

République Tunisienne

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Carthage

École Nationale d'Ingénieurs de Carthage



Rapport de stage d'été

Présenté par

Firas ABIDLI

Détection du langage inapproprié en arabe

Encadrant professionnel : Mr. Nabil BADRI

Réalisé au sein du Centre de Calcul El Khawarizmi



Année Universitaire 2021-2022

Table des matières

Introduction générale	1
-----------------------	---

1 Présentation générale du projet

1.1 Introduction	4
1.2 Présentation de l'entreprise d'accueil	4
1.3 Cadre du projet	5
1.3.1 Etude de l'existant	5
1.3.2 Problématique	5
1.3.3 Solution proposée	6
1.4 Méthode de gestion de projet	7
1.4.1 CRISP-DM	7
1.5 Conclusion	8

2 ETAT DE L'ART

2.1 Introduction	10
2.2 Traitement de texte	10
2.3 Bert	13
2.4 Bert as Transfer Learning in NLP	18
2.5 Conclusion	20

3 Réalisation et Implémentation

3.1 Introduction	22
3.2 Préparation des données d'entraînement	22
3.3 Environnement et outils de travail.	23
3.4 Réalisation	25

4.5 Conclusion	53
----------------------	----

Conclusion générale	54
---------------------	----

Table des figures

- Figure 1.1 : CRISP-DM
- Figure 2.1 Bert
- Figure 2.2 Fonctionnement de Bert
- Figure 2.3 Bert as Transfer learning
- Figure 2.4 Bert pour Q/R
- Figure 2.5 Bert for image classification
- Figure 3.1 Jeu de données
- Figure 3.2 Téléchargement du data set
- Figure 3.3 Description des données
- Figure 3.4 Labélisation du dataset
- Figure 3.5 Partition des données
- Figure 3.6 Téléchargement du tokeniseur
- Figure 3.7 Fonction Tokenize.encode
- Figure 3.8 BertforsequenceClassification
- Figure 3.9 Encodage des différentes data set
- Figure 3.10 Préparation des data set
- Figure 3.11 Entraînement du modèle
- Figure 3.12 Compute metrics
- Figure 3.13 Evaluation du modèle
- Figure 3.14 Training&Validation loss
- Figure 3.15 Classification report

- Figure 3.16 Matrice de confusion
- Figure 3.17 matrice de confusion pour dialecte marocain
- Figure 3.18 Matrice de confusion pour dialecte egyptien
- Figure 3.19 Matrice de confusion pour dialecte libanais
- Figure 3.20 Matrice de confusion pour Final dataset
- Figure 3.21 Fonction AugmentMyData
- Figure 3.22 Augmentation des données
- Figure 3.23 Repartition du nouveau dataset
- Figure 3.24 Réentraînement du modèle
- Figure 3.25 Validation du nouveau modèle
- Figure 3.27 Matrice de confusion
- Figure 3.28 Matrice de confusion pour dialecte marocain avec nouveau model
- Figure 3.29 Matrice de confusion pour multidialecte avec nouveau model
- Figure 3.30 Presentation du nouveau data
- Figure 3.31 Description de données
- Figure 3.32 Partition de données
- Figure 2.33 Evaluation du nouveau modèle
- Figure 2.34 Classification report
- Figure 2.35 Matrice de confusion
- Figure 2.36 Matrice de confusion du dialecte tunisien avec le nouveau modèle
- Figure 2.37 Matrice de confusion du dialecte marocain avec le nouveau modèle
- Figure 3.38 Matrice de confusion du dialecte marocain avec le nouveau modèle
- Figure 3.39 Matrice de confusion du dialecte libanais avec le nouveau modèle
- Figure 3.40 Sauvegarde du modele
- Figure 3.41 Sauvegarde du modèle dans le Drive

Liste des Tableaux

- Tableau 3.1 Partitions des données
- Tableau 3.2 Paramètres d'entraînement
- Tableau 3.3 Description d'une matrice de confusion
- Tableau 3.4 Métriques de test pour les différents dialectes
- Tableau 3.5 Partition de la nouvelle donnée
- Tableau 3.6 Paramètres d'entraînement
- Tableau 3.7 partitions de jeu de données

Introduction générale

En raison du développement d'Internet au cours des dernières années et du changement d'habitudes et de comportement des gens vis-à-vis de la technologie, les réseaux sociaux ont gagné en popularité et en utilisateurs, générant quotidiennement une quantité impressionnante de commentaires sur n'importe quel sujet. Malheureusement, cela implique également qu'une quantité asignificante de ces commentaires peut contenir des contenus inappropriés tels que des phrases obscènes, agressives, grossières, racistes, sexistes ou violentes. De nos jours, de plus en plus d'enfants ont accès à Internet, il est donc impératif de les empêcher d'accéder à des contenus inappropriés tels que ceux mentionnés précédemment. En ce qui concerne les textes, en raison de la grande quantité de matériel disponible, les examiner tous est une tâche ingérable qui ne peut être accomplie de manière efficace par des inspecteurs humains. Par conséquent, il est nécessaire de développer des solutions automatisées pour filtrer les contenus textuels inappropriés. Les méthodes d'apprentissage automatique appliquées à la classification de texte peuvent être utilisées pour construire des filtres parentaux pour le contenu en ligne et pour détecter les activités illégales telles que les discours de haine.

Dans ce travail, nous avons exploré l'utilisation de certaines techniques de traitement du langage naturel (NLP) et d'apprentissage automatique pour détecter les contenus violents dans les textes.

Le présent rapport résume les étapes de la mise en œuvre de ce projet. Il est structuré en trois chapitres comme suit :

- Le premier chapitre consiste à l'étude préalable du projet, c'est-à-dire son contexte, ses objectifs et la méthode de travail à adopter pour piloter le projet.
- Le second chapitre met en évidence les concepts clés et nécessaires pour implémenter
- Le troisième chapitre présente la réalisation et l'implémentation de notre solution.

Nous finissons ce rapport par une conclusion générale et quelques éventuelles perspectives qui peuvent améliorer notre solution.

Présentation générale du projet

Plan

1	Introduction	4
2	Présentation de l'entreprise d'accueil	4
3	Cadre du projet	4
4	Méthode de gestion de projet	6
5	Conclusion	9

1.1 Introduction

À travers ce chapitre, nous allons présenter le cadre générale du projet. Nous commençons tout d'abord par présenter un aperçu général sur l'entreprise d'accueil. Ensuite, nous allons définir le contexte et le problème ainsi la solution proposée pour le résoudre. À la fin, nous allons présenter la méthodologie adoptée pour le déroulement de notre projet.

1.2 Présentation de l'entreprise d'accueil

Le CCK a été créé en **octobre 1976** suite à l'adoption de l'informatique en tant que spécialité universitaire à la faculté des sciences de Tunis. Il fournissait à l'époque des services de calcul aux établissements, agences et administrations des ministères de l'éducation nationale et de l'enseignement supérieur.

Comme son nom l'indique, le **Centre de Calcul el-Khawarizmi (CCK)** a pour mission d'assurer, d'organiser, de promouvoir, et d'encourager l'utilisation des technologies numériques dans le milieu universitaire et scientifique en général. Le CCK est aussi chargé de la recherche dans ce domaine en vue d'améliorer l'utilisation de la technologie informationnelle numérique dans le milieu universitaire ainsi qu'au profit des enseignants et des étudiants.

La vocation principale du centre vise la promotion de l'activité scientifique et technique académique par la mise en ligne des cours pédagogiques, des sources d'information scientifique et technique et des applications administratives destinées aux étudiants et aux chercheurs.

Riche de son histoire et bénéficiant de son emplacement dans les campus universitaires, le **CCK** s'est orienté vers la diversification de ses activités et l'amélioration de la qualité de ses prestations dans le domaine des **TIC**. En étant principalement un producteur de contenu au service des établissements universitaires tunisiens et en s'associant avec les chercheurs en **TIC** pour offrir un cadre expérimental à la recherche dans ce domaine et œuvrer au transfert des résultats de recherche obtenus.

1.3 Cadre du projet

Dans cette section, nous allons énoncer tout d’abord la problématique. Ensuite l’objectif du projet et nous terminerons par lister les étapes suivies pour atteindre l’objectif final.

1.3.1 Etude de l’existant

Tout comme le taux de pénétration d’Internet, l’utilisation des réseaux sociaux a connu une très forte progression ces dernières années dans l’ensemble des pays arabes. Les réseaux sociaux représentent un espace de contestation, de mobilisation, d’information ainsi qu’un outil d’action, inévitable à la fois pour les citoyens et les gouvernements. Les premiers les utilisent pour diffuser des informations, organiser des manifestations ou sensibiliser l’opinion sur des événements locaux ou mondiaux. Les seconds s’en servent quant à eux pour engager un dialogue avec les citoyens et les encourager à participer aux initiatives gouvernementales. Cela ne les empêche pas de bloquer l’accès à des sites Internet ou surveiller les informations publiées sur le Web, afin d’endiguer certains mouvements de contestation. L’utilisation massive des réseaux sociaux, notamment par les jeunes générations, a clairement permis l’émergence d’un nouvel idéal au sein des sociétés civiles des pays de la région : la « citoyenneté active ». Seul l’avenir pourra déterminer quel sera son impact à long terme sur les modes de gouvernement de la région.

1.3.2 Problématique

Les réseaux sociaux sont de formidables outils pour s’informer et recevoir des commentaires. Cependant, l’abondance de propos irrespectueux, empreints de colère, de méchanceté et de violence qui y sont facilement tenus constitue souvent un frein au militantisme. Il n’est pas normal de se faire insulter, ou de recevoir des menaces de mort ou de viol, peu importe qu’on soit une personne militante, une élue, une personnalité publique ou tout cela à la fois.

