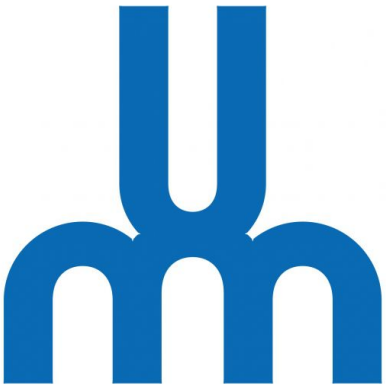
Projet 1

Samuel Fournier
20218212

Thierry Bédard-Cortey
20307759

Dans le cadre du cours
IFT 6150



Département d'informatique et de recherche opérationnelle
Université de Montréal
Canada
29 octobre 2025

Introduction

L'objectif de ce projet est en premier temps effectuer une cartographie de l'antholgie de l'ACL et en second temps d'implémenter deux tâches de classification se trouvant dans notre cartographie dans le but de comparer diverses méthodes à celles implémentées dans les articles choisis.

Contexte et problématique

L'ACL est hôte de plusieurs événements dans le domaine du TALN. Plusieurs papiers qui traitent du traitement de langue y sont publiés chaque année. Dans ce projet, on s'intéresse particulièrement aux papiers qui discutent de classification. Notre objectif est d'observer l'évolution du nombres de papiers de classification au fil des années, l'évolution du type de classification traité dans ces papier, l'évolution des méthodes utilisées ainsi que d'autre variables que l'on a jugé pertinente. Heureusement pour nous, l'ACL possède une librairie Python qui nous permet d'extraire des métadonnées des différents articles présent sur leur site web comme le titre d'un article, l'abstract de celui-ci, l'année de publication ainsi que d'autres données pertinentes. Dans le cadre de ce projet, on se limite aux événements de l'ACL suivant: ACL, NAACL, EACL, CoNLL, EMNLP, COLING, LREC ainsi que les Findings.
Lors de la recherche d'article qui traitent de classification, on s'est intéressé aux trois types suivant:

- Classification Binaire
- Classification Multi-Classe
- Classification Multi-Label

On s'est aussi intéressé à différents domaines dans lequel ces méthodes de classifications sont utilisées. Voici une liste non-exhaustive des domaines d'intérêt:

- Sentiment (opinion, emotion, etc.)
- Toxicité (discours de haine, contenu offensif, etc.)

Finalement, on s'est aussi intéressé aux différentes méthodes utilisées pour performer la classification. Voici une liste de quelques méthodes d'intérêt:

- Transformeur (BERT, RoBERTa, gpt, etc)
- RNN (GRU, HRNN, RNN-CRF, etc.)
- CNN (TCN, TextCNN, etc.)

Limitations de notre approche

Lorsque l'on cherche à déterminer si un article traite de classification, nous analysons le titre et le sommaire à la recherche des mots-clés définis. Cependant, certains articles ne possèdent pas de sommaire (la librairie Python retourne alors **None**) ou mentionnent un mot-clé sans réellement aborder le sujet. Par exemple, un article évoquant brièvement la classification binaire dans son titre ou son résumé sera tout de même identifié comme tel. Lors du processus de cartographie, nous avons constaté que la quantité d'information disponible via la librairie de l'ACL est très limitée

: seuls le titre et le sommaire contiennent du texte exploitable pour une recherche par mots-clés. Ainsi, certains articles de classification peuvent ne pas être détectés si les mots-clés sont absents de ces champs. Malgré cela, plusieurs tests nous permettent d'affirmer que la méthode demeure suffisamment robuste pour isoler la majorité des articles de classification publiés entre deux années données. Enfin, l'ACL ne fournit pas le nombre de citations des articles. La seule solution identifiée consisterait à recourir au *web scraping*, une pratique non autorisée et chronophage (nous avons d'ailleurs temporairement été bloqués par Google Scholar pour excès de requêtes).

Méthodologie de cartographie

Collecte et filtrage des articles

Le processus de cartographie est un processus simple, mais long. En premier temps, on procède par une itération de tous les volumes de l'ACL avec une boucle **for**. Ensuite, on vérifie si le volume a été publié lors d'un des événements qui nous intéressent. Finalement, si le volume est publié entre les années **self.start_year** et **self.end_year** et qu'il a été publié lors d'un des événements qui nous intéressent, on itère sur tous les articles de ce volume et on procède à l'extraction des données de cet article pour déterminer s'il est un article de classification ou non.

