Classification de textes — Analyse de sentiment (MI201 Projet 3)

1er Carlos Adrian Meneses Gamboa
Programme Ingénieur en STIC
ENSTA Paris
Paris, France
carlos.meneses@ensta-paris.fr

2e Jose Daniel Chacon Gomez
Programme Ingénieur en STIC
ENSTA Paris
Paris, France
jose-daniel.chacon@ensta-paris.fr

3e Santiago Florido Gomez
Programme Ingénieur en STIC
ENSTA Paris
Paris, France
santiago.florido@ensta-paris.fr

***Résumé*—Ce travail présente un système d'analyse de sentiment pour de courts textes en anglais et compare des méthodes classiques d'apprentissage automatique à une approche basée sur un transformeur. En utilisant des représentations standard (sac de mots et TF–IDF), plusieurs classifieurs sont entraînés et évalués, puis comparés à un modèle exploitant des embeddings BERT. Les résultats sont rapportés via l'accuracy et le Macro-F1, en mettant en évidence des différences de performance, de robustesse et de coût computationnel. L'étude fournit des repères pratiques pour choisir une chaîne de classification de sentiment adaptée à des contraintes de ressources typiques.**

***Index Terms*—analyse de sentiment, TAL, classification de textes, TF–IDF, BERT**

I. Introduction

L'analyse de sentiment de courts textes devient fondamentale lorsque les perceptions sont considérées comme un actif informationnel critique pour les responsables de produits et de services [1]. Cela est particulièrement pertinent dans le développement de systèmes centrés sur les émotions, capables de fournir des informations exploitables pour améliorer l'expérience utilisateur ou client. Par exemple, ces informations peuvent conduire à des ajustements des stratégies de support client ou à des campagnes marketing plus ciblées [2]. Dans ce contexte, les réseaux sociaux — et plus spécifiquement les messages courts tels que les tweets et les commentaires sur des plateformes multimédias — figurent parmi les sources les plus utilisées pour conduire ce type d'analyse.

Ce projet porte sur l'analyse automatique de sentiment de courts textes en anglais. Une phase exploratoire est d'abord menée : prétraitement du contenu, puis analyse préliminaire au moyen de méthodes classiques d'apprentissage automatique. Ensuite, la classification est réalisée avec des classifieurs standards (Naive Bayes, régression logistique et SVM linéaire), en utilisant plusieurs schémas de représentation (sac de mots, TF–IDF au niveau mot, et TF–IDF au niveau caractère). Les performances sont rapportées via l'accuracy, le Macro-F1 et des métriques complémentaires afin d'assurer une comparaison équitable.

Dans un second temps, un perceptron multicouche (MLP) entraîné sur des représentations vectorisées est évalué, et une alternative basée sur des embeddings BERT est étudiée pour capturer des sémantiques contextuelles. Les performances des MLP construits pour chaque famille de représentation sont comparées sur plusieurs architectures, adaptées à la quantité d'information fournie par la vectorisation (ou par BERT) et orientées vers une classification à trois classes. La profondeur est contrainte pour limiter le sur-apprentissage, et des couches de dropout sont insérées entre couches cachées afin de régulariser l'entraînement.

Enfin, afin d'améliorer la classification, des stratégies basées sur des grands modèles de langage (LLM) sont évaluées en utilisant la version API de Gemma 3-4b-it (Gemini), afin de comparer ses capacités de classification à celles des modèles entraînés. Par ailleurs, LoRA est utilisé pour réaliser un fine-tuning efficace de transformeurs basés sur BERT [3].

II. Q0—Analyse du jeu de données avec des modèles classiques d'apprentissage automatique

Le jeu de données *Sentiment Data Analysis* est composé de courts tweets en anglais, classés en trois catégories selon leur polarité : positif, neutre ou négatif (colonne `sentiment`). Des métadonnées sont également disponibles : le moment de la journée de publication, l'âge de l'utilisateur, et son pays d'origine (colonnes `Age of User` et `country`).

Un prétraitement du champ texte est d'abord effectué : suppression des valeurs nulles, puis suppression des *stopwords* avec NLTK, afin d'obtenir une colonne de texte traité débarrassée de mots très fréquents en anglais à faible contribution sémantique [4].

A. Analyse exploratoire (EDA)

La distribution des classes dans l'ensemble d'entraînement est analysée. Comme indiqué en Fig. 1, la classe neutre est la plus représentée, mais l'ensemble ne présente pas de déséquilibre substantiel susceptible d'induire un biais majeur des classifieurs.

Avant d'appliquer des stratégies d'apprentissage automatique, il est utile d'examiner le comportement des termes (sac de mots) vis-à-vis des classes de polarité, notamment via une analyse de fréquence.

Cela met en évidence le rôle d'un *vectorizer*, responsable de la conversion d'une collection de textes (telle que la colonne de texte traité) en vecteurs numériques. Le résultat est une représentation creuse (sparse) par document, appelée *vectorisation*. Selon la construction de cette matrice, plusieurs

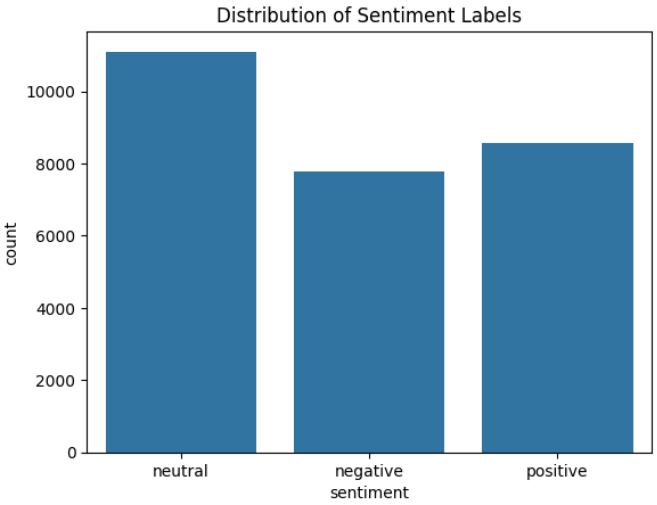


FIGURE 1 – Distribution des sentiments dans l'ensemble d'entraînement.

approches sont possibles. Ici, trois familles sont considérées : BoW (Bag of Words), TF–IDF, et TF–IDF caractères [5].

— **BoW.** Représente un document par l'occurrence des mots en n-grammes, en ignorant leur position. Un vocabulaire est construit, puis chaque document est encodé par des comptes. Dans scikit-learn, `CountVectorizer` "convertit une collection de documents texte en une matrice de comptes de tokens" [5].

— **TF–IDF.** Produit une matrice creuse comme BoW, mais pondère les termes par une fréquence inverse de document (IDF) afin de pénaliser les termes très fréquents [5]. Le poids TF–IDF d'un terme t dans un document d est donné par l'Eq. (1), avec $\mathrm{tf}(t,d)$ la fréquence du terme dans d et $\mathrm{idf}(t)$ une pondération liée à sa diffusion dans le corpus.

$$\mathrm{tfidf}(t,d) = \mathrm{tf}(t,d) \times \mathrm{idf}(t). \tag{1}$$

En pratique, une version lissée de l'IDF est souvent utilisée, Eq. (2), où n est le nombre total de documents et $\mathrm{df}(t)$ le nombre de documents contenant t.

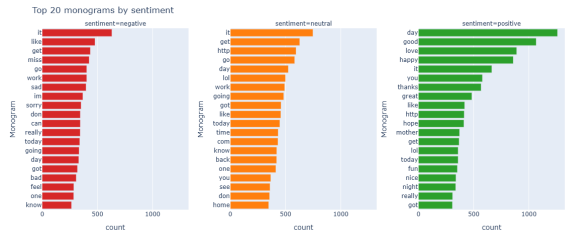
$$\mathrm{idf}(t) = \log\left(\frac{1+n}{1+\mathrm{df}(t)}\right) + 1. \tag{2}$$

Enfin, une normalisation par document stabilise l'échelle des caractéristiques. Une normalisation ℓ_2 est considérée, Eq. (3), où $\mathbf{v}$ est le vecteur TF–IDF d'un document.

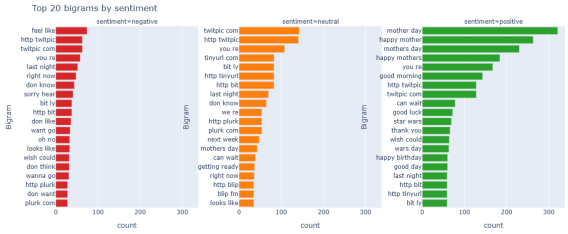
$$\mathbf{v}_{\mathrm{norm}} = \frac{\mathbf{v}}{\|\mathbf{v}\|_2} = \frac{\mathbf{v}}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}}. \tag{3}$$

— **TF–IDF caractères.** Applique TF–IDF à des n-grammes de caractères plutôt que de mots, contrôlé dans scikit-learn via le paramètre `analyzer` [6].

