Projet NLP : Classifieur de texte par langue

Saad TAHARRAOUI
CentraleSupélec
saad.taharraoui@student-cs.fr

Fabien LABARRE
CentraleSupélec
fabien.labarre@student-cs.fr

Abstract

Dans le cadre de cette compétition Kaggle, nous avons développé un classifieur de langues et ceci en plusieurs étapes. Tout d'abord, nous avons effectué un travail de nettoyage des données, visant à préparer et à structurer les textes afin d'optimiser la performance du modèle. Ensuite, nous avons fine-tuné un modèle préexistant, le DistilBERT (version distilbert-base-multilingual-cased), qui a été ajusté pour la classification de texte et cela en rajoutant une couche de classification afin de mieux adapter le modèle aux spécificités de notre tâche.

1 Contexte

Nous avons été amenés à travailler sur deux jeux de données principaux : un jeu de données d'entraînement, contenant 190 599 phrases, chacune étiquetée avec la langue correspondante, et un jeu de données de test pour lequel nous devons prédire la langue des phrases. L'objectif principal étant de développer un modèle capable de classer efficacement la langue d'un texte.

La première étape de notre démarche a été l'exploration des données. Cette étape nous a permis de nous familiariser avec les défis que présentait le projet, comme la gestion des déséquilibres potentiels entre les différentes langues et l'optimisation des performances du modèle en fonction des données disponibles. Nous avons aussi effectué un nettoyage des données pour garantir que les entrées soient de qualité optimale avant d'entraîner notre modèle.

2 Exploration des données

2.1 Valeurs manquantes

Nous avons d'abord examiné les valeurs manquantes dans le jeu de données d'entraînement. En analysant les informations, nous avons constaté qu'il y avait 500 valeurs manquantes dans la colonne des labels. Pour pallier ce problème, nous avons choisi de remplir ces valeurs manquantes par le label 'unk', signifiant "inconnu", afin de ne pas perdre de données pendant la phase de traitement. Bien que cette approche soit utile pour commencer, elle pourra être réévaluée plus tard, notamment après l'étape d'embedding, où des méthodes plus sophistiquées pourraient être envisagées pour gérer ces valeurs manquantes de manière plus ciblée.

2.2 Doublons

Lors de l'exploration des données, nous avons observé la présence de quelques doublons (76) dans les phrases du jeu de données. Cependant, ces doublons étaient associés à des étiquettes de langue différentes. Cela pourrait indiquer que, bien que les phrases soient identiques, elles pourraient provenir de dialectes différents ou de contextes linguistiques variés. Après réflexion, nous avons choisi de maintenir ces doublons dans leur état actuel, en considérant que ces variations de langue pourraient refléter des subtilités de dialectes ou de variations régionales au sein d'une même langue.

2.3 Distribution des langues

Lors de l'exploration de la distribution des labels, nous avons observé qu'il y avait 390 labels dans notre jeu de données, ce qui représente une grande diversité linguistique. Cependant, cette diversité s'accompagne d'un déséquilibre, certaines langues étant largement sous-représentées, tandis que d'autres apparaissent beaucoup plus fréquemment. Après avoir visualisé cette distribution à l'aide d'un barplot, nous avons calculé quelques statistiques, comme la moyenne, la médiane, et les quartiles. Nous avons constaté que la majorité des langues ont un nombre d'occurrences compris entre 13 et 456 (entre le premier et le troisième quartile). Cependant, certaines langues présentent un nombre d'occurrences exceptionnellement élevé, atteignant jusqu'à 1500.

En plus, certaines langues sont extrêmement

rares, avec seulement quelques occurrences, comme celles ayant seulement 1 à 15 occurrences. Afin de rendre la répartition des données plus équilibrée et de se concentrer sur les langues les plus représentées, nous avons décidé de supprimer les langues les moins fréquentes, dont le nombre d'occurrences est inférieur à 200. Après avoir retiré ces langues, le nombre de labels a été réduit à 361, ce qui a permis de rendre le problème un peu plus équilibré en termes de représentation des différentes langues.

3 Modèlisation

3.1 Modèle pré-entrainé

Le modèle que nous utilisons , DistilBERT, est une version allégée de BERT qui a été optimisée pour être plus rapide tout en conservant une grande partie de la performance du modèle original. Il a été entraîné sur des textes provenant de Wikipedia dans 104 langues, ce qui lui permet de traiter des tâches de traitement du langage naturel dans plusieurs langues. Bien qu'il présente une légère perte de précision par rapport à son prédécesseur mBERT, il offre un bon compromis entre efficacité et performances, notamment pour des applications multilingues.

Il est conçu pour traiter des tâches de traitement du langage naturel telles que la classification de séquences, la classification de tokens ou la réponse à des questions

Nous avons commencé par charger le tokenizer et le modèle DistilBERT multilingue, qui est particulièrement adapté pour les tâches de classification de séquences, comme la nôtre. Nous avons modifié la couche de sortie du modèle avec un nombre de labels correspondant exactement au nombre de langues présentes dans notre jeu de données, afin de bien aligner les classes cibles avec ce qu'on veut prédire.