Extraction d'attributs

Comme nous l'avons mentionné dans la dernière section, l'extraction des attributs d'un article est un processus très limité. En effet, les seules données auxquelles nous avons accès sont le numéro d'identification de l'article, son titre, son sommaire, son année de publication, la liste des auteurs(es), l'événement dans lequel il a été publié ainsi que son DOI (*Digital Object Identifier*). Une fois que l'on a extrait les données pertinentes, nous utilisons le titre ainsi que le sommaire du papier pour déterminer si l'un d'entre eux contient un des mots-clefs associés à la classification. Une fois que nous avons déterminé si c'est un papier qui traite de classification, on tente d'extraire toutes les informations que nous avons jugées pertinentes comme le domaine, le type de méthode, les *benchmarks*, etc. en répétant le même processus d'isolation de mots-clefs.

Processus d'extraction des *benchmarks*

Afin d'être en mesure de détecter les *benchmarks* utilisés dans chaque article, nous avons utilisé le jeu de données *pwc-archive/datasets* disponible sur *Hugging Face*. Celui-ci contient tout les *datasets* anciennement répertoriés sur la plateforme *Papers with Code* et maintenant dans la section *Trending Papers* de *Hugging Face*.
Après avoir chargé le jeu de données, la méthode *_load_benchmark_data()* commence par filtrer les entrées afin de ne conserver que celles associées à des tâches de classification de texte. Ce filtrage s'effectue en parcourant la colonne *tasks* de chaque entrée, où l'on recherche des termes comme "text classification", "sentiment analysis", "topic classification", etc.
Une fois ce sous-ensemble obtenu, chaque benchmark est enrichi avec la liste de ses alias afin de faciliter la détection de mentions dans les articles, peu importe la variante utilisé (par exemple : "GLUE", "General Language Understanding Evaluation").
Ensuite, lors de l'analyse de chaque article, la méthode *extract_benchmarks* construit un texte combinant le titre et le sommaire qui est normalisé (suppression des accents, ponctuaction, etc.)

avant d'être comparé à la liste de noms et alias de chaque *benchmark*. Finalement, le script enregistre pour chaque article la liste des *benchmarks* détectés ainsi que leur nombre total, ce qui permet ensuite de produire des statistiques agrégées sur l'utilisation des jeux de données dans la recherche en classification.

Remarques

Si on pouvait avoir accès au contenu du texte de l'article on croit que la performance de nos méthodes serait meilleure que ce que l'on a présentement. De plus, si nous avions accès aux textes, il pourrait être intéressant d'entraîner un modèle pour faire cette tâche de cartographie plus facilement et de façon plus précise, même si ce serait un peu exagéré pour ce projet.

Analyse descriptive

Notre analyse principale c'est limité entre 2010 et 2025. On observe une tendance croissante du nombre d'articles de classifications 1 en 15 ans. Cependant, il semble y avoir le début d'une tendance décroissante à partir des années 2022-2023. On voit aussi que la conférence avec le plus d'articles de classifications est celle de l'EMNLP suivi des `Findings` et ensuite l'ACL 2. Si on ignore la colonne `Unspecified` pour un instant, nous voyons que la classification *multi-label* est le type de classification le plus présent dans les articles suivi de la classification multi-classe et en dernier (et de loin) la classification binaire 3. On peut attribuer la domination de la catégorie `Unspecified` aux limites de nos méthodes d'analyse. Ensuite, une vaste majorité des articles traitent de NER avec plusieurs articlent traintant d'un sujet `Other` que l'on a pas réussi à extraire (soit qu'il n'est pas mentionné ou on le l'a pas considéré) 4. Finalement, si on ignore la catégorie `Unspecified` pour le mêmes raisons que plus haut, la vaste majorité des articles publiés dans ces années utilisent des méthodes de transformeurs comme `BERT` ou `GPT` 5. Finalement, si nous observons le nombre d'articles publiés entre 1965 et 2025, on voit une croissance exponentielle du nombre d'article d'année en années 7.

Sélection de tâches sous-exploitées

Article 1 (implémenté par Samuel Fournier)

Présentation de l'article

Après avoir cartographié l'anthologie de l'ACL, nous avons sélectionné deux articles parmis ceux cartographiés. Le premier article s'intitule *Exploring the Limits of Simple Learners in Knowledge Distillation for Document Classification with DocBERT*[1]. Dans cet article, l'équipe d'Adhikari se donnent comme but de comparer un modèle BERT *fine-tuner* pour faire de la classification de document. Cet article est cité 42 fois (selon Google scholar). Dans l'article, ils utilisent quatres datasets différents: `Reuters-21578`, `arXiv Academic Paper Dataset (AAPD)`, `IMDB Reviews` et `Yelp 2014 Reviews`. Leur objectif dans l'article est de développer un modèle BERT qui est en mesure d'obtenir des performances de l'état de l'art (en 2020). Cependant, un enjeu qu'ils décident de traiter est la taille du modèle ainsi que son cout d'entrainement. En effet, dans le sommaire du papier, ils mentionnent le fait qu'un modèle BERT *fine-tuned* est très couteux en terme de coût de calculs. Ils réussissent la compression du modèle grâce à une approche de *knowledge distillation*. Cette approche consiste à entraîner un modèle `étudiant` qui se base sur les poids d'une modèle

`enseignant` pour apprendre. Dans leur papier, ils utilisent un modèle BERT *fine-tuner* comme l'enseignant et tente d'expérimenter avec plusieurs architectures comme modèle étudiant.