Savoir reconnaître le cyber harcèlement et le cyber violence, dénoncer, porter plainte quand c'est possible, utiliser les outils numériques disponibles (par exemple, les options de blocage de commentaires ou de personnes) et sécuriser ses comptes de médias sociaux en modifiant les paramètres, voilà quelques moyens individuels qui peuvent être mis en place pour réduire les dérapages en ligne.

1.3.3 Solution proposée

Pour détecter les discours de haine en ligne, un grand nombre d'études scientifiques a été consacrées en utilisant le traitement du langage naturel (NLP) en combinaison avec des méthodes d'apprentissage machine (ML) et d'apprentissage profond (DL)

Bien que les approches supervisées basées sur l'apprentissage machine aient utilisé différentes caractéristiques basées sur l'exploration de textes, telles que les caractéristiques de surface, l'analyse des sentiments, les ressources lexicales, les caractéristiques linguistiques, les caractéristiques basées sur la connaissance ou les métadonnées basées sur l'utilisateur et la plateforme, elles nécessitent une approche d'extraction de caractéristiques bien définie. La tendance semble maintenant changer de direction, avec des modèles d'apprentissage profond utilisés à la fois pour l'extraction de caractéristiques et l'entraînement de classificateurs. Ces nouveaux modèles appliquent des approches d'apprentissage profond telles que les réseaux neuronaux convolutifs (CNN), les réseaux à mémoire à long terme (LSTM), etc ... pour améliorer les performances des modèles de détection des discours haineux.

Cependant, ils souffrent toujours du manque de données étiquetées ou de l'incapacité à améliorer la propriété de généralisation.

Nous proposons ici une approche d'apprentissage par transfert pour la compréhension des discours haineux. En utilisant une combinaison du modèle non supervisé pré-entraîné BERT et de nouvelles stratégies supervisées de réglage fin [Fine tuning].

En résumé :

Nous proposons une approche d'apprentissage par transfert en utilisant le modèle de langue pré-entraîné BERT appris sur le dialecte arabe pour améliorer la détection de discours haineux sur des ensembles de données de référence publiquement disponibles.

1.4 Méthode de gestion de projet

Chaque projet informatique est guidé par une méthode de développement adéquate pour atteindre ses objectifs d'une manière optimale et efficace.

Dans ce contexte, nous allons présenter la méthodologie que nous avons suivi pour une gestion optimal de notre projet .

1.4.1 CRISP-DM

Cross Industry Standard Process for Data Mining (CRISP - DM) est une méthode de gestion de projet data mining élaboré par IBM. Cette méthode se décompose en 6 étapes ; de compréhension du problème métier au déploiement.

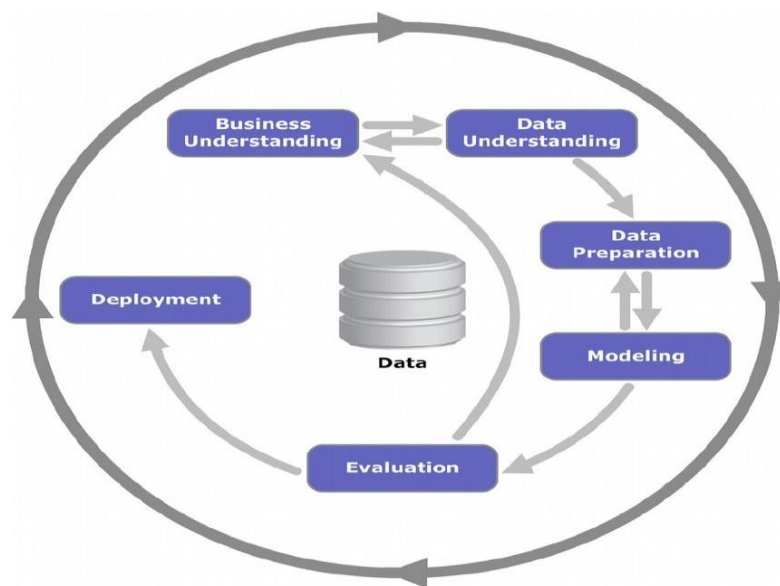


Figure 1.1 : CRISP-DM

- Business understanding : consiste à connaître les spécificités du métier, déterminer les objectifs, évaluer la situation actuelle pour le convertir en un problème de data science.
- Data understanding : cette phase consiste à collecter les données, les explorer afin d'identifier les données pertinentes et aussi vérifier la qualité (les valeurs manquantes etc. ...)
- Data preparation : cette étape comprend le prétraitement et le nettoyage des données ainsi la manipulation des données la transformation des données y compris la standardisation, normalisation etc.
- Modeling : cette phase comprend le choix des algorithmes appliqués aux données préparées (Par exemple réseaux de neurones, SVM, Random forest etc.) ainsi le choix des paramètres.
- Evaluation : cette étape consiste à évaluer la performance du modèle ainsi l'utilité et la fiabilité des résultats.
- déploiement : c'est l'étape finale du processus. En effet lorsque le modèle développé est prêt, il est déployé et intégré dans l'utilisation quotidienne. Ainsi l'objectif de cette étape est la planification de déploiement, surveillance et la maintenance.

1.5 Conclusion

Dans ce chapitre, nous avons tout d'abord présenté notre société d'accueil CCK, ensuite nous avons énoncé notre problématique suivie de la solution proposée pour atteindre nos objectifs. Nous avons également défini notre outil de gestion de projet.

ETAT DE L'ART

Plan

1	Introduction	11
2	Traitement de texte	11
3	Bert	13
4	BERT as Transfer Learning in NLP	17
5	Conclusion	19

2.1 Introduction

La NLP est la nouvelle frontière de la classification des textes. Voyons ce qu'il est et comment l'utiliser.

Le traitement du langage naturel, ou NLP, peut être défini comme un sous-domaine de l'intelligence artificielle. Son objectif est de traiter le langage humain d'une manière similaire à celle dont nous traitons les langues ; il s'agit de comprendre les langues naturelles, c'est-à-dire la manière dont les gens parlent et écrivent dans la vie quotidienne. En conséquence de cet objectif, les chercheurs en NLP essaient de concevoir des algorithmes qui modélisent les processus de compréhension humains lorsqu'ils traitent des données linguistiques.

Étant donné que la première langue écrite enregistrée remonte au sumérien et que la première langue parlée remonte à environ 7 000 ans, les mathématiques et les sciences sont impliquées dans la recherche en NLP depuis le tout début. Et si cette collaboration a toujours été un aspect fondamental de ces recherches, ce n'est que récemment que les méthodes d'intelligence artificielle et de visualisation ont été fusionnées pour faire partie des technologies de base de la NLP. À la suite de ce mariage, l'orientation du domaine s'est progressivement déplacée d'une perspective exclusivement linguistique pour inclure d'autres facteurs tels que la convivialité ou même l'esthétique...

2.2 Traitement de texte

Le traitement de texte est l'une des tâches les plus courantes dans le monde de l'apprentissage automatique comme la traduction, correction automatique, extraction d'information etc...

Comme l'apprentissage automatique ou l'apprentissage profond, Natural Language Processing (NLP) qui signifie en français «Traitement automatique du langage naturel» (TALN) est une branche de l'intelligence artificielle en constante croissance et en développement en regard de ces maints domaines d'application. Mélange linguistique, informatique et IA, le NLP permet aux machines de traiter, comprendre et d'analyser la langue tant écrite que parlée par les humains.

En effet, il s'appuie sur des algorithmes d'analyse syntaxique et sémantique ce qui permet aux machines de concevoir le langage humain c'est à dire comprendre la structure des actes et aussi de le gérer et de le générer autrement dit acquérir la signification d'un mot dans une phrase.

Autrement dit le NLP utilise à la fois des méthodes de machine learning et de deep learning et cette approche lui permet d'assimiler et de traiter efficacement des ensembles de données texte et voix non structurés.

2.2.1 Domaine d'application de NLP

Dans cette section, nous énumérerons quelques exemples pertinents d'applications NLP qui ont fait une différence dans la vie de tous les jours :

- L'intelligence au service de la finance pour l'étude de marché par exemple.
- La traduction en différents langues.
- les chatbots : très efficace pour les interactions et les services clients.
- L'évaluation de la réputation ; analyser les réactions positives et négatives ici on parle du text mining .
- La reconnaissance vocale comme l'assistance virtuelle, l'informatique main libre etc....
- Filtrage des e-mails ; La classification de texte permet de filtrer les courriels indésirables.
- la saisie quasi-automatique dans les moteurs de recherche.
- le processus automatique des cv dans la phase de recrutement.
- Les correcteurs automatiques autrement dit les vérificateurs de grammaire et orthographe.

2.2.2 Les algorithmes NLP

Dans cette section, nous mettrons en place des algorithmes de traitement de texte. Il est vrai qu'il existe de nombreuses techniques NLP, mais nous mettrons l'accent sur les plus populaires sur le marché .

2.2.2.1 Extraction de mots clés

L'extraction de mots clés ou encore la détection des mots-clés est une technique de traitement de texte qui permet l'extraction automatique et intelligente des expressions les fréquents et les plus importants d'un texte. Celui-ci résume le contenu des textes et identifie les principaux thèmes abordés. En effet, l'extraction de mots clés fait appel à l'apprentissage

automatique avec traitement du langage naturel (NLP) pour décomposer le langage humain afin qu'il puisse être compris et analysé par les machines. Il permet de retrouver des mots-clés de toutes sortes de textes par exemple les documents et les rapports commerciaux réguliers, commentaires sur les médias sociaux, forums et journaux en ligne etc...

Parmi les méthodes d'extraction de mot-clé, il existe différentes logiques qui satisfont ce dernier comme Les méthodes statistiques qui sont les plus simples. En effet, elles calculent les statistiques pour les mots clés et les utilisent pour les évaluer. Quelques-unes des méthodes statistiques les plus faciles sont la fréquence des mots, la collocation des mots et la cooccurrence. Cependant, il y a aussi les plus sophistiqué tel que TF-IDF et YAKE. Plus encore, on retrouve les méthodes basées sur des graphiques qui consistent à générer un graphique de termes connexes à partir des documents. Autrement dit le graphique relie les termes qui coexistent dans le texte. Parmi les méthodes graphiques les plus connues, on retrouve le TextRank et RAKE .