Ensuite, pour améliorer l'efficacité de l'entraînement, nous avons décidé de figer les poids des couches du modèle préexistant. Concrètement, cela signifie que nous avons figé tous les paramètres du modèle DistilBERT. Cela nous permet de ne pas recalculer les gradients pour ces paramètres lors de l'entraînement, et de se concentrer uniquement sur l'apprentissage des couches de classification, tout en tirant parti des connaissances déjà acquises par DistilBERT à partir de ses données multilingues. C'est une manière de gagner en efficacité, tout en utilisant un modèle qui a déjà beaucoup appris : le transfer learning

3.2 preprocessing des données

Dans cette étape, nous avons préparé et prétraité nos données pour l'entraînement de notre modèle de classification multilingue. Nous avons d'abord effectué un split stratifié des données afin de garantir que la distribution des labels soit la même entre les ensembles d'entraînement, de validation et de test. Ensuite, nous avons utilisé un tokenizer pour transformer les textes en une séquence de tokens compréhensibles par notre modèle. Afin d'assurer une gestion efficace des séquences de textes, nous avons ajusté la longueur maximale des séquences à 512 tokens et appliqué un padding pour garantir des dimensions uniformes. Les labels ont été mappés en valeurs numériques à l'aide d'un dictionnaire pour rendre le modèle compatible avec des labels sous forme d'entiers. Après avoir converti ces labels en tenseurs PyTorch, nous avons créé des datasets contenant les encodages des textes et les labels associés. Ces datasets ont ensuite été organisés en DataLoaders pour charger efficacement les données par batchs pendant l'entraînement.

3.3 Stratégie d'apprentissage

Notre stratégie d'apprentissage repose sur un suivi rigoureux de la performance avec un mécanisme d'early stopping, l'utilisation d'optimiseur et de fonction de perte adaptés, et une visualisation des métriques clés pour évaluer le succès de l'entraînement.

Nous utilisons l'optimiseur AdamW, qui est bien adapté aux modèles de type Transformer, avec un taux d'apprentissage de 5e-5. Pour la fonction de perte, nous avons choisi la CrossEntropyLoss, qui est couramment utilisée pour les tâches de classification multi-classes.

Nous avons défini un nombre d'Epochs de 50, mais pour éviter le sur-apprentissage, nous avons introduit une stratégie d'early stopping. Si le F1-score sur l'ensemble de validation ne montre pas d'amélioration pendant 5 epochs consécutives, l'entraînement s'arrête prématurément, ce qui permet de gagner du temps et d'éviter le surapprentissage du modèle.

4 Résulats

Le modèle a été entraîné sur 20 epochs en utilisant les GPU de Kaggle. L'analyse des courbes

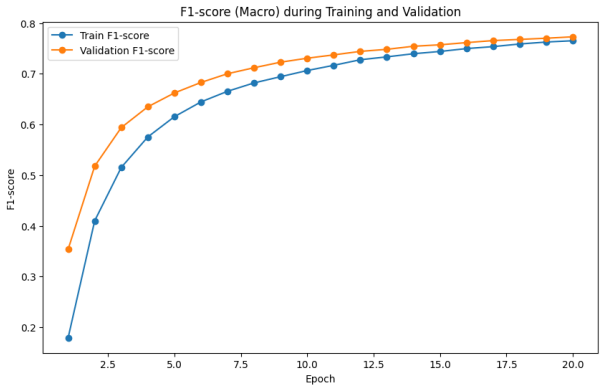

Figure 1: courbe d'apprentissage

d'apprentissage montre que les performances atteignent un palier, indiquant que le modèle a bien convergé. Ainsi, malgré les limitations computationnelles, nous obtenons un modèle assez correct en termes de performances.

Le modèle présente une **accuracy** globale de **77,59 %**, avec des scores de **précision** (78,05 %), **rappel** (77,59 %) et **F1-score** (77,21 %) relativement équilibrés.

Métrique	Précision	Rappel	F1-score
Macro avg	0.78	0.78	0.78
Weighted avg	0.78	0.78	0.77

Table 1: Résumé des performances du modèle

L'analyse des performances par classe montre que certaines catégories, comme les classes naq, gym et asm sont très bien reconnues, avec des F1-scores proches de 1. D'autres classes, comme pes, kaa ou kam, affichent des performances moyennes, suggérant des confusions possibles avec d'autres catégories. Cependant, quelques classes problématiques, notamment hrv, afb ou aym, ont des F1-scores faibles, ce qui indique que le modèle a du mal à bien les distinguer.

Une piste pour améliorer les performances du modèle serait de faire de la data augmentation afin d'équilibrer les classes. En générant de nouvelles données pour les classes sous-représentées, on pourrait aider le modèle à mieux apprendre les caractéristiques de ces classes, réduisant ainsi les déséquilibres qui peuvent nuire à sa capacité de généralisation. Cette approche permettrait de fournir un entraînement plus équilibré et potentiellement d'améliorer les scores des classes moins bien identifiées.