Résultats de l'article

Dans l'article, ils ont utilisés plusieurs architectures différentes pour comparer leur résultat. Les deux méthodes les plus simples sont la régression logistique et un SVM entrainés avec TF-IDF. Leur résultats sont plus faibles que les méthodes BERT implémentées, mais que d'environ 10% dans le cas de la régression et d'environ 5% dans le cas du SVM. Ils ont aussi testé le modèle KimCNN[3], mais ce modèle obtient les pires performances avec un score F1 de 51.4 ± 1.3. Finalement, l'autre modèle d'intérêt est un modèle LSTM[2]. Ce modèle performe très bien et obtient des résultats comparables aux modèles BERT implémentés. Finalement, ils ont utilisés le concept de *knowledge distillation* et l'ont appliqué aux modèles KimCNN, LSTM et la régression logistique. Les scores F1 de ces nouveaux modèles sont meilleures que ceux obtenus par le modèle de base. En effet, le modèle KD-LSTM a eu une amélioration de 3 points sur le jeu de données Test de l'AAPD. Le modèle KD-KimCNN est passé de 51.4 ± 1.3 à 70.6 ± 0.1. Et finalement, la régression logistique a augmenté d'environ 5% sur ce même jeu de données.
Bref, ils ont montré que l'entraînement de modèle plus petit par *knowledge distillation* leur permettait d'obtenir des réultats comparable à l'état de l'art. Cependant, cette méthode ne résout malheureusement pas le coût élevé en ressources de calculs à l'entraînement.

Article 2 (implémenté par Thierry Bédard-Cortey)

Présentation de l'article

L'article *LLMEmbed: Rethinking Lightweight LLM's Genuine Function in Text Classification* [4] propose une approche innovante pour la classification de texte à partir de représentations extraites de modèles de langage légers. Les auteurs remettent en question le paradigme dominant du *prompt learning*, souvent coûteux en ressources et dépendant de modèles de très grande taille.
L'objectif principal de l'article est de démontrer qu'un LLM léger (par exemple LLaMA2-7B) peut rivaliser avec des modèles massifs (comme GPT-3) lorsqu'il est utilisé non pas pour générer des réponses textuelles, mais pour produire des *embeddings* adaptés à une tâche de classification. L'approche, nommée **LLMEmbed**, consiste à extraire et fusionner les représentations internes du modèle (à différentes couches), avant d'entraîner un classificateur séparé. Ainsi, la méthode supprime la dépendance à la génération de texte via *prompts* et permet un apprentissage plus rapide, plus stable et considérablement moins énergivore.
Les principales contributions de l'article sont l'introduction d'un nouveau paradigme de transfert basé sur l'extraction d'embeddings issus de LLMs légers pour la classification de texte, la démonstration empirique que cette approche dépasse les méthodes de *prompt learning* avec des modèles de taille modérée, ainsi qu'une analyse détaillée du coût computationnel, énergétique et financier confirmant les gains d'efficacité de LLMEmbed.
Les expériences ont été menées sur cinq jeux de données classiques en classification de texte : SST-2, MR, AGNews, R8 et R52. Les auteurs comparent systématiquement leur approche à plusieurs variantes de *prompt-based learning* et à des méthodes supervisées traditionnelles.

Résultats de l'article

Les résultats expérimentaux démontrent que LLMEmbed obtient des performances de classification élevées tout en réduisant drastiquement le coût de calcul.
Performance de classification. Sur les cinq benchmarks (SST-2, MR, AGNews, R8, R52), la meilleure configuration de LLMEmbed (*Cat + Co + Avg + Cat*) atteint une précision moyenne de **96.20%**. Les résultats détaillés sont : SST-2 = 95.76%, MR = 95.49%, AGNews = 95.83%, R8 = 98.22%, R52 = 95.68%. Ces scores dépassent la moyenne de la méthode *prompt-based* SOTA (CARP avec GPT-3, 95.16%), malgré l'utilisation de modèles nettement plus petits.
Efficacité computationnelle. Le temps d'exécution pour SST-2 est réduit à seulement 22 minutes contre 79 h pour une approche *prompt-based*. La consommation énergétique moyenne sur les *benchmarks* testés passe de 20.93 kWh pour la méthode *CARP* à 0.38 kWh pour la méthode *LLMEmbed*. Par conséquent, cela réduit également grandement le cout monétaire. LLMEmbed élimine également la dépendance aux opérations de génération séquentielle, ce qui permet un traitement par lots (*batch processing*) efficace.
Analyse qualitative. Les auteurs observent que la fusion d'embeddings issus de différentes couches et modèles améliore la robustesse du classificateur. Les approches utilisant des stratégies de *pooling* moyenne (average pooling) surpassent celles basées sur le maximum. De plus, l'approche évite les problèmes d'hallucination ou de surcharge contextuelle souvent rencontrés avec les prompts sur des modèles légers.
Limites. Les auteurs mentionnent que la méthode reste moins performante pour les textes longs et que l'explicabilité du classificateur issu d'embeddings pourrait être améliorée.
En somme, l'article montre que les LLM légers peuvent être réutilisés de manière efficace pour la classification de texte via la génération d'embeddings plutôt que par génération de texte, réduisant ainsi considérablement les coûts de calcul et d'énergie tout en maintenant une performance compétitive.