Après définition des représentations, BoW est utilisé pour extraire les 20 uni-grammes et bi-grammes les plus fréquents. Pour TF–IDF, l'objectif est d'afficher les 20 uni-grammes et bi-grammes de poids maximal. Les n-grammes de caractères ne sont pas utilisés à cette étape car ils sont moins interprétables pour l'analyse visée.

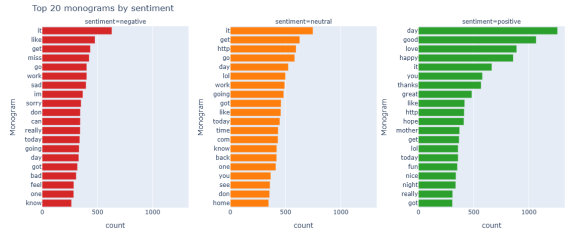


(a) Monogrammes

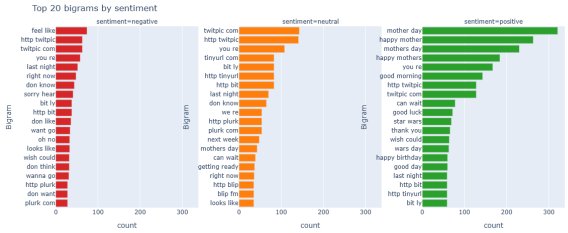


(b) Bigrams

FIGURE 2 – Top 20 n-grammes avec BoW.



(a) Monogrammes



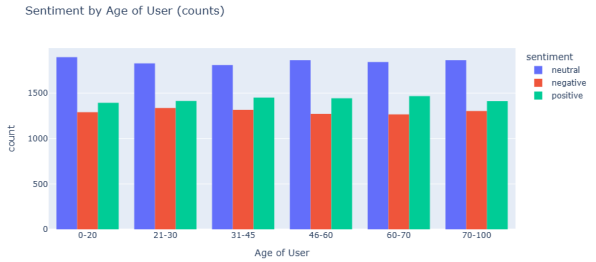
(b) Bigrams

FIGURE 3 – Top 20 n-grammes avec TF–IDF.

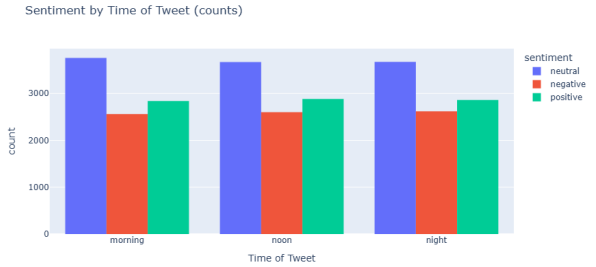
L'analyse des résultats des Figs. 2–3 montre que certains n-grammes sont cohérents avec la polarité associée (par exemple *bad* ou *sorry* pour la classe négative). Néanmoins, ces approches étant purement statistiques, des termes fréquents à faible valeur discriminante (ex. *it*) peuvent aussi apparaître. De plus, certains bigrammes a priori neutres (ex. *feels like*)

deviennent représentatifs en raison du contexte de collecte. Cela anticipe une limite des vectorisations par fréquence par rapport à des embeddings contextualisés.

Il est également utile de vérifier si des variables secondaires (âge, type de tweet, moment de la journée) ont un lien avec la polarité. Les distributions (Fig. 4) restent proches d'une catégorie à l'autre ; ces variables ne sont donc pas retenues pour l'entraînement.



(a) Sentiments par utilisateur



(b) Sentiments par type de tweet

FIGURE 4 – Distribution comparative des sentiments, stratifiée par utilisateur et type de tweet.

Concernant le pays, la Fig. 5 présente les pays les plus fréquents. Les différences de distribution restent modestes et de nombreux pays sont faiblement représentés, ce qui peut introduire des caractéristiques rares et un risque de sur-apprentissage. Cette colonne n'est donc pas retenue.

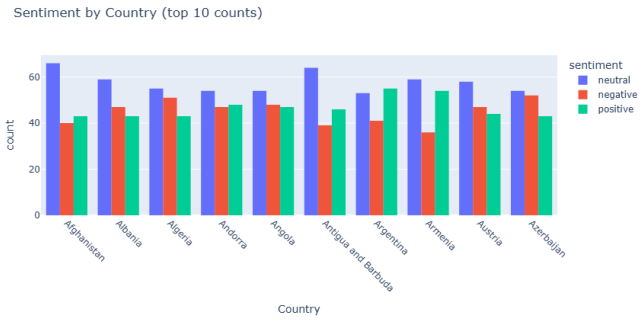


FIGURE 5 – Distribution des sentiments pour les 10 pays les plus représentés.

Enfin, la longueur du texte prétraité est examinée (Fig. 6). Les distributions par classe sont comparables ; cette variable n'est pas retenue.

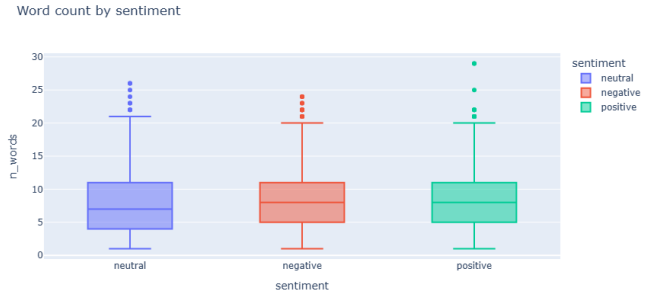


FIGURE 6 – Distribution du nombre de mots par sentiment.

B. Entraînement de modèles classiques

Après l'analyse précédente, des modèles supervisés classiques (hors deep learning) sont entraînés pour classer chaque message en trois polarités. Ces modèles s'appuient sur une représentation explicite obtenue via un vectorizer : représentation du document, apprentissage du classifieur, évaluation [7].

Pour chaque schéma de représentation, quatre classifieurs sont considérés : Multinomial Naive Bayes, régression logistique, SVM linéaire, et Random Forest. Une recherche d'hyperparamètres est effectuée sur un pipeline *vectorizer* + *modèle*, évaluée via validation croisée stratifiée. La meilleure configuration (selon une métrique cible) est ensuite réentraînée sur l'ensemble complet d'entraînement.

Les hyperparamètres principaux des vectorizers sont : `ngram_range` (tailles de n-grammes), `min_df` (seuil minimal de fréquence documentaire), `max_df` (seuil maximal), et `max_features` (limite de vocabulaire). Cette limite est particulièrement importante pour la TF–IDF caractères afin de contrôler la dimension et le coût.

Les hyperparamètres des modèles sont spécifiques et résumés ci-dessous :

— **Multinomial Naive Bayes** : `alpha` (lissage additif) pour éviter des probabilités nulles [8].
— **Régression logistique** : `C` (inverse de la régularisation), `penalty`, `solver`, `max_iter` [9].
— **SVM linéaire** : `C`, `estimator__loss`, `max_iter`. Le classifieur est calibré via `CalibratedClassifierCV` afin d'obtenir des probabilités de classe [11].
— **Random Forest** : `n_estimators`, `max_depth`, `min_samples_split`, `class_weight` [12].

Une grille est construite, la performance est estimée via validation croisée à 5 plis, puis la meilleure configuration est réentraînée sur l'ensemble complet. Les résultats et leur discussion sont présentés en Q1.

III. Q1—Classification avec des modèles classiques et analyse de performance

Au total, 12 configurations expérimentales sont entraînées (3 schémas de vectorisation $\times$ 4 modèles). L'objectif est de comparer ces combinaisons sur l'ensemble de test et de définir une base classique solide, servant ensuite de référence face à des méthodes basées sur des embeddings et des modèles de type BERT.

Les métriques rapportées sont l'Accuracy, le Macro-Recall et le Macro-F1. Les métriques macro sont particulièrement pertinentes car elles évaluent la performance de manière plus équilibrée entre classes, en réduisant le risque qu'une classe dominante pilote l'évaluation globale.

A. Temps d'entraînement

Chaque expérience basée sur `GridSearchCV` produit un fichier CSV contenant le détail de toutes les combinaisons testées, y compris les hyperparamètres du vectorizer et du classifieur. Ces CSV permettent une analyse systématique des changements de configuration et une sélection informée de la meilleure chaîne finale.

Dans ces fichiers, le champ `mean_fit_time` représente le temps moyen d'entraînement par pli pour chaque configuration évaluée. Il permet d'estimer le coût total associé à la recherche d'hyperparamètres.

