue. Une légère remontée est toutefois constatée à l'epoch 4, suggérant que les décisions finales du modèle ne s'améliorent plus malgré la baisse continue de la perte d'entraînement.

En revanche, la loss de validation suit une tendance différente. Elle diminue initialement à l'epoch 1, atteint un minimum autour des epochs 1 et 2, puis augmente progressivement aux epochs 3 et 4. Cette divergence entre la perte d'entraînement et la perte de validation est caractéristique d'un début de sur-apprentissage. À partir de l'epoch 3, le modèle continue à mieux s'adapter aux données d'entraînement, mais sa capacité de généralisation sur des données non vues commence à se dégrader.

En ce qui concerne les métriques F1, le F1 micro et le F1 macro s'améliorent entre l'epoch 1 et l'epoch 2. L'epoch 2 représente un bon compromis, avec un F1 macro d'environ 0,73 et un F1 micro proche de 0,79. À partir de l'epoch 3, ces deux métriques diminuent légèrement, confirmant que les gains supplémentaires en apprentissage ne se traduisent plus par une amélioration des performances globales.

La Subset Accuracy évolue de 26,8 % à l'epoch 1 jusqu'à environ 28,5 % à l'epoch 3, puis se stabilise. Bien que cette valeur puisse paraître faible, elle ne signifie pas que les performances du modèle sont médiocres. En effet, la Subset Accuracy est une métrique très stricte en classification multi-label, car elle exige que l'ensemble des labels d'un exemple soit prédit correctement. Une seule erreur suffit à considérer la prédiction comme incor-

recte. Dans ce contexte, une valeur proche de 28 % reste cohérente et acceptable.
Dans l'ensemble, l'analyse des métriques d'entraînement et de validation montre que l'epoch 2, voire l'epoch 3, correspond au point optimal avant l'apparition d'un sur-apprentissage plus marqué.

TABLE 9 : Résultats globaux du modèle sur le jeu de test

Métrique	**Valeur**
F1 Micro	0.7595
F1 Macro	0.7047
F1 Pondéré	0.7694
Samples Avg F1	0.7593

TABLE 10 : Scores de classification par classe sur le jeu de test

Classe	**Précision**	**Rappel**	**F1-score**
Politics & Government	0.6410	0.8039	0.7132
International Affairs & Defense	0.7031	0.7059	0.7045
Law & Justice	0.3540	0.6251	0.4520
Economics & Finance	0.8423	0.8310	0.8366
Trade & Business	0.8872	0.8130	0.8485
Employment & Labor	0.3773	0.6776	0.4847
Social Affairs & Health	0.5145	0.6916	0.5901
Technology & Science	0.5831	0.7147	0.6422
Transportation	0.8008	0.7626	0.7812
Environment	0.4998	0.6713	0.5730
Agriculture & Food	0.9103	0.8801	0.8949
Energy & Resources	0.7042	0.7133	0.7088
Industry & Manufacturing	0.7538	0.7602	0.7570
Geography & Regional	0.9201	0.8418	0.8792

L'évaluation finale sur le jeu de test permet de mesurer la capacité de généralisation du modèle. Les résultats obtenus montrent un F1 micro d'environ 0,76 et un F1 macro d'environ 0,70. L'écart entre ces deux métriques indique que le modèle est globalement plus performant sur les classes les plus fréquentes que sur les classes minoritaires, ce qui est un comportement attendu dans un contexte de données déséquilibrées.
L'analyse par classe met en évidence d'excellentes performances pour des catégories telles que Agriculture and Food, Trade and Business, Geography and Regional, ainsi que Economics and Finance, avec des scores F1 supérieurs à 0,85. Ces résultats s'expliquent par la forte représentativité de ces classes dans le jeu de données et par l'existence d'un vocabulaire spécifique facilitant leur identification.

À l'inverse, des classes comme Law and Justice ou Employment and Labor présentent des performances plus faibles. Ces catégories sont moins fréquentes dans le dataset, ce qui limite la quantité d'informations disponibles pour le modèle durant l'entraînement. Malgré l'utilisation d'une fonction de perte pondérée pour atténuer le déséquilibre des classes, ces labels restent plus difficiles à prédire de manière précise.
En conclusion, les résultats sur le jeu de test confirment que le modèle offre de bonnes performances globales, tout en mettant en évidence des axes d'amélioration possibles, notamment pour les classes minoritaires.