Approches expérimentales

Article 1

Comme l'article a comme but de comparer des méthodes plus simples et plus petites à de gros modèles, on a voulu implémenter l'algorithme de classification le plus simple: le kNN (*k-nearest neigbour*). Nous avons implémenter deux versions du kNN. Comme la tâche de classification que l'article a fait sur le jeu de données AAPD est une tâche de classification *multi-label*, il faut modifier l'approche du kNN. On a implémenter deux approches différentes pour permettre au kNN de faire de la classification *multi-label*. Dans les deux méthodes, on utilisent l'algorithme TF-IDF de `Scikit-learn` pour encoder les textes dans une formes manipulables pour les deux approches.

ML-kNN

La première approche utilise un modèle *Multi-Label kNN* (ML-kNN). Cette version du kNN est spécifiquement construite pour faire de la classification *multi-label*. On a utilisé la librairie `Scikit-learn` de python pour l'implémentation de cette approche.
Cette version du kNN fonctionne essentiellement de la même façon qu'un kNN régulier, mais ce modèle utilise une approche probabiliste pour déterminer le ou les *label(s)* associer à un nouveau point. En premier temps, on détermine quels sont les k points les plus proches. Ensuite, on applique

le principe du maximum à posteriori (MAP). Cette étape consiste à évaluer la probabilité que la nouvelle instance possède l'étiquette l_i grâce au théorème de Bayes. La formule qui évalue cette probabilité est la suivante:

$$\mathbb{P}(l_j = 1|i) = \frac{C_{i,j} + s}{R_i + s \cdot 2} \tag{1}$$

Où pour l'étiquette l_i et le compte i: $C_{i,j}$ est le nombre d'instances d'entraînement qui possèdent l'étiquette l_j et qui ont exactement i voisins qui possèdent aussi l'étiquette l_j, R_i est le nombre total d'instances d'entraînement qui ont exactement i voisins avec l'étiquette l_j et s est un hyperparamètre de lissage. Finalement, la décision est prise en fonction d'un seuil fixe (habituellement 0.5).

BR-kNN

La seconde approche implique ce qu'on appelle un *Binary Relevance kNN* (BR-kNN). Pour cette approche, on doit entraîner 54 (le nombres d'étiquette dans le jeu de données AAPD) kNN qui font de la classifications binaires. Le processus est simple. Chaque kNN binaire est entraîné pour une étiquette spécifique et lorsqu'on ajoute un nouveau point, les 54 kNN trouvent les k voisins les plus proches de ce nouveau point et font chacun un vote de majorité pour déterminer si le nouveau point possède l'étiquette (1) ou non (0). Ensuite, les prédictions des 54 kNN sont compiler dans une liste qui représente l'ensemble des étiquettes prédites par le modèle.

Résultats

Les résultats des deux versions du kNN sont intéressantes. Malgré le fait qu'elles obtiennent des scores F1 plus faibles que ceux obtenus par les différentes méthodes de l'article, elles sont tout de même assez proche de la régression logistique en terme de performance. On voit bien dans cette figure 8 que le score F1 sur l'ensemble des prédictions de ML-kNN est de 63.45 et le score F1 de BR-kNN 9 est de 59.16. Nous avons aussi implémenter la régression logistique avec `Scikit-learn` pour comparer notre score à celui de l'article et nous avons obtenus 67.79. Il est difficile d'expliquer pourquoi notre score est meilleur que celui obtenu par l'article. On assume que c'est un détail d'implémentation entre la librairie `Scikit-learn` et celle de l'article. Il serait définitivement intéressant de voir qu'est-ce que le *knowledge distillation* pourrait faire à nos modèles kNN. Comme l'objectif de cette méthode est d'utiliser un modèle plus grand (comme BERT) et l'utiliser comme *enseignant* pour entraîner un modèle plus léger (comme notre kNN), nous croyons que les performances de nos kNN augmenteraient considérablement (comme c'est le cas pour le KD-RL de l'article) et aurait l'avantage d'être un petit modèle très simple a implémenter et à entraîner par la suite.