1) Estimation du temps total (calcul séquentiel): L'estimation séquentielle est calculée comme la somme des temps moyens par pli, multipliée par le nombre de plis de validation croisée :

$$T_{\text{total}} \approx \sum_{i=1}^{n_{\text{candidates}}} \left(\texttt{mean\_fit\_time}_i \times n_{\text{splits}}\right), \qquad n_{\text{splits}} = 5. \tag{4}$$

Ici, $n_{\text{candidates}}$ correspond au nombre de lignes du CSV (nombre de combinaisons évaluées) :

— $n_{\text{candidates}} = 968$

— $T_{\text{total}} \approx 287{,}978.76$ s

2) Temps mur avec parallélisation: La valeur précédente correspond à un temps de calcul théorique séquentiel. Comme le parallélisme est activé avec `n_jobs=-1`, la charge est distribuée sur plusieurs cœurs CPU. Un temps mur approximatif peut être estimé par :

$$T_{\text{wall}} \approx \frac{T_{\text{total}}}{n_{\text{jobs}}}. \tag{5}$$

En supposant 8 cœurs effectifs :

$$T_{\text{wall}} \approx \frac{287{,}978.76}{8} \approx 35{,}997.35 \text{ s} \approx 10.00 \text{ h}.$$

B. Performance sur l'ensemble de test

La performance sur test est résumée par les résultats présentés dans les Figs. 7–9. Le Macro-F1, le Macro-Recall et l'Accuracy sont rapportés dans les Figs. 7, 8 et 9, respectivement. Pour chaque classifieur, les graphiques comparent les scores obtenus avec chaque méthode de vectorisation, ce qui rend les tendances faciles à identifier.

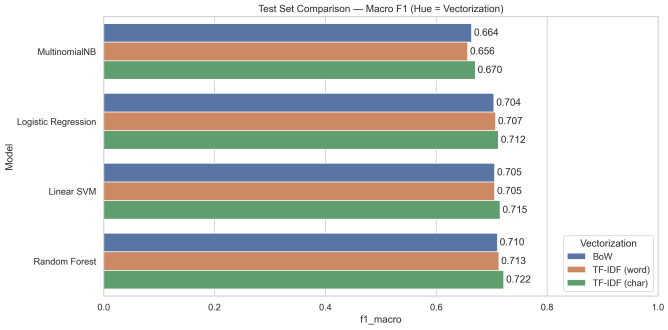


FIGURE 7 – Comparaison sur test — Macro-F1 (couleur = vectorisation).

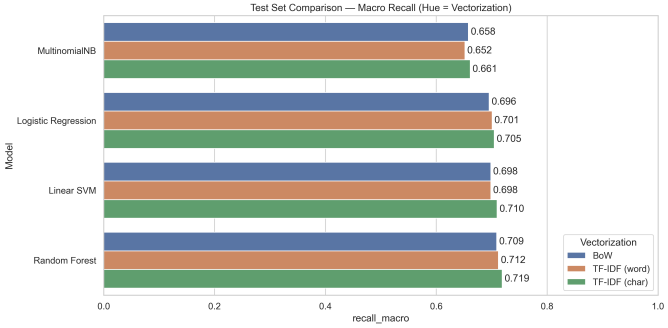


FIGURE 8 – Comparaison sur test — Macro-Recall (couleur = vectorisation).

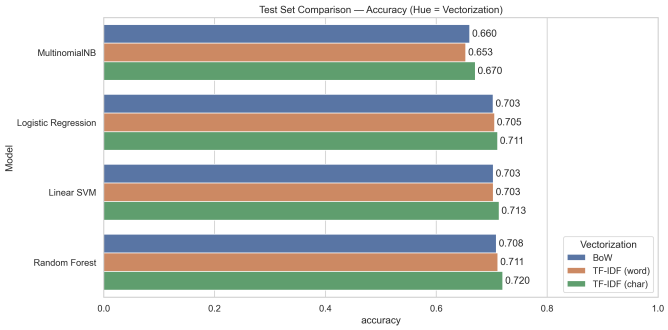


FIGURE 9 – Comparaison sur test — Accuracy (couleur = vectorisation).

1) Impact de la méthode de vectorisation: Globalement, les figures indiquent que TF–IDF caractères (TF–IDF char) tend à fournir les meilleures valeurs sur les trois métriques. Ce comportement est cohérent avec la nature des tweets (courts, informels), où la modélisation au niveau caractère capture des signaux sub-lexicaux importants : fautes d'orthographe, variations morphologiques, allongements (ex. "soooo good"), hashtags, abréviations, et indices préfixe/suffixe.

2) Comparaison des modèles et compromis: Parmi les classifieurs, le SVM linéaire et la régression logistique se distinguent par leur constance, en particulier combinés à TF–IDF (notamment TF–IDF char). Les représentations BoW/TF–IDF produisant des matrices de très grande dimension et

creuses, les modèles linéaires sont généralement efficaces et peu coûteux.

Bien que Random Forest soit très compétitif (voire meilleur), plusieurs limites sont importantes en haute dimension :

— **Scalabilité et coût :** l'entraînement de nombreux arbres dans un espace TF–IDF augmente fortement le temps et la mémoire, surtout sous `GridSearchCV`.
— **Risque de sur-apprentissage :** TF–IDF char introduit de nombreux n-grammes très spécifiques ; un modèle non-linéaire peut capturer des motifs accidentels et dégrader la robustesse hors-domaine.
— **Interprétabilité pratique plus faible :** une forêt est plus difficile à justifier qu'un modèle linéaire.

Ainsi, même en cas de gain marginal, les modèles linéaires restent attractifs pour leur stabilité, leur coût et leur clarté méthodologique.

IV. Q2—Architectures MLP pour la classification de sentiment de textes courts

Cette section étudie des architectures de perceptron multicouche (MLP) afin d'identifier efficacement la polarité de courts textes en anglais. Le pipeline considéré comprend : (i) une représentation (vectorisation) du texte ; (ii) la construction d'un `Dataset`/`DataLoader` ; (iii) l'entraînement d'un MLP ; puis (iv) l'évaluation.

Le premier élément du pipeline est le vectorizer (BoW, TF–IDF mot, ou TF–IDF caractère). Les hyperparamètres retenus pour chaque vectorizer correspondent à la meilleure configuration observée dans la grille associée à ce vectorizer au stade Q1.

Pour BoW et TF–IDF mot, le nombre de caractéristiques est de 6689. Pour TF–IDF caractères, le nombre de caractéristiques dépasse 100,000 si aucune limite n'est imposée. Afin de garder une dimension comparable et de rendre l'entraînement tractable, la configuration TF–IDF caractères est conservée, mais `max_features` est fixé à 10,000. Cela permet de comparer des architectures MLP équivalentes (mêmes profondeurs et têtes), en ne modifiant que la dimension d'entrée.

La matrice creuse produite par le vectorizer est convertie en format CSR, puis utilisée via un `SparseBoWDataset` [13]. Un `DataLoader` PyTorch itère ensuite sur l'ensemble d'entraînement ; les échantillons sont mélangés à chaque époque. La fonction `collate` densifie les lots et retourne un dictionnaire {`"x"` : `X_batch`, `"label"` : `y_batch`} directement utilisable pour le passage avant et le calcul de la perte [14].

A. Architectures MLP proposées (représentations creuses)

Afin de respecter les contraintes d'entrée et l'objectif de classification à trois classes, quatre réseaux de profondeur modérée (3–4 couches cachées) sont proposés. La profondeur est limitée car les représentations BoW/TF–IDF condensent déjà une partie de l'information discriminante au niveau des caractéristiques ; augmenter excessivement la profondeur tend à accroître la variance et le risque de sur-apprentissage, sans gain proportionnel en généralisation.

Les différences entre architectures portent principalement sur : (i) des structures en entonnoir (funnel) ; et (ii) le degré de régularisation via dropout.

La taille d'entrée dépend du vectorizer : 6689 pour BoW/TF–IDF mot, et 10,000 pour TF–IDF caractères. La taille de batch est fixée à 128.

— **MLP_1024_512_256_drop0_3**
— **MLP_2048_1024_512_drop0_2_gelu**
— **MLP_1536_768_384_192_drop0_25_SiLU**
— **MLP_4096_2048_1024_drop0_1_ReLU**

L'entraînement utilise une perte entropie croisée. Chaque réseau est entraîné jusqu'à 50 époques, avec un critère d'arrêt rapide à 1×10^{-4} basé sur la variation de perte entre deux époques consécutives. L'optimisation est effectuée avec Adam, avec un `lr` identique entre architectures afin de conserver des conditions comparables.