2.2.2.2 Reconnaissance d'entité nommée

Reconnaissance d'entité nommée en anglais Named Entity Recognition NER est l'une des méthodes les plus utilisé et commune dans le traitement de text NLP, elle permet d'identifier, localiser et répartir les entités nommées dans un document en catégories prédéfinies tel que les noms, organisations, lieux, dates, adresse, email, numéros de téléphone etc...

Le processus automatique de la reconnaissance d'entité nommée permet d'extraire plus facilement des éléments importants à partir de données textuelles et Cela aide de nombreuses entreprises et organisations à approfondir les connaissances de grands ensembles de données non structurées .

On retrouve l'implémentation de NER dans nombreux application telques l'indexation de documents, les systèmes de questions-réponses, la traduction machine, la classification etc...

2.3 Bert

BERT (Bidirectional Encoder Representations from Transformers) est une impressionnante recherche en traitement du langage naturel (NLP) publiée par des chercheurs de Google AI Language en 2018. Elle a fait sensation dans la communauté de l'apprentissage automatique en présentant des résultats de pointe dans une grande variété de tâches de NLP, notamment la traduction automatique neuronale, la réponse aux questions, la tâche de classification des paires de phrases, l'analyse des sentiments, le résumé de texte et autres. À partir de l'abréviation de BERT, nous pouvons comprendre certaines de ses caractéristiques.

- Il est bidirectionnel
- Il utilise une représentation d'encodeur
- Il a une architecture basée sur un transformateur



Figure 2.1 Bert

Ainsi, l'innovation technique clé de BERT consiste à appliquer la formation bidirectionnelle de Transformer, un modèle d'attention populaire, à la modélisation du langage. Cela contraste avec les efforts précédents qui ont examiné une séquence de texte soit de gauche à droite, soit en combinant l'entraînement de gauche à droite et de droite à gauche. Les résultats de l'article montrent qu'un modèle de langage formé de manière bidirectionnelle peut avoir un sens plus profond du contexte et du flux du langage que les modèles de langage unidirectionnels. Dans l'article, les chercheurs détaillent une nouvelle technique appelée Masked Language Modelling (MLM) qui permet l'entraînement bidirectionnel dans des modèles où cela était auparavant impossible. Nous reviendrons sur tous les détails plus tard dans ce chapitre .

2.3.1 Le monde avant BERT :

Pour comprendre ce qu'est BERT et quelle est l'idée innovante derrière ce modèle nous allons nous baser sur une tâche typique de NLP. Prenons la phrase suivante : “ *La personne va au supermarché et achète une ____ de chaussures.* ” Il est clair qu'ici l'objectif est de compléter cette phrase, la réponse est évidente pour un humain, mais moins pour un algorithme.

Les **modèles pré-BERT** auraient regardé cette séquence de texte de gauche à droite ou bien en combinant de gauche à droite et de droite à gauche. Cette *approche unidirectionnelle* fonctionne bien pour générer des phrases : nous pouvons prédire le mot suivant, l'ajouter à la séquence, puis prédire le mot suivant jusqu'à obtenir une phrase complète. Ici des modèles pré-BERT donneraient environ 70% du temps le mot (paire) et le reste du temps des mots comme “caddie” ou “valise”. Bert lui ne procède pas de cette façon. BERT signifie **Bidirectional Encoder Representations from Transformers**. Comme son nom l'indique ce modèle procède de façon bidirectionnelle, ce qui lui permet d'avoir une bien meilleure compréhension du texte.

2.3.2 La méthode BERT :

Reprenons la même tâche et regardons ce que fait BERT. Au lieu de prédire le mot suivant dans une séquence, BERT utilise une nouvelle technique appelée **Masked LM (MLM)** : il **masque aléatoirement des mots** dans la phrase, puis il essaie de les prédire. Le masquage signifie que le modèle regarde dans les deux sens et qu'il utilise le contexte complet de la phrase, à gauche et à droite, afin de prédire le mot masqué. Contrairement aux modèles de langage précédents, il prend en compte les mots précédents et suivants en même temps. Les modèles existants manquaient cette **approche simultanée** .

2.3.3 Comment ça fonctionne techniquement ? :

Bert est un modèle de type **Transformers**. Un transformer est un modèle qui fonctionne en effectuant un petit nombre constant d'étapes. À chaque étape, il applique un **mécanisme**

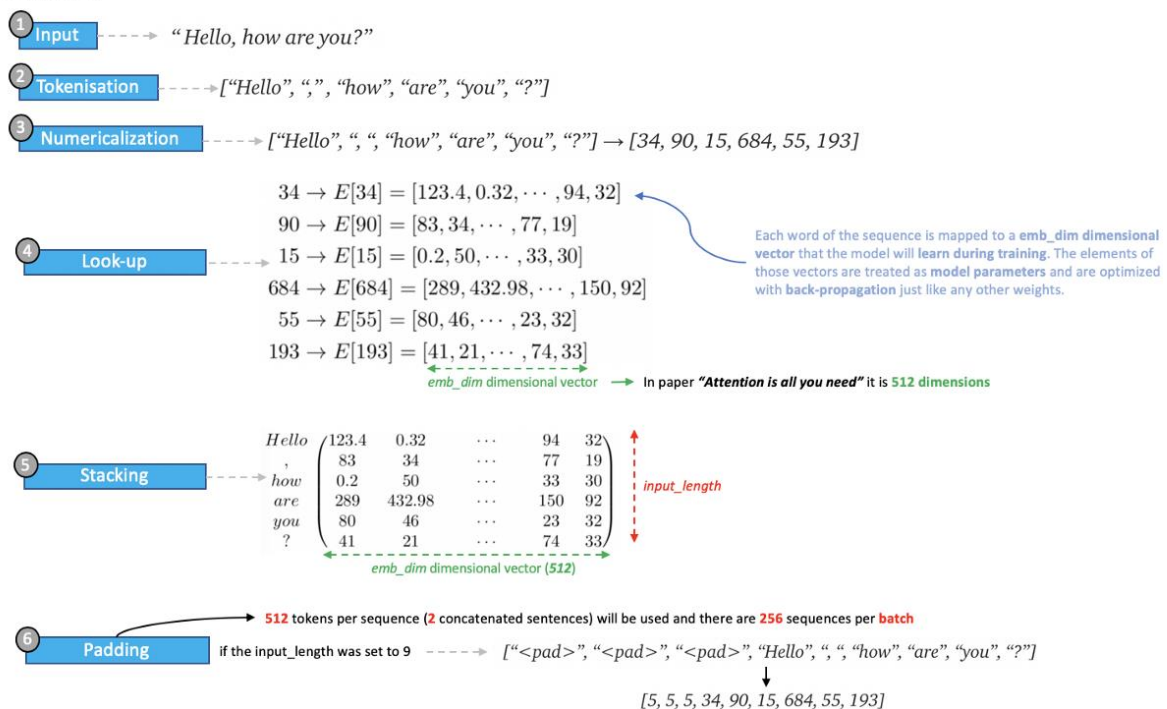
d'attention pour comprendre les relations entre les mots de la phrase, quelles que soient leurs positions respectives.

Prenons un exemple simple : “Tu as une nouvelle souris pour ton ordinateur ?”

Pour déterminer le sens du mot *souris*, l’objet et non l’animal, le transformer va **prêter attention** au mot “ordinateur” et prendre une décision en une étape basée sur ça. Pour permettre cela, BERT se base donc sur l’architecture des transformers, c’est-à-dire consistant en un *encodeur* pour lire le texte et un *décodeur* pour faire une prédiction. BERT se limite à un encodeur, car son objectif est de créer un modèle de représentation du langage qui sera ensuite utilisable pour des tâches de NLP. (Il permet de comprendre le langage).

Nous avons déjà vu l’aperçu de base de BERT. Fondamentalement, BERT est l’empilement d’encodeurs. Maintenant, il est grand temps d’expliquer pourquoi on utilise seulement l’encodeur ? C’est parce que BERT peut être utilisé dans toutes sortes de tâches. En conséquence, ils construisent leur modèle en modèle autorégressif (prédiction du mot suivant). Ainsi, ils ont pu définir une architecture de transformateur unidirectionnel (de gauche à droite). Pour construire une architecture de transformateur bidirectionnel, BERT utilise l’encodeur du transformateur au lieu des décodeurs. Ainsi, BERT construit un modèle non seulement de gauche à droite mais aussi de droite à gauche. Une simple vue d’ensemble de BERT est la suivante :