Article 2

Deux approches complémentaires ont été experimentées afin d'évaluer la performance et l'interprétabilité des représentations textuelles utilisées pour la classification. En raison du temps computationnel très élevé pour la génération des représentations, nous avons concentré notre expérience sur le jeu de donnée SST-2.

Régression logistique avec TF-IDF

La première approche repose sur un *pipeline* classique combinant une vectorisation TF-IDF et une régression logistique linéaire. Chaque phrase est transformée en un vecteur creux représentant la fréquence pondérée des termes (*Term Frequency-Inverse Document Frequency*). Ce modèle, implémenté à l'aide de `scikit-learn`, permet une interprétation directe de l'importance des mots à travers les poids appris par le classificateur. Afin d'assurer une évaluation robuste, une recherche d'hyperparamètres (*Grid Search*) a été effectuée sur plusieurs valeurs du coefficient de régularisation C et des plages de n-grammes, combinée à une validation croisée stratifiée. Cette approche constitue une ligne de base fiable, car elle est rapide à entraîner, peu coûteuse en ressources, et fournit une référence solide pour mesurer les gains apportés par des représentations plus riches.

Classificateur linéaire sur les représentations.

La seconde approche introduit un classificateur linéaire léger appliqué directement sur les représentations produites par les trois modèles de langage utilisés dans l'article (LLama-2-7b, RoBERTa et Bert). Le modèle `LinearClassifier` consiste en trois couches linéaires indépendantes projetant les trois représentations dans un espace commun, suivies d'une normalisation et d'une classification linéaire finale. Cette architecture simple mais flexible agit comme un classificateur supervisé entraîné sur des représentations préexistantes, permettant d'analyser la contribution de chaque modèle dans la décision finale. Ainsi, elle offre un compromis idéal entre performance et interprétabilité : contrairement à des modèles profonds, les poids du classificateur linéaire peuvent être directement interprétés pour quantifier l'importance relative de chaque source de représentation, ce qui est particulièrement utile dans un cadre d'analyse comparative des LLMs.

Résultats

Analyse des résultats

La Table 1 présente les performances obtenues sur le jeu de test SST-2 pour différentes configurations de classificateurs. Le modèle *LLMEmbed* obtient les meilleurs scores globaux, surpassant la régression logistique basée sur des caractéristiques TF-IDF ainsi que le classificateur linéaire entraîné directement sur les représentations issues des modèles de langage. Ce dernier atteint une exactitude de 0.8853 et un F1 de 0.8906, servant ici de référence pour l'analyse des contributions des différents modèles d'encodage.

Ces résultats mettent en évidence la robustesse du modèle *LLMEmbed* sur la tâche SST-2. En se basant sur le seul hyperparamètre $\sigma \in [0.1, 0.5]$ décrit dans l'article, les performances demeurent stables dans cette plage, avec un optimum autour de $\sigma = 0.1$ (Acc $= 0.9278$). La variation modérée de la précision en fonction de σ suggère que le modèle n'est pas fortement sensible à ce paramètre, ce qui témoigne d'une bonne généralisation. À titre de comparaison, la régression logistique classique atteint 0.9097 d'exactitude, tandis que le classificateur linéaire sur représentations brutes reste légèrement en dessous (0.8853). Ces résultats confirment l'intérêt de la combinaison multi-modèle et du lissage contrôlé par σ proposée par *LLMEmbed*.

Analyse de l'importance de chaque modèle

Pour mieux comprendre comment les représentations de chaque modèle contribuent à la décision finale, nous avons entraîné un classificateur linéaire recevant trois blocs de représentations correspondant à `LLaMA-2`, `BERT` et `RoBERTa`. L'analyse des poids, des contributions et des permutations permet d'identifier l'importance de chaque représentation.

L'analyse des normes L2 des poids du classificateur (voir Figure 10) montre que les représentations issues de `LLaMA-2` présentent l'importance globale la plus élevée (1.73), suivies de celles de `BERT` (1.07) et de `RoBERTa` (0.70). Cette hiérarchie suggère que le modèle s'appuie principalement sur les embeddings de `LLaMA-2` pour discriminer les classes, tandis que `BERT` joue un rôle secondaire et `RoBERTa` un rôle marginal.
Les contributions moyennes par classe (voir Figure 11) révèlent une dynamique plus fine :

- Pour la classe **0** (phrases négatives), les représentations de `RoBERTa` et de `BERT` contribuent positivement à la décision, tandis que celles de `LLaMA-2` ont une influence légèrement négative.
- Pour la classe **1** (phrases positives), la situation s'inverse : `LLaMA-2` domine largement la décision avec une contribution moyenne supérieure à 1.3, `BERT` joue un rôle de soutien mineur et `RoBERTa` contribue négativement.