B. Motivation pour des embeddings contextualisés

Les vectorisations basées sur des fréquences fournissent une représentation sans information sémantique explicite : elles ne capturent que partiellement l'ordre et le contexte. Cela motive l'introduction d'une représentation dense incorporant le contexte via une tokenisation et un encodage par un modèle de type BERT.

C. Pipeline avec embeddings BERT

Changer de représentation implique d'adapter le `DataLoader` : au lieu de fournir des caractéristiques finales, il fournit des tenseurs tokenisés, car les embeddings sont produits à travers le passage avant de BERT. Dans ce cas, la fonction `collate` n'est pas requise, car le tokenizer produit des tenseurs de taille compatible. Le `DataLoader` retourne `"input_ids"`, `"attention_mask"` et `"label"` [15].

Une fonction d'extraction d'embeddings exécute BERT en mode évaluation, puis applique un pooling pour obtenir un vecteur par texte. Les lots sont concaténés pour former une matrice dense. Cette matrice est standardisée via `StandardScaler` (moyenne nulle, variance unité sur train), afin de stabiliser l'entraînement et d'éviter qu'une dimension domine par sa variance [16].

D. Architectures MLP au-dessus de BERT

Avec BERT, la dimension d'entrée devient fixe et faible (768 pour $\text{BERT}_{\text{BASE}}$). Les embeddings sont denses, contrairement aux vecteurs TF–IDF creux. La complexité représentative est essentiellement portée par l'encodeur (BERT), et la tête de classification peut rester simple ; dans certains cas, une tête linéaire suffit [17].

Les architectures considérées sont :

1) **LinearHead_baseline (single linear layer)**
2) **MLP_256_64_drop0_2**
3) **MLP_128_32_drop0_2**
4) **MLP_512_128_drop0_3**

La perte et la procédure d'entraînement restent identiques (entropie croisée, Adam, 50 époques max, arrêt rapide), afin de conserver une comparaison cohérente. Les résultats de l'ensemble des pipelines de cette section sont analysés en Q3.

V. Q3—Analyse comparative des performances MLP et implications pour la sélection de baseline

Cette section propose une analyse interprétative des résultats MLP introduits en Q2. Bien que Q1 se concentre sur des baselines classiques, il est utile d'évaluer si l'introduction de non-linéarité via des MLP apporte un gain significatif, et quelles familles de représentations en bénéficient. L'analyse s'appuie sur le Macro-F1, le Macro-Recall et l'Accuracy sur test, complétés par l'inspection des métriques d'entraînement afin de caractériser les écarts train–test.

A. Analyse préliminaire des MLP (Q2)

Les résultats sur test pour Macro-F1, Macro-Recall et Accuracy sont présentés en Fig. 10, Fig. 11 et Fig. 12.

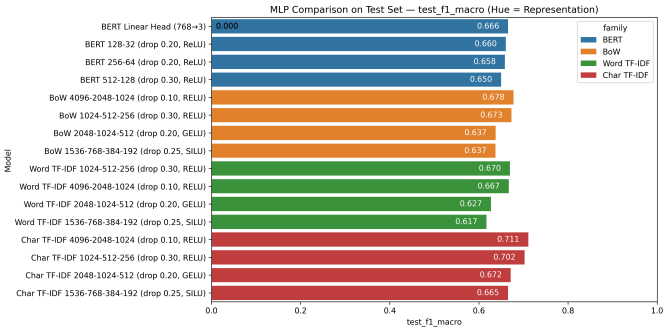


FIGURE 10 – Comparaison MLP sur test — Macro-F1 (couleur = famille de représentation).

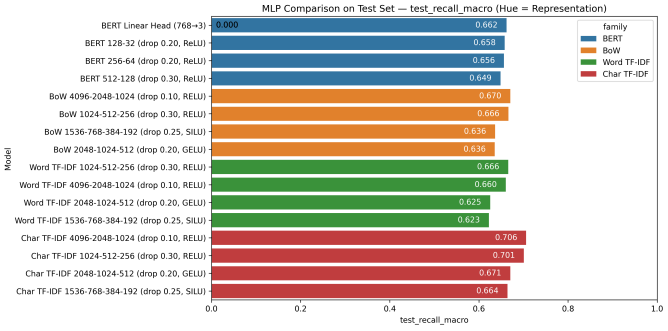


FIGURE 11 – Comparaison MLP sur test — Macro-Recall (couleur = famille de représentation).

Une tendance stable ressort : les configurations basées sur **Char TF–IDF** dominent la performance des MLP. En particulier, le MLP Char TF–IDF (4096–2048–1024, dropout 0.10, ReLU) atteint les meilleurs scores (Macro-F1 = 0.711, Macro-Recall = 0.706, Accuracy = 0.708). Une variante (1024–512–256, dropout 0.30, ReLU) reste proche (0.702 / 0.701 / 0.699). Cela suggère que, pour des tweets, les signaux au

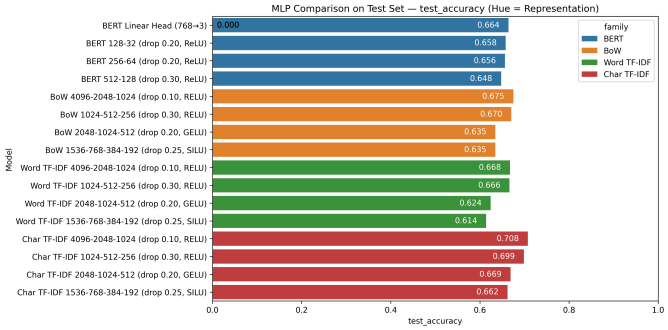


FIGURE 12 – Comparaison MLP sur test — Accuracy (couleur = famille de représentation).

niveau caractère (abréviations, variations orthographiques, motifs suffixe/préfixe, langage informel) restent très informatifs, y compris avec un classifieur non-linéaire.

Les familles **BoW** et **TF–IDF mot** constituent un second niveau, avec des meilleurs cas autour de 0.67–0.68. Enfin, les variantes **BERT** (têtes MLP au-dessus d'embeddings figés) restent compétitives mais plus limitées dans ce régime (environ 0.65–0.67), ce qui est cohérent avec l'utilisation d'embeddings contextualisés sans fine-tuning end-to-end de l'encodeur.

B. Comparaison par familles (qualité prédictive)

La comparaison au niveau des familles est cohérente sur les trois métriques : **Char TF–IDF** produit les MLP les plus performants et de manière régulière.

1) Char TF–IDF (meilleure famille en performance): Les MLP Char TF–IDF atteignent les meilleures performances. Le meilleur modèle est Char TF–IDF 4096–2048–1024 (dropout 0.10, ReLU) avec Macro-F1 ≈ 0.711, Macro-Recall ≈ 0.706 et Accuracy ≈ 0.708. Le comportement est cohérent avec le domaine tweet : la granularité caractère est robuste aux fautes, abréviations, allongements (ex. "soooo"), hashtags et variations d'écriture.

2) BoW et TF–IDF mot (niveau intermédiaire): Les MLP BoW et TF–IDF mot sous-performent Char TF–IDF. Le meilleur cas BoW atteint environ Macro-F1 ≈ 0.678, et le meilleur cas TF–IDF mot environ Macro-F1 ≈ 0.670. Sur des textes très courts, la présence/absence de termes saillants (BoW) peut être aussi utile qu'une pondération de rareté au niveau mot, tandis que les représentations mot restent sensibles au bruit orthographique.

3) BERT (compétitif mais non supérieur à Char TF–IDF dans ce régime): Pour les modèles BERT, le meilleur résultat est obtenu par l'option la plus simple : BERT + tête linéaire (768→3), avec Macro-F1 ≈ 0.666. L'ajout de couches MLP au-dessus des embeddings n'améliore pas les performances et tend à les dégrader légèrement. Une interprétation cohérente est que l'espace d'embeddings BERT est déjà raisonnablement séparable linéairement, et qu'augmenter la capacité de la tête ajoute des paramètres qui ne se traduisent pas en meilleure généralisation dans ce régime.

Pour caractériser la généralisation, les métriques d'entraînement sont examinées (Figs. 13–15).

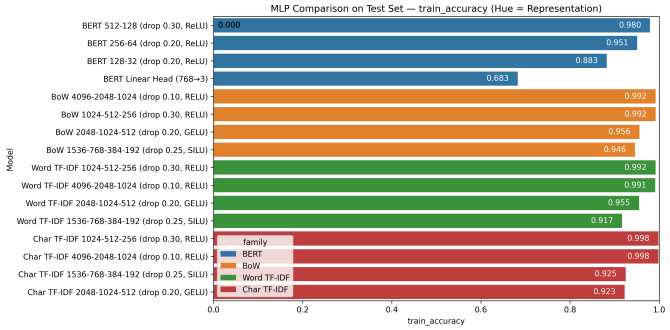


FIGURE 13 – Comparaison MLP sur train — Accuracy (couleur = famille de représentation).