©AdrienSIEG



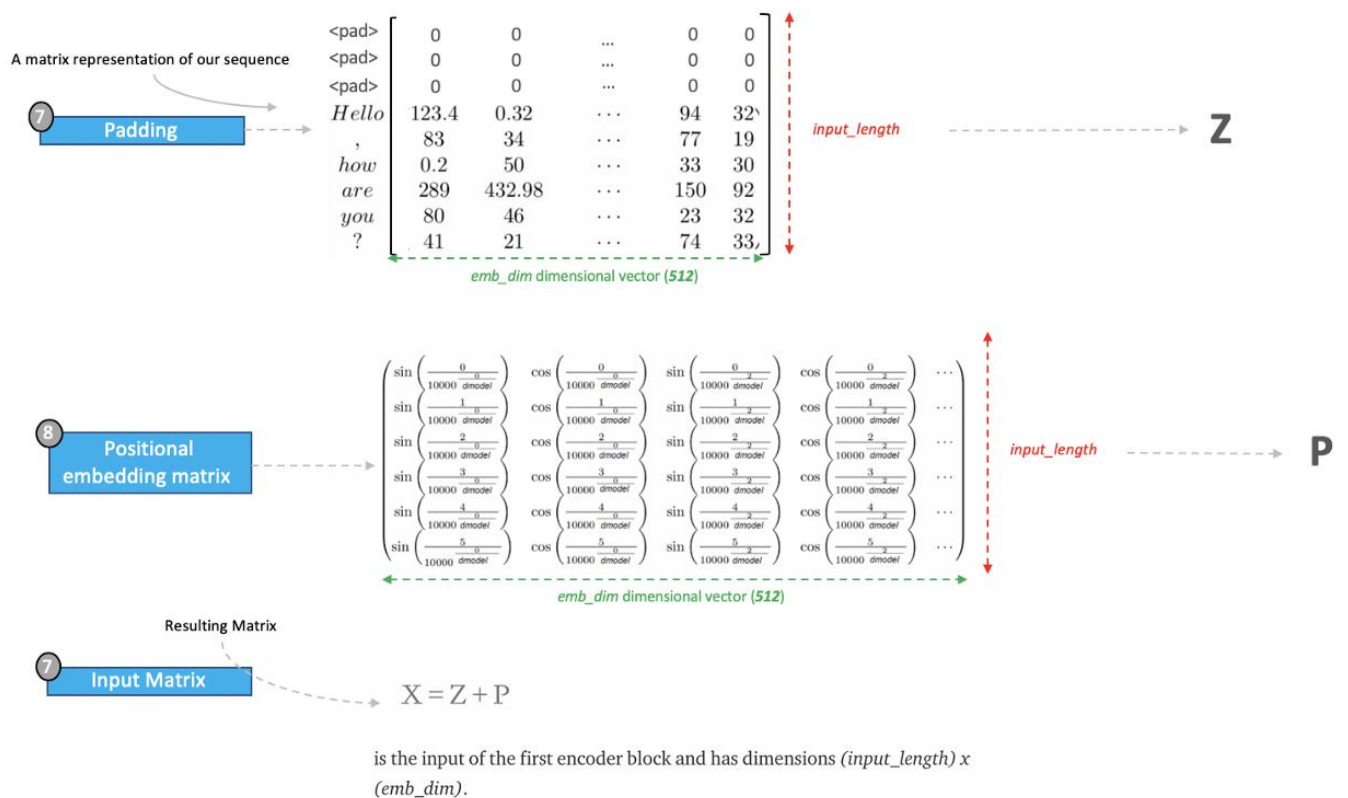


Figure 2.2 Fonctionnement de Bert

- **Tokenization** : consiste à le découper en morceaux, appelés tokens, en éliminant éventuellement certains caractères, comme la ponctuation.
- **Using wordpieces** (par exemple, jouer -> jouer + ##ing) au lieu de mots. Cette méthode permet de réduire la taille du vocabulaire et d'augmenter la quantité de données disponibles pour chaque mot.
- **Numericalization** : vise à mettre en correspondance chaque token avec un entier unique dans le vocabulaire du corpus.
- **Token embedding** : consiste à obtenir l'incorporation (c'est-à-dire un vecteur de nombres réels) pour chaque mot de la séquence. Chaque mot de la séquence est mis en correspondance avec un vecteur dimensionnel emb_dim que le modèle apprendra pendant la formation. Il s'agit en fait d'une recherche de vecteur pour chaque mot. Les éléments de ces vecteurs sont traités comme des paramètres du modèle et sont optimisés par rétropropagation comme tout autre poids.

- **Padding** : est utilisé pour que les séquences d'entrée d'un lot aient la même longueur.

" C'est-à-dire que nous augmentons la longueur de certaines séquences en ajoutant des tokens ".

- **Positional encoding** : Rappelons que le codage positionnel est conçu pour aider le modèle à apprendre une certaine notion des séquences et du positionnement relatif des tokens. Ceci est crucial pour les tâches basées sur le langage, surtout ici, car nous n'utilisons pas d'unités récurrentes traditionnelles telles que RNN, GRU ou LSTM.

Intuitivement, notre objectif est de pouvoir modifier la signification représentée d'un mot spécifique en fonction de sa position. Nous ne voulons pas changer la représentation complète du mot, mais nous voulons la modifier un peu pour encoder sa position en ajoutant des nombres entre $[-1,1]$ en utilisant des fonctions sinusoïdales prédéterminées (non apprises) aux encastresments de jetons. Pour le reste de l'encodeur, le mot sera représenté de manière légèrement différente selon la position dans laquelle il se trouve (même s'il s'agit du même mot).

L'encodeur doit être capable d'utiliser le fait que certains mots sont dans une position donnée alors que, dans la même séquence, d'autres mots sont dans d'autres positions spécifiques. En d'autres termes, nous voulons que le réseau soit capable de comprendre les positions relatives et pas seulement les positions absolues.

- **Sentence embedding** : représentent des phrases entières et leurs informations

sémantiques sous forme de vecteurs. Cela aide la machine à comprendre le contexte, l'intention et d'autres nuances dans le texte entier. Elle remarque simplement quels mots appartiennent à quelles phrases. Un marqueur indiquant la phrase A ou la phrase B est ajouté à chaque jeton. Cela permet au modèle de distinguer les phrases.

2.4 BERT as Transfer Learning in NLP

Comme BERT prend des paires de phrases non étiquetées et que MLM comprend mieux le contexte de la langue, BERT peut être entraîné sur un énorme ensemble de données qui peut être utilisé comme modèle pré-entraîné dans les tâches en aval. La procédure de pré-entraînement suit largement la littérature existante sur le pré-entraînement des modèles de langage. Pour le corpus de pré-entraînement, BERT utilise le BooksCorpus (800 millions de mots) et l'encyclopédie Wikipedia (2 500 millions de mots). Il est essentiel d'utiliser un corpus au niveau du document plutôt qu'un corpus au niveau de la phrase, comme le Billion Word Benchmark, afin d'extraire de longues séquences contiguës.

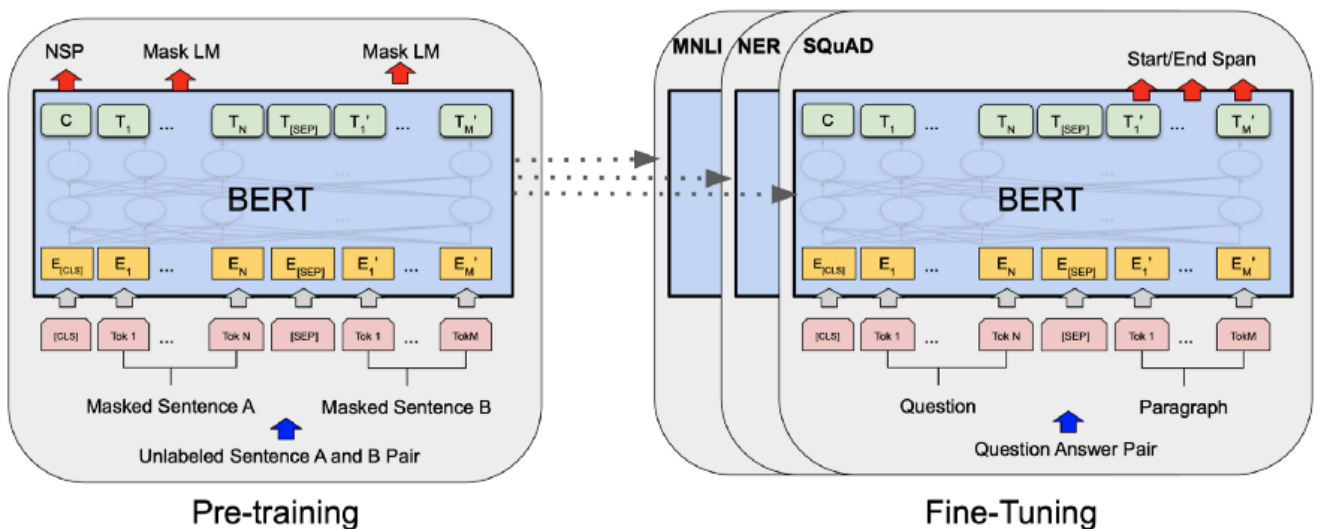


Figure 2.3 Bert as Transfer learning

Les procédures globales de pré-formation et d'ajustement fin pour le BERT sont présentées dans l'image ci-dessus. À l'exception des couches de sortie, les mêmes architectures sont utilisées pour le pré-entraînement et le réglage fin. Les mêmes paramètres de modèle pré-entraînés sont utilisés pour initialiser les modèles pour les différentes tâches en aval. Lors de l'ajustement fin, tous les paramètres sont ajustés. [CLS] est un symbole spécial ajouté devant chaque exemple d'entrée, et [SEP] est un symbole spécial de séparation (par exemple, pour séparer les questions/réponses). Bien que BERT soit entraîné sur des données non supervisées, il peut être affiné pour les tâches supervisées.