Ces observations sont confirmées par le test de permutation, qui mesure la baisse de performance lorsqu'un bloc d'embeddings est aléatoirement mélangé. La permutation du bloc `LLaMA-2` entraîne une chute drastique de l'exactitude (de 0.8853 à 0.5585, soit une perte de +0.3268), tandis que celle du bloc `BERT` n'engendre qu'une faible dégradation (+0.0390). Enfin, la permutation du bloc `RoBERTa` a un effet quasi nul (+0.0034), confirmant son importance marginale dans la décision finale.
En somme, le modèle linéaire apprend à s'appuyer presque exclusivement sur les représentations de `LLaMA-2`, celles de `BERT` apportant une information complémentaire limitée et celles de `RoBERTa` contribuant peu, voire négativement, selon la classe prédite.

Conclusion et travaux futurs

Dans ce projet, nous avons effectué une recherche approfondie des articles dans l'anthologie de l'ACL afin d'en extraire ceux traitant de classification. Comme mentionné précédemment, nous avons rencontré plusieurs difficultés liés à l'extraction des meta-données. Comme on faisait une recherche de mots-clefs dans le texte, nous étions limités au titre et au sommaire (si présent) d'un article. Malgré ces limites, nous avons pu identifier un grand nombre d'articles pertinents ainsi que plusieurs *benchmarks* associés.
Nous avons également reproduit et comparé les méthodes de classification de deux articles récents de l'ACL. Le premier montre que des approches simples comme le kNN ou la régression logistique peuvent encore rivaliser avec les méthodes à l'état de l'art (en 2020). Il serait intéressant de répliquer cette étude à l'aide de modèles modernes (par ex. LLM) afin d'observer l'évolution des performances. Le concept de modèles *enseignant–étudiant* nous a particulièrement interpellés : dans un contexte où la taille des modèles croît sans cesse, cette approche pourrait permettre de rendre des modèles plus petits, performants et accessibles au public.
Le second article confirme cette tendance : notre régression logistique obtient des performances proches de celles du modèle `LLMEmbed`, tout en soulignant les gains énergétiques importants offerts par cette méthode. Toutefois, celle-ci montre des limites lorsqu'il s'agit de classer de longs textes.
Les deux études explorent des pistes d'optimisation pour réduire le coût de la classification, devenu considérable avec les modèles modernes. Un prolongement naturel serait d'évaluer la compatibilité entre les deux approches étudiées. Une telle combinaison pourrait conduire à des modèles à la fois performants et économes en ressources, contribuant ainsi à atténuer l'un des grands défis actuels de l'intelligence artificielle.

References

[1] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, William L. Hamilton, and Jimmy Lin. Exploring the limits of simple learners in knowledge distillation for document classification with DocBERT. In Spandana Gella, Johannes Welbl, Marek Rei, Fabio Petroni, Patrick Lewis, Emma Strubell, Minjoon Seo, and Hannaneh Hajishirzi, editors, *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 72–77, Online, July 2020. Association for Computational Linguistics.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[3] Yoon Kim. Convolutional neural networks for sentence classification. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[4] Chun Liu, Hongguang Zhang, Kainan Zhao, Xinghai Ju, and Lin Yang. Llmembed: Rethinking lightweight llm's genuine function in text classification. pages 7891–7907, 2024.