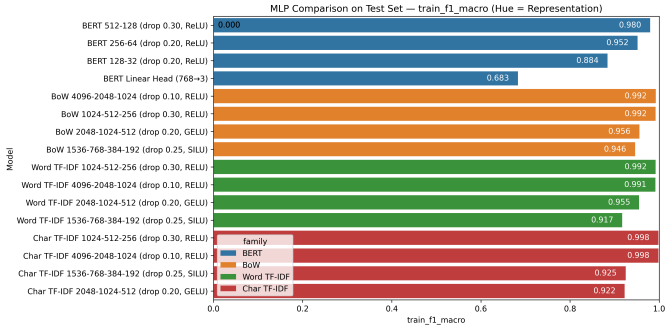


FIGURE 14 – Comparaison MLP sur train — Macro-F1 (couleur = famille de représentation).

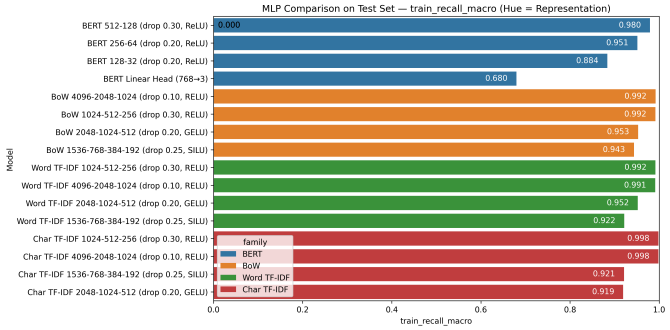


FIGURE 15 – Comparaison MLP sur train — Macro-Recall (couleur = famille de représentation).

Ces figures mettent en évidence un phénomène important : les MLP sur BoW/TF–IDF (mot ou caractère) atteignent souvent des scores très élevés sur train (parfois proches de 1.0), même pour des architectures modestes. Cela indique une forte capacité d'ajustement dans des espaces creux et de grande dimension. Toutefois, ce plafonnement sur train ne garantit pas un gain proportionnel sur test : un écart train–test demeure, ce qui justifie d'interpréter ces résultats comme un compromis (capacité élevée mais risque structurel de sur-apprentissage).

Ainsi, le meilleur MLP en performance sur test est Char TF–IDF 4096–2048–1024 (dropout 0.10). Il surpasse la meilleure variante BERT (tête linéaire). Néanmoins, du point de vue généralisation, les MLP sur TF–IDF requièrent un contrôle explicite (régularisation, arrêt anticipé, contrôle de capacité effective). En comparaison, BERT + tête linéaire constitue une alternative plus conservatrice : performance plus basse, mais structure plus simple et moins sensible à l'augmentation de capacité de la tête.

VI. Q4 : Analyse comparative avec des grands modèles de langage (LLM)

Cette analyse compare la meilleure architecture basée sur BERT à un grand modèle de langage moderne. L'objectif est d'évaluer si un modèle génératif généraliste, utilisé uniquement en inférence, peut rivaliser avec un encodeur spécialisé ayant été adapté à la tâche.

A. Méthodologie : inférence générative

Le modèle gemma-3-4b-it est retenu (instruction-tuned, $\sim$4 milliards de paramètres). Contrairement à la baseline BERT de cette partie — embeddings BERT figés avec une tête linéaire (`LinearHead(768→3)`) — le LLM est évalué via *few-shot prompting* avec l'API Google GenAI, sans mise à jour de paramètres.

Le protocole est défini comme suit :

- **Prompt engineering :** le modèle est explicitement instruit à se comporter comme un expert en analyse de sentiment.
- **Contexte few-shot :** trois exemples annotés (un par classe : Positive, Negative, Neutral) sont fournis avant la requête cible.
- **Contraintes de sortie :** la génération est contrainte à un label mono-mot mappé vers les classes numériques $(0, 1, 2)$.
- **Sous-ensemble d'évaluation :** pour des raisons de latence et de quotas API, l'évaluation est menée sur 1 000 exemples représentatifs du test.

B. Évaluation de performance

Les capacités génératives de Gemma sont comparées au modèle `BERT Linear Head`, qui présente la meilleure performance parmi les configurations BERT testées précédemment.

1) Résultats LLM (Gemma-3-4b-it): La Fig. 16 présente le rapport de classification du modèle génératif. Gemma atteint une **Accuracy de 0.632** et un **Macro-F1 de 0.630**.

La matrice de confusion met en évidence une faiblesse : une tendance à classer des tweets polarisés (positifs/négatifs) comme neutres. Ce comportement est souvent associé à l'alignement (RLHF) des modèles instruction-tuned, qui peut favoriser des sorties plus neutres en cas d'ambiguïté.

2) Résultats basés sur BERT: La Fig. 17 présente la performance de la baseline `BERT Linear Head`. Ce modèle atteint une **Accuracy de 0.664** et un **Macro-F1 de 0.666**.

Le modèle BERT présente une diagonale plus marquée dans la matrice de confusion, ce qui indique un meilleur pouvoir discriminant sur cette distribution de données.

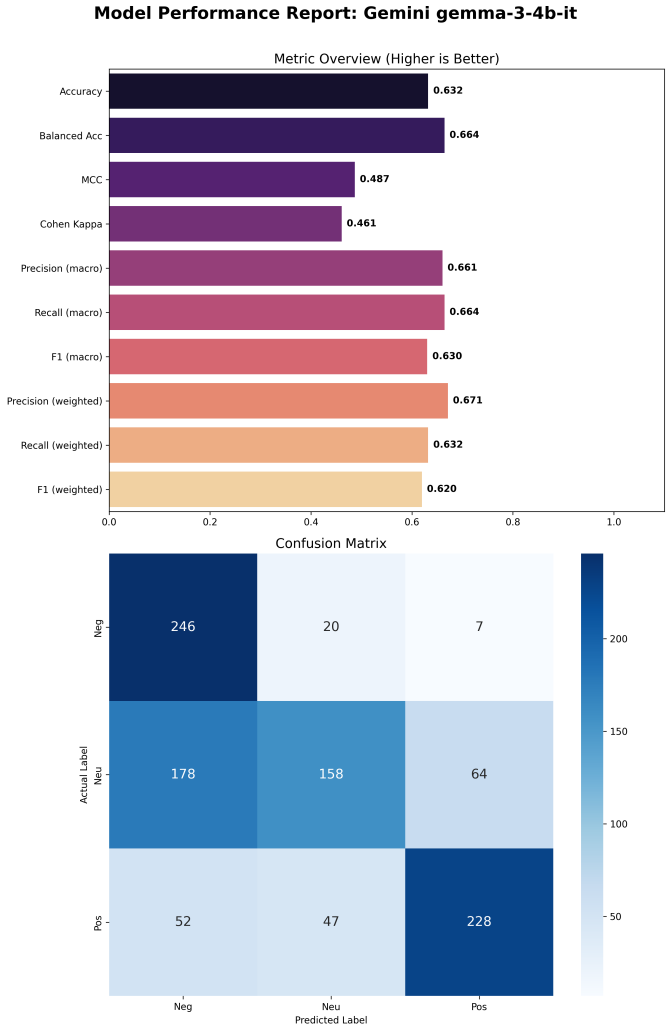


FIGURE 16 – Rapport de performance pour Gemma-3-4b-it (inférence few-shot). Le modèle présente une performance modérée et relativement équilibrée, avec une confusion notable entre sentiments polarisés et classe neutre.

TABLE I – Comparaison directe : encodeur fine-tuné vs LLM few-shot

Modèle	Params	Méthode	Macro F1
BERT (linéaire)	**110M**	**Fine-tuning**	**0.666**
Gemma-3-4b-it	4B	Few-shot	0.630

C. *Discussion : spécialisation vs généralisation*

La comparaison entre l'encodeur spécialisé (BERT) et le décodeur généraliste (Gemma) est résumée en Table I.

1) **Écart de performance :** BERT surpasse Gemma d'environ **3.6% en Macro-F1**. Cela confirme que, pour ce jeu de données, un modèle plus petit adapté au domaine est plus efficace qu'un modèle massif utilisé en few-shot.
2) **Efficacité de ressources :** l'écart est important. BERT (110M) peut être déployé avec une faible latence sur du matériel standard, tandis que Gemma (4B) requiert une VRAM significative ou une dépendance API.