Si nous utilisons BERT comme modèle pré-entraîné dans une tâche supervisée en aval (disons une tâche de réponse à des questions), l'image complète sera la suivante :

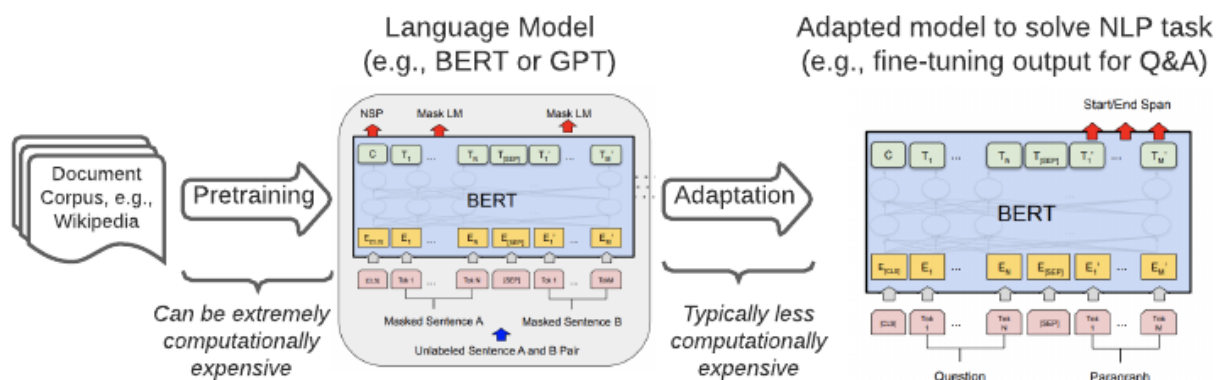


Figure 2.4 Bert pour Q/R

Si nous voulons utiliser BERT pour la classification d'images, le processus d'apprentissage par transfert pour BERT sera le suivant :

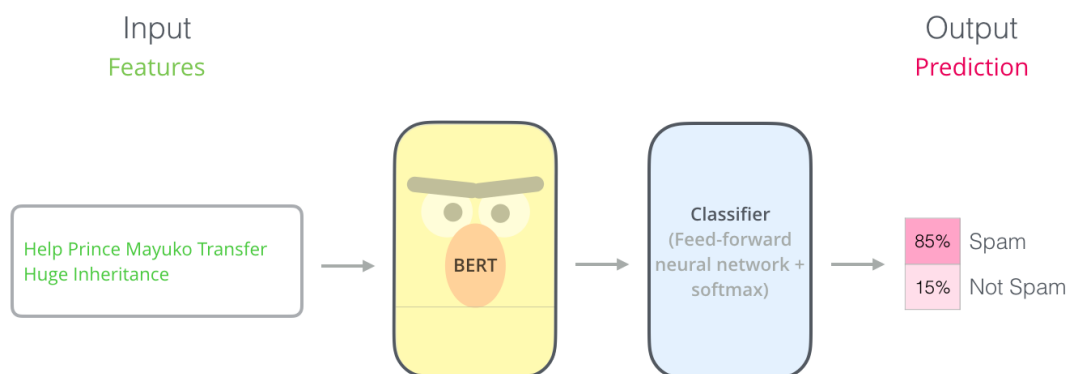


Figure 2.5 Bert for image classification

2.5 Conclusion

Dans ce chapitre, nous avons étudié les différents concepts qu'on va utiliser dans la réalisation de notre projet qui sont le traitement de texte et la modélisation en se basant sur Bert et ajustement fin (fine-tuning). Dans le chapitre suivant, nous allons présenter la partie d'implémentation.

Réalisation et Implémentation

Plan

1	Introduction	28
2	Préparation des données d'entraînement	28
3	Environnement et outils de travail	30
4	Réalisation	31
5	Conclusion	47

3.1 Introduction

Dans ce chapitre, nous allons présenter notre pipeline de modélisation et nous détaillerons étape par étape chaque phase de ce dernier.

3.2 Préparation des données d'entraînement :

La préparation des données en machine Learning consiste à collecter, nettoyer et organiser les data d'apprentissage en vue de les utiliser pour entraîner les modèles d'intelligence artificielle. Le data set d'entraînement est utilisé en amont du processus de machine Learning. C'est une base d'apprentissage (par exemple dans notre cas le data set sera un ensemble varié en dialecte arabe) utilisée pour entraîner le modèle et lui permettre ensuite de réaliser des prédictions sur la base de nouvelles données (soit reconnaître les phrases de contenus violents dans notre exemple).

Vous trouverez ci-dessous les principales étapes :

- L'étape 1 : Il y'a plusieurs méthodes pour collecter des données d'entraînement soit en cherchant sur des sites open source comme Kaggle , Search dataset , Github, Paper with code ou soit avec le webscraping pour collecter les commentaires ou les avis des utilisateurs d'internet .
- L'étape 2 : Après avoir collecté les données, il faut bien évaluer le contenu de données, par exemple, il vaut mieux qu'ils sont variés, en dialecte arabe et portent sur des sujets inappropriés.
- L'étape 3 : C'est l'étape d'étiquetage de texte extrait.
- L'étape 4 : Une fois qu'on a étiqueté les morceaux de texte, on effectue la phase de prétraitement de texte qui consiste à nettoyer le texte.
- L'étape 5 : stocker les data set des différents dialectes arabe pour les utiliser plus tard dans la phase de modélisation.

3.3 Environnement et outils de travail

Dans cette section, nous illustrons les environnements et les technologies de travail sélectionnés pour mettre en œuvre notre système.

4.3.1 Environnement de travail

4.3.1.1 Jupyter

Jupyter est un environnement web de développement interactif pour créer et partager des documents informatiques et la visualisation des données. Son interface adaptable offre aux utilisateurs la possibilité de configurer et d'organiser des flux de travail dans les domaines de la science des données, l'analyse de données et de l'apprentissage automatique.

4.3.3 Packages et outils

Plusieurs paquets ont été utilisés dans notre travail, nous en énumérons quelques-uns ci-dessous :

4.3.3.2 Pandas

Pandas est une bibliothèque open source qui fournit des structures de données et des outils d'analyse de données performants en langage de programmation Python et ils sont faciles à utiliser et à exploiter.

4.3.3.3 Numpy

NumPy est une bibliothèque pour langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.

4.3.3.3 Scikit-learn

Scikit-learn est une bibliothèque libre Python destinée à l'apprentissage automatique. Elle est développée par de nombreux contributeurs notamment dans le monde académique par des instituts français d'enseignement supérieur et de recherche comme Inria

4.3.3.4 PyTorch-Transformers

PyTorch-Transformers est une bibliothèque de modèles pré-formés de pointe pour le traitement du langage naturel (NLP).

J'ai pris cette section de la documentation de PyTorch-Transformers. Cette bibliothèque contient actuellement des implémentations PyTorch, des poids de modèle pré-entraînés, des scripts d'utilisation et des utilitaires de conversion pour les modèles suivants :

1. **BERT (de Google)** publié avec l'article BERT : Pré-formation des transformateurs bidirectionnels profonds pour la compréhension du langage
2. **GPT (d'OpenAI)** publié avec l'article Improving Language Understanding by Generative Pre-Training
3. **GPT-2 (d'OpenAI)** publié avec l'article Les modèles de langage sont des apprenants multitâches non supervisés
4. **Transformer-XL (de Google/CMU)** publié avec l'article Transformer-XL : Attentive Language Models Beyond a Fixed-Length Context
5. **XLNet (de Google/CMU)** publié avec l'article XLNet : Generalized Autoregressive Pretraining for Language Understanding
6. **XLNet (de Facebook)** publié avec le document Cross-lingual Language Model Pretraining

Tous les modèles ci-dessus sont les meilleurs de leur catégorie pour diverses tâches de NLP.

La plupart des modèles de pointe nécessitent des tonnes de données de formation et des jours de formation sur du matériel GPU coûteux, ce que seules les grandes entreprises technologiques et les laboratoires de recherche peuvent se permettre. Mais avec le lancement de PyTorch-

Transformers, tout le monde peut désormais utiliser la puissance des modèles à la pointe de la technologie.

4.3.3.5 *Nlpgaug*

Nlpgaug est une bibliothèque pour l'augmentation textuelle dans les expériences d'apprentissage automatique. L'objectif est d'améliorer les performances du modèle d'apprentissage en profondeur en générant des données textuelles. Il est également capable de générer des exemples contradictoires pour empêcher les attaques contradictoires.

4.3.3.6 Matplotlib

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy

3.4 Réalisation

Dans cette partie, On va détailler chaque étape du workflow générale accompagnées par une explication de chaque outil utilisé afin de subdevenir à notre résultat final qui est la détection de langage inapproprié

3.4.1 Ajustement fin pour Bert

Nous allons utiliser BERT pour entraîner un classificateur de texte. Plus précisément, nous allons prendre le modèle BERT pré-entraîné, ajouter une couche de neurones non entraînée à la fin, et entraîner le nouveau modèle pour notre tâche de classification. Pourquoi faire cela plutôt que de former un modèle d'apprentissage profond spécifique (un CNN, un BiLSTM, etc.) qui est bien adapté à la tâche NLP spécifique dont vous avez besoin ?

- Un développement plus rapide :

Premièrement, les poids du modèle BERT pré-entraîné encodent déjà beaucoup d'informations sur notre langue. C'est comme si nous avions déjà entraîné les couches inférieures de notre réseau de manière intensive et que nous devions seulement les ajuster doucement tout en utilisant leur sortie comme caractéristiques pour notre tâche de classification. En fait, les auteurs recommandent seulement 2-4 époques d'entraînement pour affiner BERT sur une tâche NLP spécifique (comparé aux centaines d'heures GPU nécessaires pour entraîner le modèle BERT original ou un LSTM à partir de zéro !)

- Moins de données :

De plus, et c'est peut-être tout aussi important, grâce aux poids pré-entraînés, cette méthode nous permet d'affiner notre tâche sur un ensemble de données beaucoup plus petit que celui qui serait nécessaire pour un modèle construit à partir de zéro. Un inconvénient majeur des modèles NLP construits à partir de zéro est que nous avons souvent besoin d'un ensemble de données prohibitif afin d'entraîner notre réseau avec une précision raisonnable, ce qui signifie que beaucoup de temps et d'énergie ont dû être consacré à la création de l'ensemble de données. En affinant BERT, nous sommes maintenant en mesure d'entraîner un modèle avec de bonnes performances sur une quantité beaucoup plus petite de données d'entraînement.