Annexes

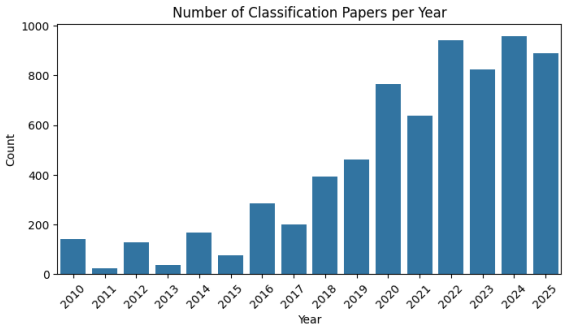


Figure 1: Nombre d'article de classifications entre 2010 et 2025

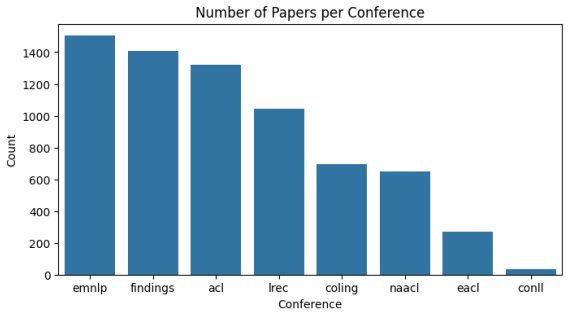


Figure 2: Nombre d'articles par conférences entre 2010 et 2025

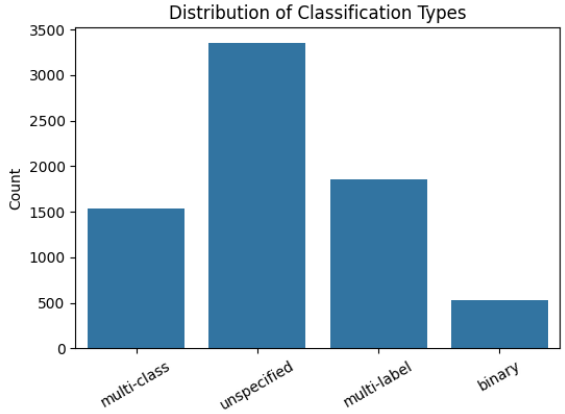


Figure 3: Nombre d'articles par type de classification entre 2010 et 2025

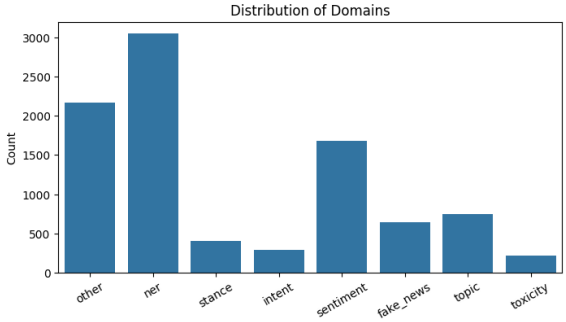


Figure 4: Nombre d'articles par domaine entre 2010 et 2025

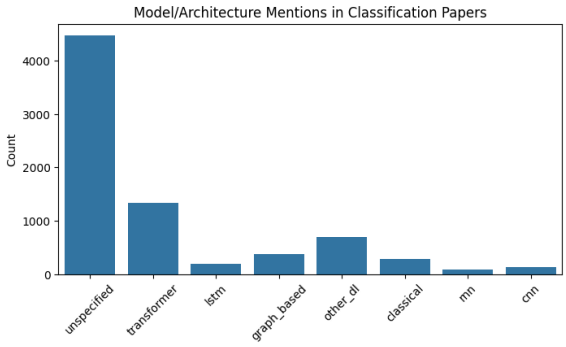


Figure 5: Nombre d'articles par méthodes entre 2010 et 2025

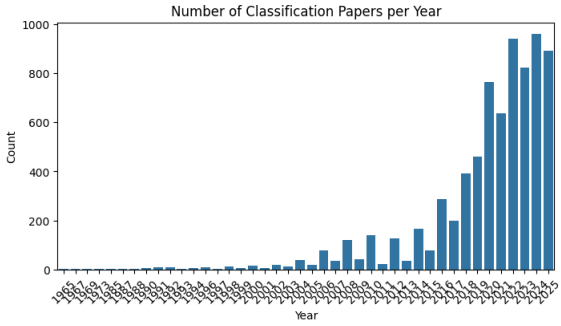


Figure 6: Nombre d'articles de classification entre 1965 et 2025

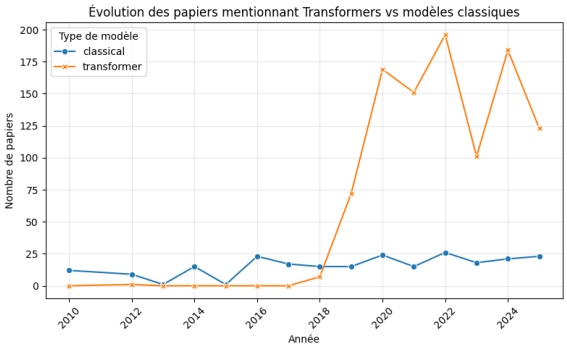


Figure 7: Évolution des papiers mentionnant Transformers vs modèles classiques

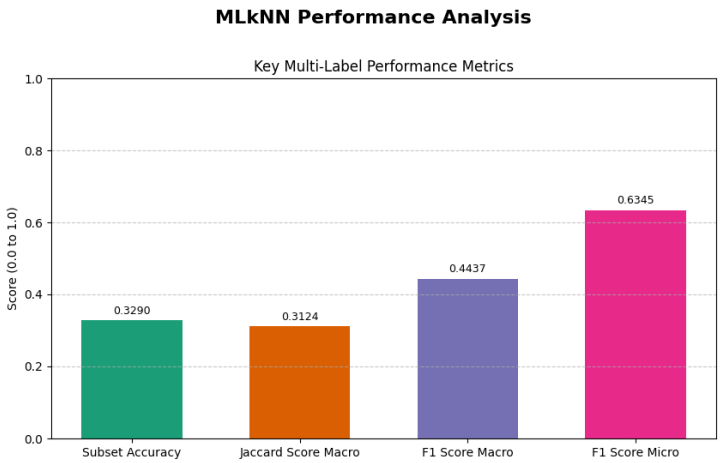


Figure 8: Performances du ML-kNN sur AAPD[Test]

Table 1: Performance des différents modèles sur SST-2.