6 Comparaison des Modèles

Après avoir présenté individuellement les quatre modèles retenus, cette section propose une comparaison systématique de leurs caractéristiques et performances pour identifier les forces et faiblesses de chaque approche.

6.1 Critères de Comparaison

La comparaison s'articule autour de trois axes principaux.

Architecture et Paradigme d'Apprentissage Ce critère examine les fondements théoriques de chaque modèle : le paradigme d'apprentissage (superficiel versus profond), la nature des représentations (statiques, denses, contextuelles), et les mécanismes de régularisation. Cette dimension permet de comprendre pourquoi certains modèles capturent mieux certains patterns linguistiques.

Performance Prédictive L'évaluation repose sur les métriques de classification multi-label. Le Macro F1 constitue notre métrique principale car il équilibre la performance sur toutes les catégories indépendamment de leur fréquence. Le Micro F1 et l'accuracy stricte complètent l'analyse.

Complexité Computationnelle Ce critère englobe le temps d'entraînement, les ressources mémoire requises, la taille du modèle final, et la vitesse d'inférence. Ces considérations sont déterminantes pour le déploiement en production.

6.2 Comparaison en Termes d'Architecture

Les quatre modèles représentent une progression dans la sophistication des approches de classification textuelle, du plus simple au plus complexe.

SVM : Séparation Linéaire dans l'Espace Vectoriel Le Support Vector Machine représente l'approche classique de classification basée sur la recherche d'hyperplans séparateurs optimaux dans l'espace des features. Son architecture se limite à une transformation linéaire des vecteurs d'entrée vers l'espace de décision. Le noyau linéaire utilisé impose des frontières de décision linéaires, ce qui restreint sa capacité à modéliser des relations complexes entre features.
L'architecture repose fondamentalement sur une représentation "sac de mots" où chaque terme contribue indépendamment à la décision, sans considération de l'ordre ou du contexte.
La stratégie OneVsRest décompose le problème multi-label en problèmes binaires indépendants, chaque classificateur traitant une catégorie isolément. Cette indépendance architecturale empêche la modélisation des corrélations entre labels.

XGBoost : Architecture d'Ensemble Séquentielle XGBoost adopte une architecture radicalement différente basée sur l'agrégation séquentielle de modèles faibles. Chaque composant est un arbre de décision CART qui partitionne récursivement l'espace des features selon des règles conditionnelles. Cette structure arborescente permet naturellement de capturer des interactions non-linéaires et des dépendances entre features sans transformation explicite.
L'architecture d'ensemble fonctionne selon un principe additif : chaque nouvel arbre est construit pour corriger les erreurs résiduelles du modèle combiné précédent. Cette séquentialité contraste avec les forêts aléatoires où les arbres sont indépendants. Le mécanisme de Gradient Boosting guide cette construction en optimisant directement la fonction de perte via l'information du gradient et du hessien.
La régularisation est architecturalement intégrée à plusieurs niveaux : contraintes sur la profondeur des arbres, pénalités sur le nombre de feuilles (gamma), et régularisation L1/L2 sur les poids des feuilles. L'approche Binary Relevance reste utilisée, maintenant l'indépendance entre les classificateurs de labels.

BERT : Architecture Transformer avec Auto-Attention BERT marque une rupture architecturale en introduisant le paradigme Transformer basé entièrement sur des mécanismes d'attention. L'unité fondamentale est la couche d'auto-attention multi-têtes qui permet à chaque token de calculer des représentations pondérées de tous les autres tokens du document. Cette connectivité complète contraste radicalement avec les architectures précédentes où les mots sont traités isolément.

L'architecture se compose d'un empilement de couches d'encodeur identiques, chacune contenant deux sous-modules : une couche d'auto-attention multi-têtes suivie d'un réseau feed-forward. Les connexions résiduelles et la normalisation de couche assurent la stabilité de l'entraînement profond. La bidirectionnalité de BERT signifie que chaque token est encodé en tenant compte simultanément du contexte gauche et droit, ce qui est impossible avec les approches TF-IDF statiques.

Le pré-entraînement constitue un élément architectural clé : BERT est d'abord entraîné sur des tâches auto-supervisées (Masked Language Modeling, Next Sentence Prediction) avant d'être adapté à la classification. Cette séparation permet de tirer parti de vastes corpus non annotés. Pour la classification multi-label, une couche linéaire dense est ajoutée au-dessus de la représentation du token spécial [CLS], servant d'encodage global du document.