- De meilleurs résultats :

Enfin, cette simple procédure de réglage fin (qui consiste à ajouter une couche entièrement connectée à BERT et à s'entraîner pendant quelques époques) a permis d'obtenir des résultats de pointe avec des ajustements minimaux pour une grande variété de tâches : classification, inférence linguistique, similarité sémantique, réponse à des questions, etc. Plutôt que de mettre en œuvre des architectures personnalisées et parfois obscures qui se sont avérées efficaces pour une tâche spécifique, le simple réglage fin de BERT s'avère être une meilleure alternative (ou au moins égale).

3.4.2 Implémentation :

3.4.2.1 Description et préparation du data set :

Nous avons collecter notre données d'entraînement du différentes site open source comme Kaggle , GitHub et principalement à l'aide du dataset search de google , les dialectes choisis sont : Tunisien , Egyptien , Libanais et marocain .

dataset_egyptien	22/05/2022 13:37	Feuille de calcul M...	87 Ko
dataset_libanais	22/05/2022 13:37	Feuille de calcul M...	521 Ko
dataset_marocain_balanced	18/07/2022 13:12	Feuille de calcul M...	1 213 Ko
dataset_marrocin	26/07/2022 10:33	Feuille de calcul M...	318 Ko
dataset_tunisien	22/05/2022 13:37	Feuille de calcul M...	457 Ko
final_multidialecte_imbalanced	26/07/2022 10:28	Feuille de calcul M...	1 647 Ko

Figure 3.1 Jeu de données

Dans ce qui suit nous allons monter un brève exemple du data set tunisien ainsi que la phase de labélisation des classes.

```
[ ] from google.colab import drive
drive.mount("/content/drive/")

[ ] from google.colab import files
import io
uploaded =files.upload()

Sélect. fichiers Aucun fichier choisi Upload widget is only available when the cell has been
Saving dataset1and2.xlsx to dataset1and2.xlsx

[ ] df = pd.read_excel(io.BytesIO(uploaded['dataset1and2.xlsx']))
df.head()
```

	Unnamed: 0	commentaire	classe
0	0	اسغي ياشعب تونس تدعوا بالاسلام كفار الحمد لله ن	hate
1	1	قطع يد السارق توفر شرط الحد الأدنى قيم	normal
2	2	تلموش لطفي لعبدلي شرف	normal
3	3	مستغرب شعب يسمع نقاهة شانو لي الدرجة الشعب تاف	normal
4	4	ههههه عزلتني مافهمتش شمدخلها الموضوع تنتظر وحده	normal

Figure 3.2 Téléchargement du data set

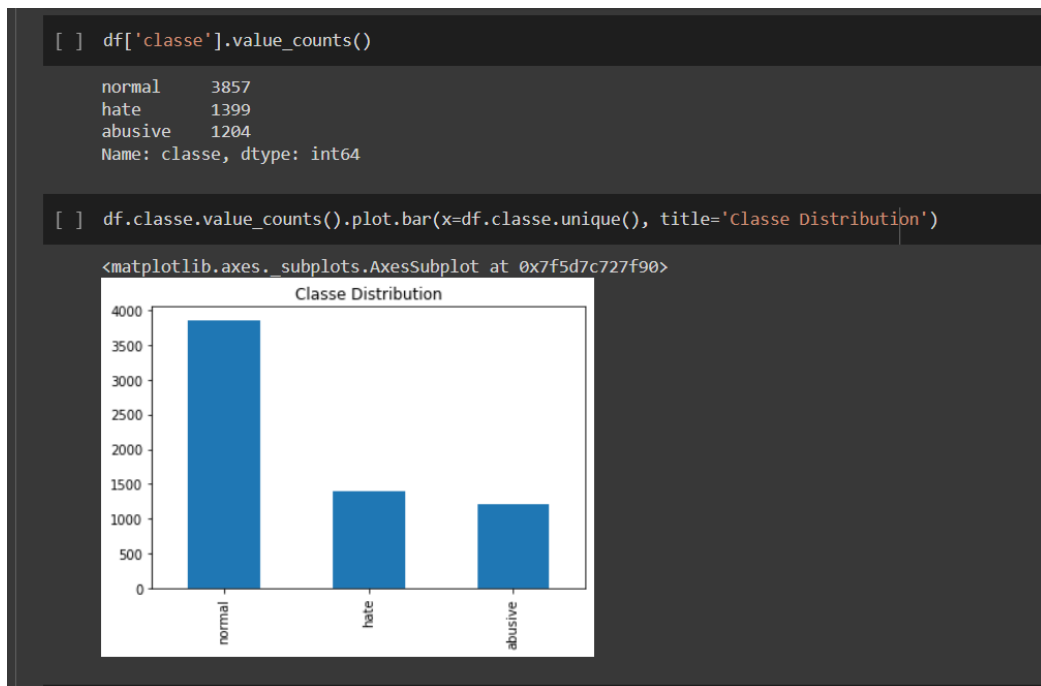


Figure 3.3 Description des données

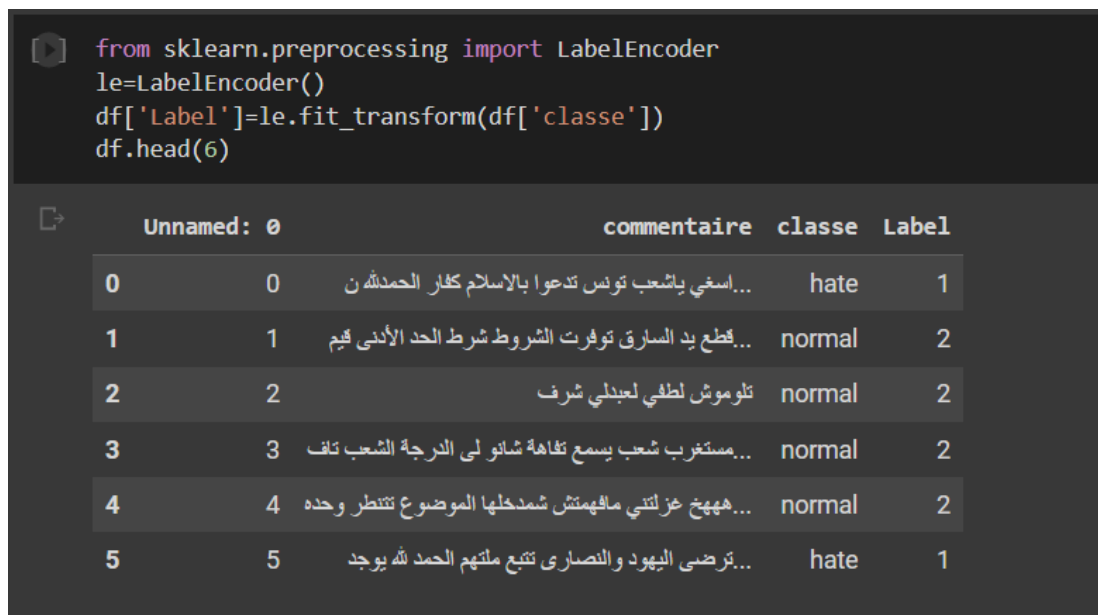


Figure 3.4 Labélisation du dataset

Dans la phase de nettoyage de données (preprocessing), nous avons utilisé les bibliothèques python Pyarabic et Farasapy :

Pyarabic : Une bibliothèque spécifique à la langue arabe pour Python, qui fournit des fonctions de base pour manipuler les lettres arabes et le texte, comme la détection des lettres arabes, les groupes de lettres arabes et leurs caractéristiques, la suppression des diacritiques, etc.

Farasapy : Farasapy est une bibliothèque Python typiquement utilisée dans les applications d'intelligence artificielle et de traitement du langage naturel. Farasapy n'a pas de bogues, elle n'a pas de vulnérabilités, elle a un fichier de construction disponible, elle a une licence permissive et elle a un faible soutien. Elle est généralement utilisée pour les tâches suivantes : Segmentation, dédoublement, reconnaissance d'entités nommées (NER), étiquetage POS (Part Of Speech tagging) , diacritisation.

Partition du data set :

```
[ ] train_texts, temp_texts, train_labels, temp_labels = train_test_split(df['commentaire'], df['Label'], random_state=42,
                                                                    test_size=0.3)

train_texts=train_texts.apply(preprocess)
temp_texts=temp_texts.apply(preprocess)

val_texts, test_texts, val_labels, test_labels = train_test_split(temp_texts, temp_labels, random_state=42,
                                                                    test_size=0.5)

len(train_texts), len(val_texts), len(test_texts)

(4522, 969, 969)
```

Figure 3.5 Partition des données

Train Data	70% ⇔ 4522
Validation Data	15% ⇔ 969
Test Data	15% ⇔ 969

Total Data	100% ⇔ 6490
------------	-------------

Tableau 3.1 Partitions des données

3.4.2.2 Expérimentation et évaluation

- Expérimentation

Nous avons choisi trois modèle pré-entraînée pour l'ajustement fin : DziriBert , Arabertv2 , Bert base arabic .

Dans cette partie nous allons focaliser sur l'ajustement fin du modèle **DziriBert** avec le data set **du dialecte tunisien**.

NB : Ajustement fin des deux autres modèles est identique à celui-ci avec ce modèle.

Pour alimenter notre texte à BERT, il doit être divisé en tokens, puis ces tokens doivent être mis en correspondance avec leur index dans le vocabulaire du tokenizer. La tokénisation doit être effectuée par le tokéniseur inclus dans BERT - la cellule ci-dessous le téléchargera pour nous.

```

- DziriBert

[ ] BERT_MODEL_NAME = 'alger-ia/dziribert'
bert = AutoModel.from_pretrained(BERT_MODEL_NAME)

tokenizer_dziribert = BertTokenizerFast.from_pretrained(BERT_MODEL_NAME)

Downloading config.json: 100% ██████████ 620/620 [00:00<00:00, 11.9kB/s]
Downloading pytorch_model.bin: 100% ██████████ 475M/475M [00:15<00:00, 30.8MB/s]
Some weights of the model checkpoint at alger-ia/dziribert were not used when initializing BertModel: ['cls.predictions.tr
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with anothe
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly iden
Some weights of BertModel were not initialized from the model checkpoint at alger-ia/dziribert and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Downloading tokenizer_config.json: 100% ██████████ 176/176 [00:00<00:00, 2.46kB/s]
Downloading vocab.txt: 100% ██████████ 436k/436k [00:00<00:00, 8.49kB/s]

```

Figure 3.6 Téléchargement du tokeniseur

Lorsque nous convertirons réellement toutes nos phrases, nous utiliserons la fonction `tokenizer.encode` pour gérer les deux étapes, plutôt que d'appeler `tokenizer` et `convert_tokens_to_ids` séparément.