Modèle	Test acc.	Test F1
Régression logistique ($C = 4.0$, TF-IDF unigrammes & bigrammes)	0.9097	0.9089
Classificateur linéaire sur représentations	0.8853	0.8906
LLMEmbed ($\sigma = 0.1$)	0.9278	0.9302
LLMEmbed ($\sigma = 0.2$)	0.9128	0.9195
LLMEmbed ($\sigma = 0.25$)	0.9243	0.9278
LLMEmbed ($\sigma = 0.3$)	0.9197	0.9239
LLMEmbed ($\sigma = 0.4$)	0.9048	0.9122
LLMEmbed ($\sigma = 0.5$)	0.9220	0.9269

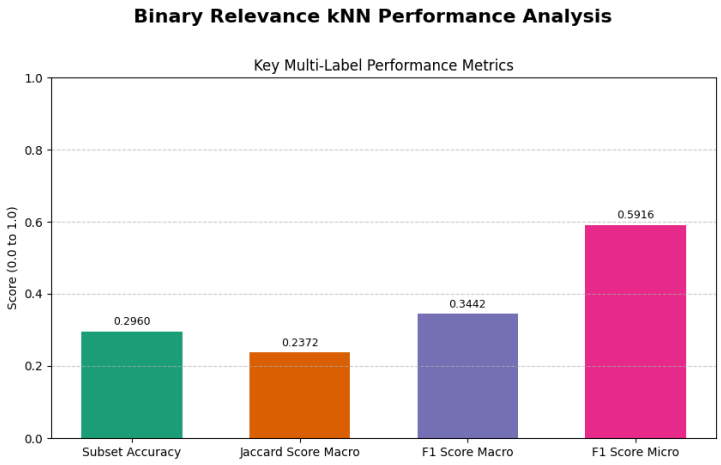


Figure 9: Performances du BR-kNN sur AAPD[Test]

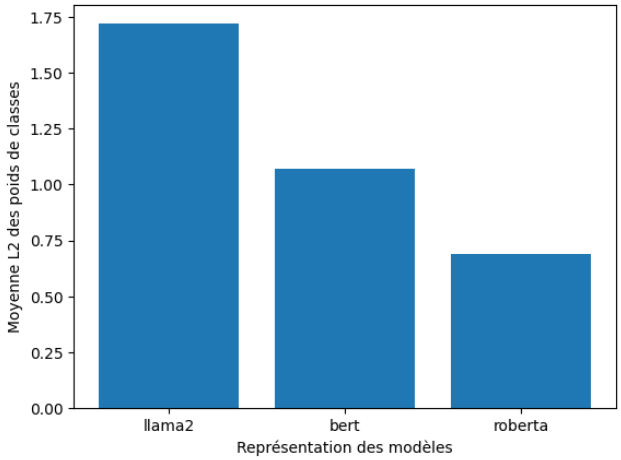


Figure 10: Importance globale de chaque modèle selon la norme L2 des poids du classificateur.

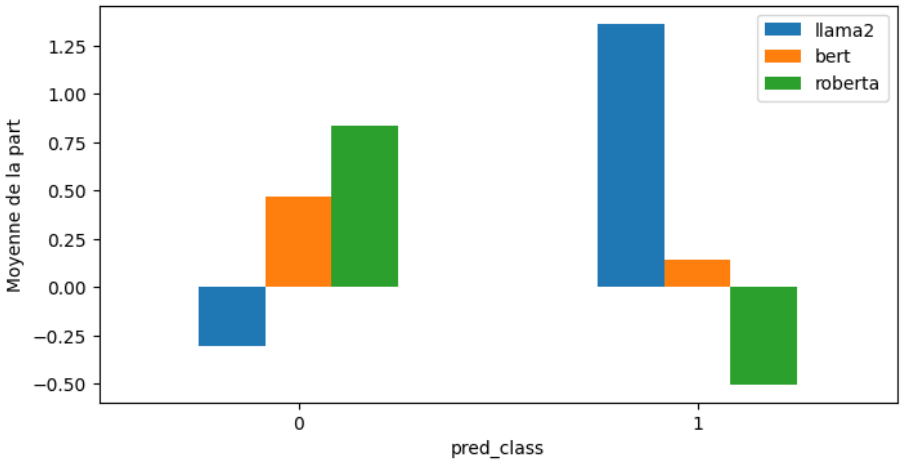


Figure 11: Moyenne de la part de contribution de chaque modèle par classe prédite.