RoBERTa : Architecture Transformer Optimisée RoBERTa partage l'architecture Transformer exacte de BERT : même nombre de couches, mêmes dimensions cachées, même mécanisme d'attention. Les différences se situent au niveau du protocole de pré-entraînement plutôt que de la structure du réseau. L'élimination de la tâche Next Sentence Prediction simplifie l'architecture de pré-entraînement en ne conservant que le Masked Language Modeling.

Le tokenizer Byte-Pair Encoding de RoBERTa utilise un vocabulaire plus étendu, permettant une granularité supérieure dans la décomposition des mots. Cette différence, bien que technique, influence la capacité du modèle à traiter des termes spécialisés ou rares. Le masquage dynamique, qui change les tokens masqués à chaque passage, peut être vu comme une forme d'augmentation de données au niveau architectural.

TABLE 11 : Synthèse Comparative des Architectures

Aspect	**SVM**	**XGBoost**	**BERT**	**RoBERTa**
Paradigme	Apprentissage superficiel	Méthode d'ensemble	Apprentissage profond	Apprentissage profond
Structure	Hyperplan linéaire	Arbres séquentiels	12 couches Transformer	12 couches Transformer
Type de données	Vectorielles tabulaires	Vectorielles tabulaires	Séquences textuelles	Séquences textuelles
Représentation	Statique vectorielle	Statique dense	Contextuelle (768 dim)	Contextuelle (768 dim)
Mécanisme clé	Maximisation de marge	Gradient boosting	Auto-attention multi-têtes	Auto-attention multi-têtes
Non-linéarité	Limitée (noyau)	Forte (partitions)	Maximale (attention)	Maximale (attention)
Contextualisation	Absente	Absente	Bidirectionnelle complète	Bidirectionnelle complète
Pré-entraînement	Non applicable	Non applicable	Masked LM + NSP	Masked LM optimisé
Multi-label	OneVsRest	Binary Relevance	Couche dense partagée	Couche dense partagée

6.3 Comparaison des Résultats d'Entraînement

Cette sous-section analyse comparativement les performances des quatre modèles sur les ensembles d'entraînement, de validation et de test, en se concentrant sur quatre métriques clés de la classification multi-label : le *Hamming Loss*, le *F1-score Macro*, le *F1-score Micro* et la *Subset Accuracy*. Ces métriques offrent une vision complémentaire de la qualité prédictive, allant de l'évaluation fine au niveau des labels jusqu'à l'exactitude stricte des prédictions complètes.

Analyse Globale des Performances Les résultats montrent une hiérarchie claire entre les approches classiques basées sur TF-IDF et les modèles profonds de type Transformer. Le modèle SVM avec Classifier Chain et optimisation par recherche de grille présente des performances solides et équilibrées, notamment en termes de généralisation. Il atteint sur l'ensemble de test un F1 Micro de 0.8464 et un F1 Macro de 0.763, avec une Subset Accuracy de 0.4704, ce qui en fait le meilleur modèle classique en termes de compromis biais-variance.

XGBoost affiche des performances très élevées sur l'ensemble d'entraînement (F1 Micro = 0.9742, Subset Accuracy = 0.8692), traduisant une forte capacité d'apprentissage.

Toutefois, la dégradation marquée observée sur les ensembles de validation et de test (Subset Accuracy chutant à 0.2498 et Hamming Loss atteignant 0.0971) indique un sur-apprentissage significatif, malgré l'utilisation de pondérations de classes et de réduction dimensionnelle par SVD.
Les modèles Transformer, BERT et RoBERTa, montrent une meilleure stabilité entre validation et test. Bien que leur Subset Accuracy reste modérée, ces modèles conservent des performances robustes en F1-score, en particulier en Macro F1, ce qui est crucial dans un contexte de déséquilibre de classes. RoBERTa obtient sur l'ensemble de test un F1 Micro de 0.7595 et un F1 Macro de 0.7047, légèrement supérieurs à ceux de BERT, confirmant l'impact positif de son protocole de pré-entraînement optimisé.