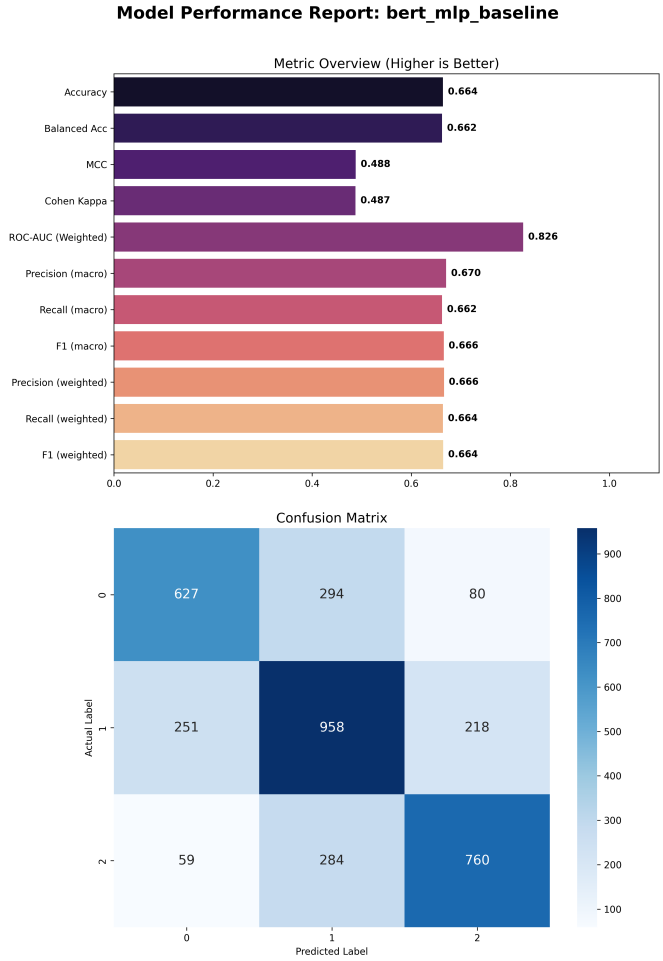


FIGURE 17 – Rapport de performance pour la baseline BERT Linear Head.

VII. Q5—EMBEDDINGS BERT VS REPRÉSENTATIONS BRUTES

Bien que la différence entre (i) des représentations fondées sur des fréquences (BoW/TF–IDF) et (ii) des embeddings BERT ait déjà été abordée en Q2, cette section détaille plus explicitement les différences conceptuelles entre ces familles.

BERT est un encodeur transformeur préentraîné pour produire des représentations contextuelles bidirectionnelles : chaque token est encodé en tenant compte du contexte à gauche et à droite [18]. Après pooling, un embedding dense de dimension fixe (768) est obtenu par texte, et sert d'entrée au classifieur. Cette réduction de dimension par rapport à TF–IDF est un premier facteur distinctif.

Pour quantifier l'effet sur la taille du classifieur, le nombre de paramètres d'une couche pleinement connectée est rappelé en Eq. (6) [19].

$$params = (infeatures \cdot outfeatures) + outfeatures \quad (6)$$

En appliquant cette formule à un réseau `INPUT_DIM` $\rightarrow 1024 \rightarrow 512 \rightarrow 256 \rightarrow 3$, un total de 10,897,923 paramètres est obtenu pour un MLP avec entrée à 10,000 dimensions (TF–IDF caractères limité), Eq. (7), alors que la même structure avec entrée à 768 dimensions (embedding BERT) conduit à 1,444,355 paramètres, Eq. (8), soit une réduction d'environ 86.75%. Cette réduction diminue le risque de sur-apprentissage et stabilise l'entraînement, tout en réduisant le coût de calcul pour la tête.

$$\begin{aligned} P_1\,(10000 \rightarrow 1024) &= 10000 \cdot 1024 + 1024 = 10{,}241{,}024, \\ P_2\,(1024 \rightarrow 512) &= 1024 \cdot 512 + 512 = 524{,}800, \\ P_3\,(512 \rightarrow 256) &= 512 \cdot 256 + 256 = 131{,}328, \\ P_4\,(256 \rightarrow 3) &= 256 \cdot 3 + 3 = 771, \\ P_{\text{total}} &= P_1 + P_2 + P_3 + P_4 = 10{,}897{,}923. \end{aligned} \quad (7)$$

$$\begin{aligned} P_1\,(768 \rightarrow 1024) &= 768 \cdot 1024 + 1024 = 787{,}456, \\ P_2\,(1024 \rightarrow 512) &= 1024 \cdot 512 + 512 = 524{,}800, \\ P_3\,(512 \rightarrow 256) &= 512 \cdot 256 + 256 = 131{,}328, \\ P_4\,(256 \rightarrow 3) &= 256 \cdot 3 + 3 = 771, \\ P_{\text{total}} &= P_1 + P_2 + P_3 + P_4 = 1{,}444{,}355. \end{aligned} \quad (8)$$

Les vectorisations par fréquences produisent typiquement une matrice creuse : l'information se concentre sur peu d'indices actifs. Cela rend les MLP sensibles à la largeur et à la régularisation, et complique la sélection d'architecture, car des variations modestes peuvent accroître le sur-apprentissage. À l'inverse, les embeddings BERT sont denses et de faible dimension, ce qui permet des têtes plus simples et des choix de régularisation moins critiques.

Enfin, la différence la plus marquée concerne la sensibilité au contexte : BERT encode le contexte via un préentraînement sur de grands corpus, tandis que BoW/TF–IDF restent des représentations essentiellement statistiques. Cette propriété permet d'explorer des classifieurs plus simples au-dessus d'embeddings BERT (parfois une tête linéaire), car une part importante de la complexité est déjà absorbée par l'encodeur.

Il est cependant important de distinguer l'extraction d'embeddings figés du fine-tuning : adapter l'encodeur à la tâche (par exemple via LoRA, Q7) permet d'aligner les représentations internes avec la distribution de données et l'objectif de classification. Cela atténue l'effet *domain-agnostic* d'un extracteur figé et peut améliorer substantiellement la performance.

VIII. Q6—Architecture BERT et cadre théorique

BERT (*Bidirectional Encoder Representations from Transformers*) marque un changement de paradigme en apprentissage de représentations de langage. Contrairement à des modèles unidirectionnels (p. ex. GPT) ou à des concaténations peu profondes (p. ex. ELMo), BERT préentraîne des représentations profondes bidirectionnelles en conditionnant simultanément à gauche et à droite à tous les niveaux [20].

A. Architecture du modèle

L'architecture est un encodeur Transformer bidirectionnel, basé sur Vaswani et al. [21], composé d'une pile de L blocs identiques.

Deux tailles sont classiquement considérées [20] :

— **$\text{BERT}_{\text{BASE}}$:** $L = 12$, taille cachée $H = 768$, $A = 12$ têtes d'attention, ∼110M paramètres.

— **$\text{BERT}_{\text{LARGE}}$:** $L = 24$, $H = 1024$, $A = 16$, ∼340M paramètres.

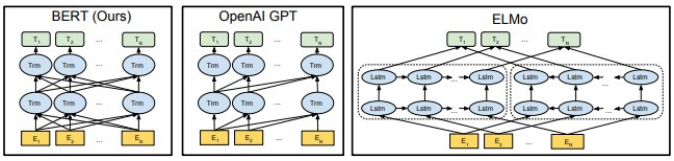


FIGURE 18 – Différences d'architectures de préentraînement. BERT utilise un Transformer bidirectionnel [20].

Chaque bloc comprend (i) une attention multi-têtes et (ii) un réseau feed-forward positionnel, avec connexions résiduelles et normalisation de couche.

1) Attention multi-têtes: La sortie est donnée par une attention à produit scalaire normalisé :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (9)$$

où d_k est la dimension des clés, introduite comme facteur d'échelle.

2) Réseau feed-forward et activation GELU: Le FFN applique deux transformations linéaires avec une non-linéarité GELU :

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2, \quad (10)$$

avec une dimension intermédiaire de 3072 pour $\text{BERT}_{\text{BASE}}$.

3) Connexions résiduelles et normalisation:

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x)). \quad (11)$$

B. Représentation d'entrée

1) Tokenisation WordPiece: BERT utilise un vocabulaire WordPiece (30 000 tokens), limitant l'OOV via des sous-mots (ex. "playing" → `play` + `##ing`).

2) Somme d'embeddings: Chaque token est représenté par la somme :

$$\mathbf{E} = \mathbf{E}_{token} + \mathbf{E}_{segment} + \mathbf{E}_{position}. \quad (12)$$

Comme illustré en Fig. 19, l'embedding d'entrée de chaque token combine : (i) l'identité lexicale (token), (ii) l'appartenance à un segment (A/B), et (iii) la position (ordre). Chaque séquence commence par le token `[CLS]` ; l'état caché final associé à `[CLS]` ($C \in \mathbb{R}^H$) est couramment utilisé comme représentation agrégée pour la classification. Le token `[SEP]` sert à délimitation/séparation des séquences.