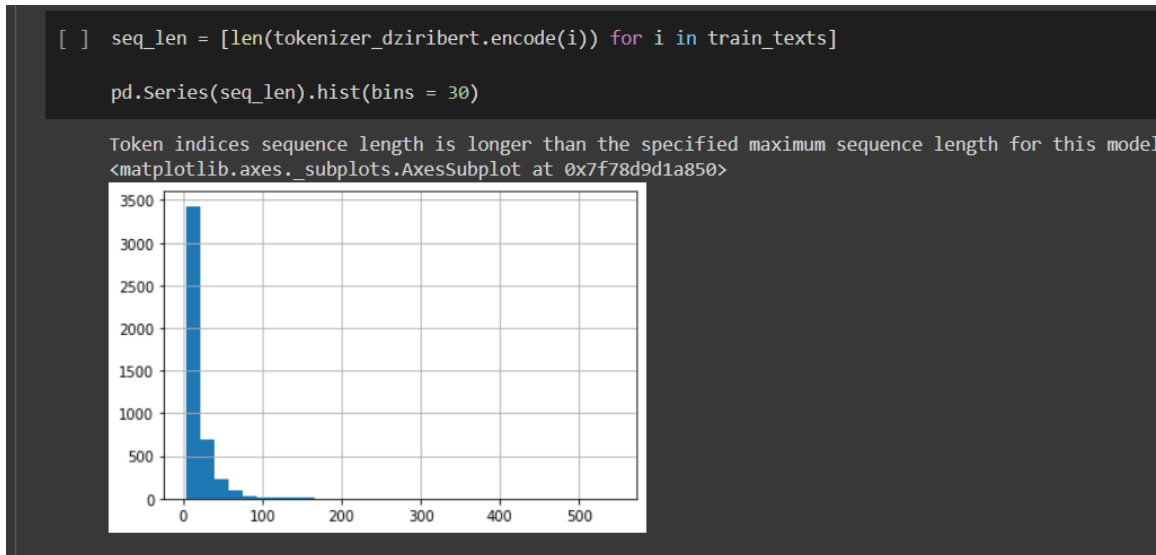


Figure 3.7 Fonction `Tokenizer.encode`

`BertForSequenceClassification` :

Pour cette tâche, nous voulons d'abord modifier le modèle BERT pré-entraîné pour obtenir des sorties pour la classification, puis nous voulons continuer à entraîner le modèle sur notre jeu de données jusqu'à ce que le modèle entier, de bout en bout, soit bien adapté à notre tâche. Heureusement, l'implémentation de `huggingface pytorch` comprend un ensemble d'interfaces conçues pour une variété de tâches NLP. Bien que ces interfaces soient toutes construites à partir d'un modèle BERT entraîné, chacune d'entre elles possède des couches supérieures et des types de sortie différents conçus pour s'adapter à leur tâche NLP spécifique.

Nous utiliserons `BertForSequenceClassification`. Il s'agit du modèle BERT normal avec une couche linéaire unique ajoutée au sommet pour la classification que nous utiliserons comme classificateur de phrases. Lorsque nous alimentons les données d'entrée, l'ensemble du modèle BERT pré-entraîné et la couche de classification supplémentaire non entraînée sont entraînés à notre tâche spécifique.

```
model = AutoModelForSequenceClassification.from_pretrained(BERT_MODEL_NAME, num_labels=3)
```

Figure 3.8 BertforsequenceClassification

Les phrases de notre ensemble de données ont évidemment des longueurs variables, alors comment BERT gère-t-il cela ?

BERT a deux contraintes : Toutes les phrases doivent être complétées ou tronquées à une longueur unique et fixe. La longueur maximale des phrases est de 512 tokens .

```
train_encodings = tokenizer_dziriBERT(train_texts.to_list(), truncation=True, padding=True, max_length=max_seq_len)
val_encodings = tokenizer_dziriBERT(val_texts.to_list(), truncation=True, padding=True, max_length=max_seq_len)
test_encodings = tokenizer_dziriBERT(test_texts.to_list(), truncation=True, padding=True, max_length=max_seq_len)
```

Figure 3.9 Encodage des différentes data set

```
[ ] class preparerDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels.to_list()

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = preparerDataset(train_encodings, train_labels)
val_dataset = preparerDataset(val_encodings, val_labels)
test_dataset = preparerDataset(test_encodings, test_labels)
```

Figure 3.10 Préparation des data set

Finalement la configuration des arguments d'entraînement du modèle DziriBERT et le lancement d'ajustement fin pour ce dernier.

Hyperparameter	Options
N° epochs	5
Batch size per device during training	16
Batch size per device during evaluation	32
Warmup_steps	500
Weight decay	0.01

Tableau 3.2 Paramètres d'entraînement

```
training_args = TrainingArguments(  
    output_dir='./results',          # output directory  
    num_train_epochs=5,              # total number of training epochs  
    per_device_train_batch_size=16,  # batch size per device during training  
    per_device_eval_batch_size=32,   # batch size for evaluation  
    warmup_steps=500,                # number of warmup steps for learning rate scheduler  
    weight_decay=0.01,               # strength of weight decay  
    logging_strategy='epoch',  
    evaluation_strategy='epoch'  
)  
  
dziribert = Trainer(  
    model=model,                     # the instantiated Transformers model to be trained  
    args=training_args,              # training arguments, defined above  
    train_dataset=train_dataset,     # training dataset  
    eval_dataset=val_dataset,        # evaluation dataset  
    compute_metrics=compute_metrics  
)  
  
dziribert.train()
```

Figure 3.11 Entraînement du modèle

- Evaluation :

Dans cette section, nous explorons les possibilités et les limites des différentes approches ML que nous avons évaluées. Pour ce faire, nous avons utilisé les mesures habituelles dans les tâches NLP, y compris precision, recall, F1 score et accuracy.

Pour bien comprendre ces mesures nous vous invitons à jeter un œil sur la description suivante.

Actual Class	Predicted class		
		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Tableau 3.3 Description d'une matrice de confusion

Les vrais positifs et les vrais négatifs sont les observations correctement prédites et donc affichées en vert. Nous voulons minimiser les faux positifs et les faux négatifs afin qu'ils soient affichés en rouge. Ces termes prêtent un peu à confusion. Prenons donc chaque terme un par un et comprenons-le pleinement.

Vrais positifs (TP) - Ce sont les valeurs positives correctement prédites, ce qui signifie que la valeur de la classe réelle est oui et que la valeur de la classe prédite est également oui. Par exemple, si la valeur de classe réelle indique que ce passager a survécu et que la classe prévue vous dit la même chose.

Vrais négatifs (TN) - Ce sont les valeurs négatives correctement prédites, ce qui signifie que la valeur de la classe réelle est non et que la valeur de la classe prédite est également non. Par exemple, si la classe réelle indique que ce passager n'a pas survécu et que la classe prévue vous dit la même chose.

Faux positifs et faux négatifs, ces valeurs se produisent lorsque votre classe réelle est en contradiction avec la classe prédite.

Faux positifs (FP) - Lorsque la classe réelle est non et la classe prédite est oui. Par exemple, si la classe réelle indique que ce passager n'a pas survécu, mais que la classe prévue vous indique que ce passager survivra.

Faux négatifs (FN) - Lorsque la classe réelle est oui mais que la classe prédite est non. Par exemple, si la valeur de classe réelle indique que ce passager a survécu et que la classe prédite vous indique que le passager va mourir.

Une fois que vous avez compris ces quatre paramètres, nous pouvons calculer l'exactitude, la précision, le rappel et le score F1.

Accuracy – c'est la mesure de performance la plus intuitive et il s'agit simplement d'un rapport entre l'observation correctement prédite et le nombre total d'observations. On peut penser que, si nous avons une grande précision, notre modèle est le meilleur. Oui, la précision est une excellente mesure, mais uniquement lorsque vous avez des ensembles de données symétriques où les valeurs des faux positifs et des faux négatifs sont presque identiques. Par conséquent, vous devez examiner d'autres paramètres pour évaluer les performances de votre modèle. Pour notre modèle, nous avons 0,803, ce qui signifie que notre modèle est d'env. 80% précis.

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

Précision - La précision est le rapport entre les observations positives correctement prédites et le nombre total d'observations positives prédites. La question à laquelle cette réponse métrique est de tous les passagers étiquetés comme ayant survécu, combien ont réellement survécu ? La haute précision est liée au faible taux de faux positifs. Nous avons une précision de 0,788, ce qui est plutôt bon.

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

Recall (sensibilité) - Le rappel est le rapport entre les observations positives correctement prédites et toutes les observations de la classe réelle - oui. La question à laquelle le rappel répond est : de tous les passagers qui ont vraiment survécu, combien en avons-nous étiquetés ? Nous avons un rappel de 0,631, ce qui est bon pour ce modèle car il est supérieur à 0,5.