Comparaison selon les Métriques Multi-label Le F1-score Macro, métrique principale de cette étude, met en évidence la supériorité relative du SVM (0.763) par rapport à XGBoost (0.6698) et aux modèles Transformer, dont les scores se situent autour de 0.70. Cette observation suggère que, malgré leur expressivité limitée, les modèles linéaires régularisés restent compétitifs pour la couverture uniforme des catégories rares lorsque la représentation TF-IDF est bien calibrée.
Le F1 Micro favorise les modèles capturant efficacement les classes majoritaires. Dans ce cadre, le SVM conserve l'avantage sur l'ensemble de test, tandis que XGBoost et les Transformers présentent des performances proches mais inférieures. Le Hamming Loss confirme cette tendance : le SVM affiche la plus faible perte sur les labels (0.0671), indiquant une meilleure précision globale au niveau des décisions binaires.
La Subset Accuracy, métrique la plus stricte, révèle la difficulté intrinsèque de la tâche multi-label. Aucun modèle ne dépasse 50% sur l'ensemble de test, ce qui est attendu compte tenu du nombre élevé de labels et de leurs cooccurrences complexes. Le SVM reste néanmoins le plus performant sur cet axe, tandis que les modèles profonds privilégient une meilleure couverture partielle des labels plutôt qu'une exactitude parfaite.

TABLE 12 : Comparaison des performances sur l'ensemble de test

Modèle	**F1 Macro**	**F1 Micro**	**Hamming Loss**	**Subset Accuracy**
SVM (CC + TF-IDF + SVD)	0.7630	0.8464	0.0671	0.4704
XGBoost (TF-IDF + SVD)	0.6698	0.7748	0.0971	0.2498
BERT	≈0.70	≈0.76	≈0.10	≈0.28
RoBERTa	0.7047	0.7595	0.1030	0.2838

Synthèse Comparative des Performances

Discussion Ces résultats mettent en évidence un compromis fondamental entre expressivité du modèle et capacité de généralisation. Les modèles Transformer excellent dans la capture du contexte sémantique et des relations complexes entre mots, mais nécessitent des volumes de données importants et une régularisation fine pour surpasser les méthodes classiques. À l'inverse, le SVM, bien que conceptuellement plus simple, démontre une robustesse remarquable dans ce cadre multi-label, faisant de lui une solution compétitive lorsque les ressources computationnelles ou les données annotées sont limitées.

6.4 Comparaison en Termes de Complexité

6.5 Comparaison en Termes de Complexité

Cette sous-section compare les quatre modèles selon leur complexité computationnelle, en tenant compte du nombre de paramètres, de la taille mémoire du modèle, des ressources matérielles requises (CPU/GPU), du temps d'entraînement ainsi que des considérations pratiques liées à l'implémentation et au déploiement. Ces critères sont essentiels pour évaluer la faisabilité opérationnelle des modèles dans un contexte réel.

Analyse des Ressources et du Temps d'Entraînement Les modèles classiques basés sur des représentations TF-IDF présentent un avantage net en termes de coût computationnel. XGBoost se distingue comme le modèle le plus léger et le plus rapide à entraîner. Avec environ 838 744 nœuds, une taille de modèle de seulement 31.6 MB et un temps d'entraînement inférieur à 5 minutes, il constitue une solution particulièrement

adaptée aux environnements contraints en ressources. Sa consommation GPU reste modérée (environ 4.4 GB), ce qui facilite son exécution sur des infrastructures standards.
Le SVM, bien que conceptuellement simple, présente une complexité mémoire élevée due à la très grande dimension de l'espace TF-IDF. Le nombre effectif de paramètres atteint près de 198 millions, ce qui se traduit par une taille de modèle avoisinant 1.9 GB et une consommation mémoire CPU comprise entre 5 et 9 GB. Le temps d'entraînement, estimé à environ 3 heures, reste toutefois inférieur à celui des modèles Transformer. Malgré ces coûts, le SVM demeure relativement simple à implémenter et à optimiser pour le déploiement.
À l'opposé, les modèles profonds de type Transformer sont nettement plus exigeants. BERT, avec ses 110 millions de paramètres et une taille de modèle de 439 MB, requiert entre 14 et 15 GB de mémoire GPU pour l'entraînement, avec un temps total d'environ 6 heures. RoBERTa accentue encore cette complexité avec 125 millions de paramètres, une taille de 498 MB et un temps d'entraînement proche de 8 heures. Ces exigences élevées limitent leur accessibilité et imposent l'utilisation de matériels spécialisés.