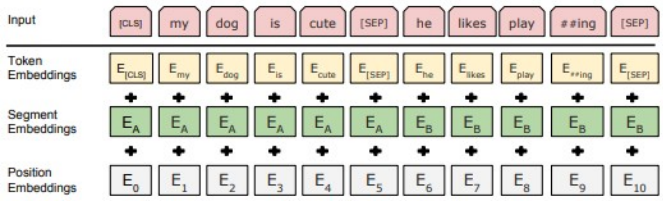


FIGURE 19 – BERT input representation [20].

C. Objectifs de préentraînement

BERT est préentraîné sur BooksCorpus et Wikipedia via deux tâches : MLM et NSP.

1) Masked Language Model (MLM): 15% des tokens sont sélectionnés : 80% remplacés par `[MASK]`, 10% par un token aléatoire, 10% inchangés, afin de réduire le décalage préentraînement/fine-tuning.

2) Next Sentence Prediction (NSP): BERT prédit si une phrase B suit A (IsNext) ou est aléatoire (NotNext), afin de modéliser des relations inter-phrases.

D. Stratégies d'application

BERT peut être utilisé en fine-tuning end-to-end ou en approche *feature-based* (extraction de caractéristiques). La Fig. 20 illustre des résultats en NER où la concaténation des quatre dernières couches est compétitive.

System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	**93.1**
Fine-tuning approach		
$\text{BERT}_{\text{LARGE}}$	96.6	92.8
$\text{BERT}_{\text{BASE}}$	96.4	92.4
Feature-based approach ($\text{BERT}_{\text{BASE}}$)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

FIGURE 20 – Approche feature-based sur CoNLL-2003 NER.

IX. Q7—Fine-tuning avec LoRA pour la classification de sentiment

LoRA (*Low-Rank Adaptation*) est une méthode de *Parameter-Efficient Fine-Tuning* (PEFT) permettant d'adapter des modèles préentraînés (p. ex. Transformers) sans mettre à jour l'ensemble des paramètres. Les poids d'origine sont gelés et de petites matrices de rang faible, entraînables, sont injectées dans certaines couches (souvent des projections linéaires de l'attention). Cela réduit fortement le nombre de paramètres entraînables et le coût d'entraînement/mémoire, tout en conservant la capacité de spécialisation à la tâche.

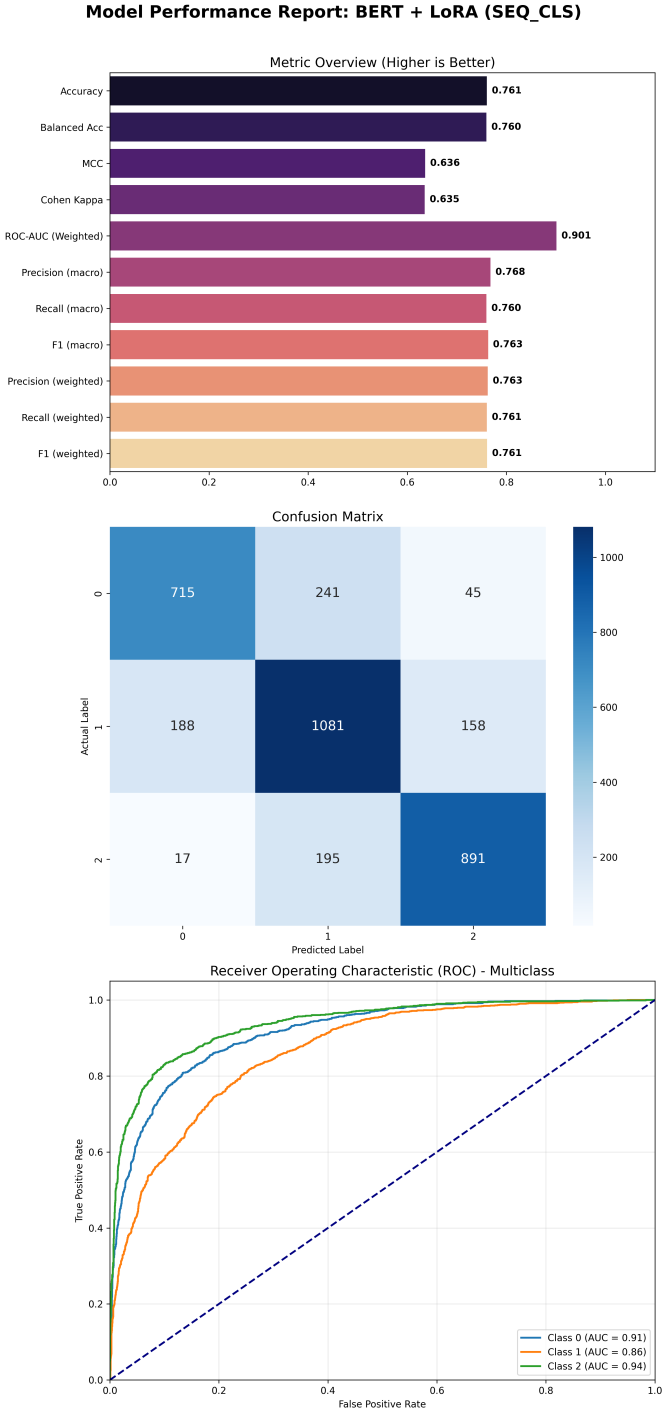


FIGURE 21 – Rapport d'évaluation complet pour **BERT + LoRA (SEQ_CLS)** sur test.

A. Aperçu d'implémentation

Le pipeline suit un schéma standard : (1) préparation du jeu de données et split *train/validation/test* ; (2) tokenisation via le tokenizer du modèle de base ; (3) chargement d'un modèle de classification (`AutoModelForSequenceClassification`) ; (4)

injection des adaptateurs LoRA via `get_peft_model` ; (5) entraînement avec `Trainer` et `TrainingArguments` ; (6) évaluation finale sur test (voir Fig. 21).

B. Configuration LoRA (justification)

La configuration retenue est :

- **`task_type=TaskType.SEQ_CLS`** : PEFT configuré pour la classification de séquences.
- **`target_modules=["query","value"]`** : injection dans des projections influentes (Q/V), bon compromis capacité/coût.
- **`r=8`** : rang (capacité) modéré, adapté à l'analyse de sentiment.
- **`lora_alpha=16`** : facteur d'échelle des mises à jour LoRA.
- **`lora_dropout=0.1`** : régularisation du chemin LoRA pour limiter le sur-apprentissage.

C. Résultats sur test : BERT + LoRA (SEQ_CLS)

Le modèle **BERT + LoRA (SEQ_CLS)** obtient :

- Accuracy = 0.761
- Macro-Recall = 0.760
- Macro-F1 = 0.763

Il rapporte en outre un ROC-AUC pondéré de 0.901, et des AUC par classe de 0.91 (classe 0), 0.86 (classe 1) et 0.94 (classe 2), ce qui indique une bonne séparabilité globale (Fig. 21).

D. Comparaison directe avec la baseline embeddings BERT (LinearHead 768→3)

Par rapport à la baseline précédente (*Approach = MLP ; Representation = BERT ; Model = LinearHead(768→3)*), qui rapporte Acc = 0.664, Macro-R = 0.662, et Macro-F1 = 0.666, LoRA améliore de manière cohérente toutes les métriques :

- **Accuracy :** 0.761 vs. $0.664 \Rightarrow +0.097$ (environ $+14.6\%$ relatif)
- **Macro-Recall :** 0.760 vs. $0.662 \Rightarrow +0.098$ (environ $+14.8\%$ relatif)
- **Macro-F1 :** 0.763 vs. $0.666 \Rightarrow +0.097$ (environ $+14.6\%$ relatif)

Ces gains sont cohérents avec l'objectif de LoRA : au lieu d'entraîner uniquement une tête sur des embeddings figés, l'encodeur est légèrement adapté via des mises à jour de rang faible dans des modules d'attention, ce qui améliore la représentation interne pour la classification.

E. Positionnement dans le projet

Avec Macro-F1 = 0.763, **BERT + LoRA (SEQ_CLS)** dépasse la baseline BERT + LinearHead ainsi que les meilleurs pipelines classiques/MLP rapportés auparavant (cf. Table **??**). Ce modèle constitue donc, à ce stade, le meilleur modèle entraîné du projet.