Facilité d'Implémentation et Déploiement Du point de vue de l'ingénierie logicielle, les modèles SVM et XGBoost sont les plus simples à mettre en œuvre. Ils reposent sur des bibliothèques matures, offrent des pipelines d'entraînement stables et permettent un déploiement direct sans dépendances lourdes. Leur optimisation en production est généralement aisée, notamment grâce à des formats de modèles compacts et des temps d'inférence réduits.
À l'inverse, BERT et RoBERTa présentent une complexité opérationnelle plus élevée. Leur implémentation nécessite la maîtrise de frameworks de deep learning, la gestion fine de la mémoire GPU et des stratégies spécifiques d'optimisation (quantification, pruning, distillation) pour envisager un déploiement efficace. Ces contraintes rendent leur intégration en production plus délicate, en particulier dans des contextes industriels à ressources limitées.

TABLE 13 : Comparaison de la complexité computationnelle des modèles

Modèle	Paramètres	Taille	Mémoire GPU	Temps Train	Implémentation	Déploiement
SVM	198 M	∼1.9 GB	2–5.5 GB	∼3 h	Facile	Facile
XGBoost	0.84 M	31.6 MB	∼4.4 GB	∼5 min	Facile	Facile
BERT	110 M	439 MB	14–15 GB	6 h 17 min	Modérée	Difficile
RoBERTa	125 M	498 MB	10–15 GB	7 h 47 min	Modérée	Difficile

Synthèse Comparative de la Complexité

Discussion Cette analyse met en évidence un compromis clair entre performance sémantique et coût computationnel. Les modèles classiques, en particulier XGBoost, offrent une excellente efficacité en termes de ressources et de rapidité, ce qui les rend adaptés à des scénarios de déploiement à grande échelle. Les modèles Transformer, bien que plus performants dans la modélisation du contexte linguistique, impliquent des coûts matériels et opérationnels significatifs. Le choix du modèle dépend ainsi directement des contraintes applicatives, du volume de données disponible et des ressources de calcul accessibles.

7 deploimenet

8 Déploiement de l'Application

8.1 Architecture Générale

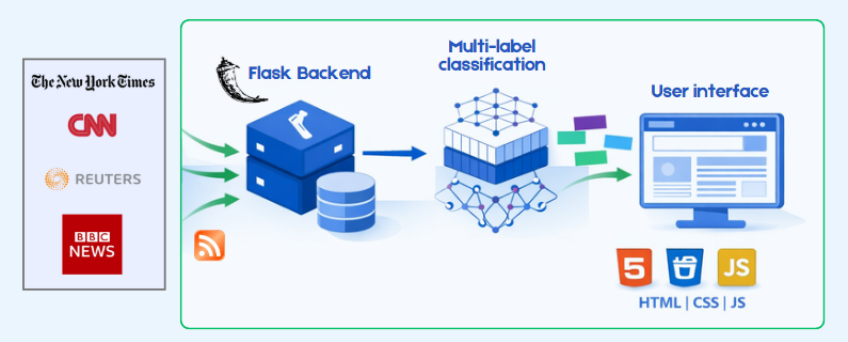


FIGURE 14 : Architecture de l'application

L'application de classification d'articles de presse a été déployée sous forme d'une application web utilisant le framework Flask pour le backend et une interface HTML/CSS/-

JavaScript pour le frontend. L'architecture suit un modèle client-serveur classique où le serveur Flask gère les requêtes HTTP, orchestre la récupération des données depuis les flux RSS, applique le modèle de classification, et renvoie les résultats au client sous forme de JSON.

Le système est composé de trois couches principales. La première couche est l'interface utilisateur qui s'exécute dans le navigateur web de l'utilisateur et communique avec le serveur via des requêtes HTTP asynchrones. La deuxième couche est le serveur Flask qui expose trois endpoints principaux : la route racine pour servir l'interface HTML, la route `/api/news` pour récupérer et classifier les articles, et la route `/api/categories` pour obtenir la liste des 14 catégories. La troisième couche comprend les composants de traitement, notamment le parser RSS qui utilise la bibliothèque feedparser, et le modèle RoBERTa fine-tuné chargé en mémoire avec la bibliothèque transformers de Hugging Face.

8.2 Composants Techniques

Le fichier `app.py` contient le serveur Flask principal qui définit les routes de l'application et gère le flux de données entre les différents composants. Lorsqu'un utilisateur accède à la route `/api/news`, le serveur invoque la fonction `parse_rss_feeds` pour récupérer les articles depuis quatre sources d'actualités internationales (CNN, BBC, New York Times et Reuters), limitant la récupération à 10 articles par source. Pour chaque article récupéré, la fonction `predict_labels` du module `utils.py` est appelée pour générer les prédictions de catégories.

Le module `utils.py` encapsule la logique métier de l'application. Il contient la fonction `load_model` qui charge le modèle RoBERTa et le tokenizer une seule fois au démarrage pour optimiser les performances. La fonction `predict_labels` prend en entrée le titre et la description d'un article, les concatène, tokenize le texte avec une longueur maximale de 512 tokens, et applique le modèle pour obtenir les logits. Une fonction sigmoïde est ensuite appliquée aux logits pour obtenir des probabilités, et un seuil de 0.5 est utilisé pour déterminer les catégories positives. Si aucune catégorie ne dépasse le seuil, la catégorie avec la probabilité la plus élevée est retournée.

L'interface frontend est développée en HTML, CSS et JavaScript vanilla sans framework externe. Le fichier `index.html` structure la page avec un header, une section de contrôles contenant le bouton de récupération et un système de filtrage par catégories, une zone

d'affichage des statistiques, et une grille pour présenter les cartes d'articles. Le fichier `style.css` définit un design moderne avec un gradient de fond violet, des cartes blanches avec des ombres portées, et des animations de survol. Le fichier `script.js` gère l'interactivité de l'application, notamment le chargement des articles via des requêtes fetch asynchrones, la mise à jour dynamique de l'interface, et le système de filtrage multi-sélection qui permet à l'utilisateur de sélectionner plusieurs catégories simultanément pour affiner les résultats affichés.

8.3 Fonctionnement du Système

Le flux de fonctionnement de l'application commence lorsque l'utilisateur clique sur le bouton "Fetch & Classify News". Cette action déclenche une requête asynchrone vers l'endpoint `/api/news` qui active le processus de récupération des flux RSS. Le serveur parse les flux XML des quatre sources configurées, extrait les informations pertinentes de chaque article (titre, description, lien, date de publication, source), et nettoie le texte des balises HTML. Pour chaque article, le texte est ensuite passé au modèle RoBERTa qui effectue la classification multi-étiquettes.

Le modèle génère des prédictions pour les 14 catégories définies, et les résultats sont agrégés dans une structure JSON contenant les articles avec leurs métadonnées et leurs catégories prédites. Cette réponse JSON est envoyée au client qui met à jour l'interface en créant dynamiquement des cartes d'articles. Chaque carte affiche le titre de l'article, sa description, sa source, sa date de publication, et des badges colorés représentant les catégories prédites. L'utilisateur peut ensuite utiliser le système de filtrage pour afficher uniquement les articles appartenant aux catégories sélectionnées, avec une mise à jour instantanée de l'affichage et des statistiques.

8.4 Performance et Optimisations

Le système présente de bonnes performances avec un temps de traitement total de 30 à 40 secondes pour récupérer et classifier environ 40 articles. Ce temps se décompose en environ 10 à 15 secondes pour la récupération des flux RSS et 20 à 25 secondes pour la classification des articles. Le chargement initial du modèle RoBERTa (498 MB) Démarre en processeur et prend environ 10 à 15 secondes au démarrage de l'application, mais cette opération n'est effectuée qu'une seule fois grâce à l'utilisation de variables globales qui

maintiennent le modèle en mémoire.

Le système fonctionne également correctement sur CPU, bien qu'avec des temps d'inférence légèrement supérieurs. L'interface frontend utilise des techniques d'optimisation JavaScript modernes, notamment la délégation d'événements et la mise à jour ciblée du DOM pour minimiser les recalculs de layout et assurer une expérience utilisateur fluide.

8.5 Déploiement et Configuration

Le déploiement de l'application nécessite Python 3.8 ou supérieur avec les bibliothèques Flask, feedparser, torch et transformers. L'application peut être lancée en mode développement avec la commande `python app.py`, qui démarre le serveur Flask sur le port 5000. Pour un déploiement en production, il est recommandé d'utiliser un serveur WSGI comme Gunicorn combiné avec un reverse proxy Nginx pour gérer le load balancing et servir les fichiers statiques efficacement.

La configuration système minimale requiert 2 GB de RAM pour charger le modèle, bien qu'il soit recommandé d'avoir au moins 4 GB pour assurer des performances optimales. Un GPU n'est pas obligatoire mais peut améliorer significativement les temps d'inférence, particulièrement lors du traitement de grands volumes d'articles. L'application est compatible avec les principales plateformes cloud (AWS, Azure, Google Cloud Platform) et peut être containerisée avec Docker pour faciliter le déploiement et la scalabilité horizontale.