X. Conclusion

Cette section vise à établir un ensemble de conclusions concernant la mise en œuvre de méthodes de classification de sentiments pour de courts textes en anglais. Tout d'abord, il est important de souligner l'analyse réalisée durant la phase préliminaire du jeu de données et de ses caractéristiques principales. À partir de cette analyse, on observe que, compte tenu des informations d'entraînement disponibles, la variable d'intérêt principale pour entraîner ce type de classifieur est sans ambiguïté le texte prétraité après suppression des stopwords. En effet, la distribution des classes vis-à-vis des autres variables du jeu de données reste soit constante (comme pour les tranches d'âge), soit peut induire un risque accru de mémorisation de motifs non pertinents, car les éléments de chaque classe au sein de ces catégories ne sont pas représentatifs de l'ensemble de la collecte (comme pour la variable pays).

Concernant l'entraînement des modèles classiques d'apprentissage automatique, les résultats atteignent de bonnes performances, principalement grâce à l'évaluation de multiples pipelines combinant un vectoriseur et un classifieur, ainsi qu'aux variations de leurs grilles d'hyperparamètres. Cela conduit à un total de 956 modèles entraînés, dont les meilleures configurations sont sélectionnées. De plus, parmi les trois types de représentations, les meilleurs résultats sont obtenus avec le SVM, la régression logistique et la forêt aléatoire, avec de légers avantages en F1 et en accuracy pour la forêt aléatoire dans la plupart des cas. Néanmoins, des modèles tels que la régression logistique et le SVM sont plus intelligibles et interprétables. Par conséquent, et considérant que les différences de performance entre la forêt aléatoire et la régression logistique / le SVM ne sont pas suffisamment marquées pour conclure que la forêt aléatoire est la meilleure option en NLP, il est important de noter que, malgré des métriques légèrement supérieures, la forêt aléatoire implique un coût d'entraînement sensiblement plus élevé en temps de calcul et en ressources.

Bien que, dans le cas des modèles classiques, l'importance de la méthode de vectorisation et de son choix soit abordée, l'utilisation d'une recherche d'hyperparamètres permet de limiter cet effet lors de la sélection d'un pipeline, puisque les paramètres du vectoriseur font partie intégrante de la procédure de recherche. En revanche, dans le cas des MLP, le nombre plus réduit de configurations candidates, l'influence de chaque type de vectorisation sur la structure du réseau, ainsi que l'intégration d'embeddings BERT figés avant l'entraînement, conduisent à considérer non seulement le meilleur modèle, mais plus largement le meilleur pipeline, incluant à la fois la représentation et la classification.

Lors de l'analyse des résultats de test des MLP entraînés avec des entrées TF–IDF au niveau des caractères, l'écart entre les performances d'entraînement et de test indique une forte capacité d'ajustement et, par conséquent, un risque élevé de surapprentissage. Des modifications de l'architecture du réseau, principalement en termes de largeur des couches et de régularisation, compte tenu de la matrice creuse compressée

en entrée, ne se traduisent pas toujours par des améliorations proportionnelles sur le jeu de test, car le risque de surapprentissage augmente. À l'inverse, l'utilisation d'embeddings BERT figés, bien qu'elle présente des performances attendues avec un nombre de paramètres inférieur à celui des modèles TF–IDF caractères, bénéficie du caractère dense de la matrice d'entrée et d'une complexité nettement réduite du classifieur par rapport aux données d'entraînement. Cela rend les structures plus simples associées à cette représentation particulièrement attractives, comme dans le cas d'une couche de décision unique pour classifier les vecteurs produits par BERT. En effet, pour ce type de représentation, l'augmentation de la profondeur et de la complexité globale du réseau peut être associée à une dégradation des performances sur le jeu de test, précisément parce que, compte tenu des propriétés de BERT et de la densité d'information des embeddings, la structure du réseau et le nombre de paramètres nécessaires peuvent rester limités.

Après l'entraînement des MLP, les modèles classiques, en combinaison avec la vectorisation TF–IDF au niveau des caractères, conservent de meilleures performances que les MLP, y compris ceux utilisant des embeddings BERT, pour deux raisons principales. La première est la rigueur de la recherche d'hyperparamètres réalisée pour les modèles classiques. La seconde est que, pour les modèles basés sur des embeddings BERT, il existe encore une marge d'amélioration, en particulier pour la structure la plus conservatrice qui présente de meilleures performances et un risque réduit de surapprentissage ; cette amélioration est rendue possible par le fine-tuning via LoRA, qui permet d'adapter le processus de représentation au domaine spécifique du jeu de données d'entraînement. Les résultats obtenus montrent que, parmi l'ensemble des modèles évalués, le modèle BERT affiné avec LoRA atteint la meilleure performance en termes de F1 et d'accuracy avec une marge notable ; il constitue donc une structure fortement recommandée pour développer des modèles de classification de courts textes fondés sur la polarité ou le sentiment.

RÉFÉRENCES

[1] T. Finn and A. Downie, "How can sentiment analysis be used to improve customer experience ?," *IBM Think*, accessed Jan. 18, 2026. [Online]. Available : https ://www.ibm.com/think/insights/how-can-sentiment-analysis-be-used-to-improve-customer-experience

[2] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and Trends in Information Retrieval*, vol. 2, no. 1–2, pp. 1–135, 2008.

[3] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA : Low-Rank Adaptation of Large Language Models," *arXiv preprint* arXiv :2106.09685, Jun. 2021, doi : 10.48550/arXiv.2106.09685.

[4] The NLTK Project, "Sample usage for corpus," *NLTK Documentation*, accessed Jan. 18, 2026. [Online]. Available : https ://www.nltk.org/howto/corpus.html

[5] scikit-learn developers, "Feature extraction," *scikit-learn Documentation*, accessed Jan. 18, 2026. [Online]. Available : https ://scikit-learn.org/stable/modules/feature_extraction.html

[6] scikit-learn developers, "sklearn.feature_extraction.text.TfidfVectorizer," *scikit-learn Documentation*, accessed Jan. 18, 2026. [Online]. Available : https ://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[7] F. Sebastiani, "Machine Learning in Automated Text Categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, Mar. 2002, doi : 10.1145/505282.505283. [Online]. Available : https ://nmis.isti.cnr.it/sebastiani/Publications/ACMCS02.pdf

[8] scikit-learn developers, "sklearn.naive_bayes.MultinomialNB," *scikit-learn Documentation*. [Online]. Available : https ://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html. Accessed : Jan. 18, 2026.

[9] scikit-learn developers, "sklearn.linear_model.LogisticRegression," *scikit-learn Documentation*. [Online]. Available : https ://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. Accessed : Jan. 18, 2026.

[10] scikit-learn developers, "sklearn.svm.LinearSVC," *scikit-learn Documentation*, accessed Jan. 18, 2026. [Online]. Available : https ://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[11] scikit-learn developers, "sklearn.calibration.CalibratedClassifierCV," *scikit-learn Documentation*, accessed Jan. 18, 2026. [Online]. Available : https ://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html

[12] scikit-learn developers, "sklearn.ensemble.RandomForestClassifier," *scikit-learn Documentation*, accessed Jan. 18, 2026. [Online]. Available : https ://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[13] SciPy developers, "scipy.sparse.csr_matrix," *SciPy Documentation*. [Online]. Available : https ://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html. Accessed : Jan. 18, 2026.

[14] PyTorch contributors, "Data Loading and Processing Tutorial," *PyTorch Tutorials*. [Online]. Available : https ://docs.pytorch.org/tutorials/beginner/data_loading_tutorial.html. Accessed : Jan. 18, 2026.

[15] Hugging Face, "BERT," *Transformers Documentation*. [Online]. Available : https ://huggingface.co/docs/transformers/en/model_doc/bert. Accessed : Jan. 18, 2026.

[16] scikit-learn developers, "sklearn.preprocessing.StandardScaler," *scikit-learn Documentation*, accessed Jan. 18, 2026. [Online]. Available : https ://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[17] Sutriawan, Supriadi Rustad, Guruh Fajar Shidik, and Pujiono, "Performance Evaluation of Text Embedding Models for Ambiguity Classification in Indonesian News Corpus : A Comparative Study of TF-IDF, Word2Vec, FastText, BERT, and GPT," *Ingénierie des Systèmes d'Information*, vol. 30, no. 6, pp. 1469–1482, June 2025, doi : 10.18280/isi.300606. [Online]. Available : https ://www.iieta.org/journals/isi/paper/10.18280/isi.300606

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding," *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, Jun. 2019, doi : 10.18653/v1/N19-1423. [Online]. Available : https ://aclanthology.org/N19-1423/

[19] PyTorch contributors, "torch.nn.Linear," *PyTorch Documentation*, accessed Jan. 18, 2026. [Online]. Available : https ://docs.pytorch.org/docs/stable/generated/torch.nn.Linear.html

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, vol. 1, Minneapolis, Minnesota, Jun. 2019, pp. 4171–4186.

[21] A. Vaswani et al., "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, Long Beach, CA, Dec. 2017.