9 Conclusion et Perspectives

9.1 Conclusion

Ce projet a exploré la classification multi-label d'articles de presse à travers l'implémentation et la comparaison de quatre approches distinctes, allant des méthodes classiques d'apprentissage automatique aux architectures Transformer modernes. L'objectif principal était d'évaluer la capacité de ces modèles à organiser et étiqueter automatiquement le contenu journalistique dans un contexte de déséquilibre des classes et de corrélations complexes entre catégories.

Les résultats obtenus confirment que le choix du modèle dépend fortement du compromis entre performance prédictive et complexité computationnelle. Le SVM avec Classifier Chain et représentation TF-IDF a démontré une robustesse remarquable, atteignant un F1 Macro de 0.763 sur l'ensemble de test, avec une excellente capacité de généralisation et un temps d'entraînement raisonnable. XGBoost, malgré ses performances exceptionnelles en entraînement, a révélé un surapprentissage significatif sur les classes rares, soulignant les limites des méthodes d'ensemble sur des données textuelles déséquilibrées.

Les modèles Transformer, BERT et RoBERTa, ont montré une meilleure capacité à capturer le contexte sémantique et les dépendances linguistiques complexes, avec des performances stables entre validation et test. RoBERTa a légèrement surpassé BERT avec un F1 Macro de 0.7047, confirmant l'impact positif de son protocole de pré-entraînement optimisé. Cependant, ces modèles imposent des coûts computationnels élevés, nécessitant entre 6 et 8 heures d'entraînement sur GPU et une empreinte mémoire significative, ce qui peut limiter leur accessibilité dans des environnements à ressources contraintes.

L'analyse par catégorie a mis en évidence un biais persistant envers les classes majoritaires pour tous les modèles. Les catégories fréquentes comme Agriculture Food et Trade Business ont systématiquement obtenu des scores F1 supérieurs à 0.85, tandis que les catégories rares comme Law Justice et Employment Labor n'ont pas dépassé 0.50, malgré l'utilisation de techniques de rééquilibrage. Ce phénomène souligne la nécessité de stratégies plus sophistiquées pour traiter le déséquilibre des classes en classification multi-label.

Le déploiement de l'application web basée sur Flask a démontré la faisabilité d'une solu-

tion opérationnelle de classification automatique en temps réel. L'intégration du modèle RoBERTa permet de classifier environ 40 articles en 30 à 40 secondes, offrant une interface utilisateur intuitive avec filtrage multi-catégories et visualisation dynamique des résultats. Cette implémentation constitue une preuve de concept viable pour l'automatisation de l'organisation de contenu journalistique dans les médias.

9.2 Perspectives

Plusieurs axes d'amélioration peuvent être explorés pour renforcer les performances et l'applicabilité des modèles développés. L'adoption de stratégies avancées de gestion du déséquilibre des classes, telles que l'augmentation de données par rétro-traduction ou paraphrase automatique, pourrait enrichir la représentation des catégories minoritaires sans simple duplication. L'exploration d'architectures spécialisées en classification multi-label, comme les réseaux de corrélation de labels ou les modèles graphiques, permettrait de mieux capturer les dépendances entre catégories.

L'utilisation de modèles Transformer de plus grande taille, tels que RoBERTa-Large ou DeBERTa, ou le recours à des modèles pré-entraînés sur des corpus juridiques et journalistiques spécialisés, pourrait améliorer significativement la performance sur les catégories rares. L'implémentation de mécanismes d'attention hiérarchique pour traiter les documents longs dépassant 512 tokens constitue également une piste prometteuse pour exploiter l'intégralité du contenu textuel.

Du point de vue opérationnel, l'optimisation du modèle par quantification, distillation de connaissances ou pruning permettrait de réduire les coûts d'inférence et de faciliter le déploiement sur des infrastructures à capacité limitée. L'extension de l'application à une classification multilingue exploitant la nature multilingue du dataset MultiEURLEX ouvrirait des perspectives pour le traitement de contenus internationaux. Enfin, l'intégration de mécanismes d'apprentissage actif pour identifier et annoter les exemples les plus informatifs permettrait d'améliorer continuellement les performances du système en production.