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

Score F1 - Le score F1 est la moyenne pondérée de la précision et du rappel. Par conséquent, ce score prend en compte à la fois les faux positifs et les faux négatifs. Intuitivement, ce n'est pas aussi facile à comprendre que la précision, mais F1 est généralement plus utile que la précision, surtout si vous avez une distribution de classe inégale. La précision fonctionne mieux si les faux positifs et les faux négatifs ont un coût similaire. Si le coût des faux positifs et des faux négatifs est très différent, il est préférable de regarder à la fois la précision et le rappel. Dans notre cas, le score F1 est de 0,701.

$$\text{Score F1} = 2 * (\text{Recall} * \text{Précision}) / (\text{Recall} + \text{Précision})$$

Ci-dessous la configuration pour que nous puissions visualiser la partie de validation du modèle :

```
[ ] from sklearn.metrics import precision_recall_fscore_support, accuracy_score

# function to pass to out trainer in order to compute accuracy, f1 score, precision and recall
def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='macro')
    acc = accuracy_score(labels, preds)
    return {
        'accuracy': acc,
        'f1': f1,
        'precision': precision,
        'recall': recall
    }
```

Figure 3.12 Compute metrics

```

Instantaneous batch size per device = 16
Total train batch size (w. parallel, distributed & accumulation) = 16
Gradient Accumulation steps = 1
Total optimization steps = 1415
[1415/1415 36:52, Epoch 5/5]

Epoch Training Loss Validation Loss Accuracy F1 Precision Recall
1 0.781100 0.618913 0.726522 0.646182 0.680288 0.671942
2 0.494100 0.527565 0.808050 0.759007 0.762105 0.756460
3 0.221100 0.760149 0.786378 0.732281 0.736860 0.730812
4 0.072300 1.108537 0.791538 0.745949 0.746145 0.752648
5 0.015500 1.199071 0.803922 0.758088 0.756531 0.759791

**** Running Evaluation ****
Num examples = 969
Batch size = 32
Saving model checkpoint to ./results/checkpoint-500
Configuration saved in ./results/checkpoint-500/config.json
Model weights saved in ./results/checkpoint-500/pytorch_model.bin
**** Running Evaluation ****
Num examples = 969
Batch size = 32
**** Running Evaluation ****

```

Figure 3.13 Evaluation du modèle

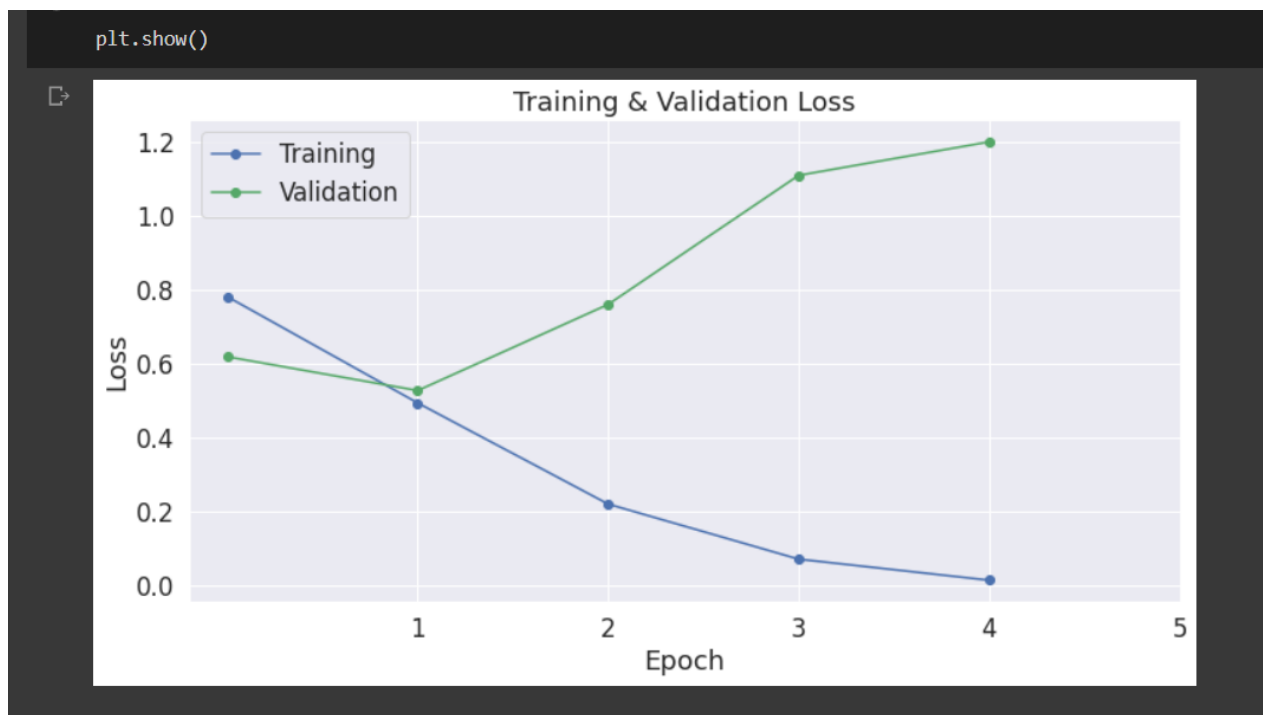


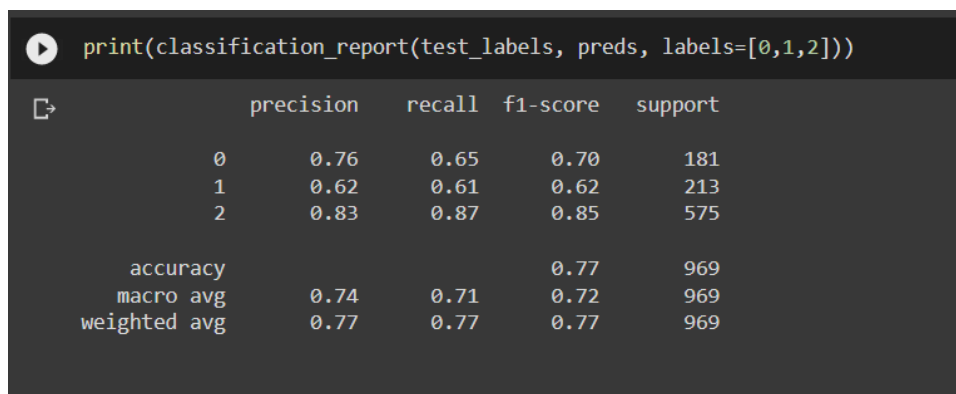
Figure 3.14 Training&Validation loss

Remarquez que, alors que la perte d'apprentissage diminue à chaque époque, la perte de validation augmente ! Cela suggère que nous entraînons notre modèle trop longtemps et qu'il s'adapte trop aux données d'entraînement.

(Pour référence, nous utilisons 4522 échantillons d'entraînement et 969 échantillons de validation).

La perte de validation est une mesure plus précise que la précision, parce qu'avec la précision, nous ne nous soucions pas de la valeur de sortie exacte, mais simplement de quel côté d'un seuil elle se situe. Si nous prédisons la bonne réponse, mais avec moins de confiance, alors la perte de validation en tiendra compte, alors que la précision ne le fera pas.

❖ Classification report



```
print(classification_report(test_labels, preds, labels=[0,1,2]))
```

	precision	recall	f1-score	support
0	0.76	0.65	0.70	181
1	0.62	0.61	0.62	213
2	0.83	0.87	0.85	575
accuracy			0.77	969
macro avg	0.74	0.71	0.72	969
weighted avg	0.77	0.77	0.77	969

Figure 3.15 Classification report

❖ Matrice de confusion

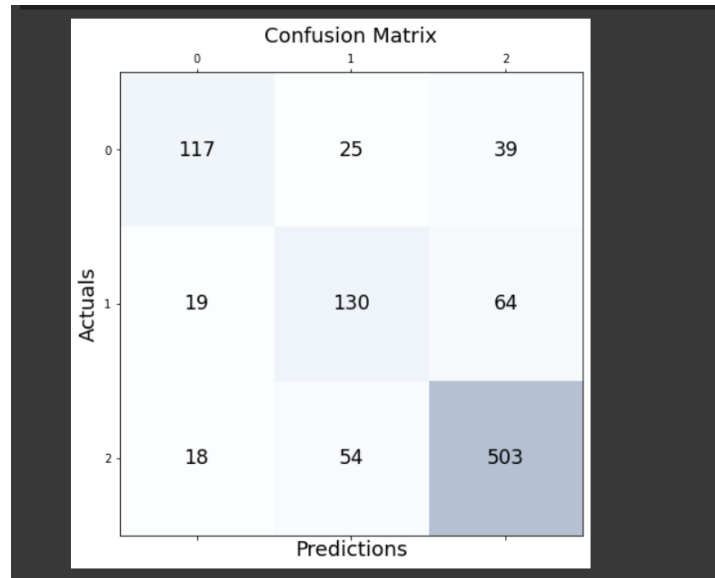


Figure 3.16 Matrice de confusion

❖ Test du modèle avec les différents dialectes :

Dans cette section nous allons tester le modèle qui est préalablement ajusté en dialecte tunisien avec les autres dialectes en traçant la matrice de confusion pour chacun.

- Test avec dialecte marocain :

Pour une partie test de 583 exemples :

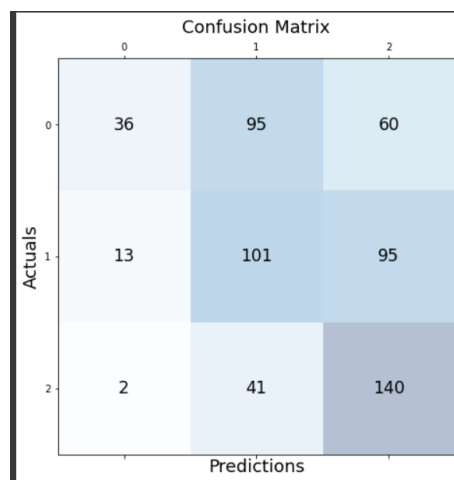


Figure 3.17 matrice de confusion pour dialecte marocain

- Test avec dialecte égyptien :

Pour une partie de test de 165 exemples :

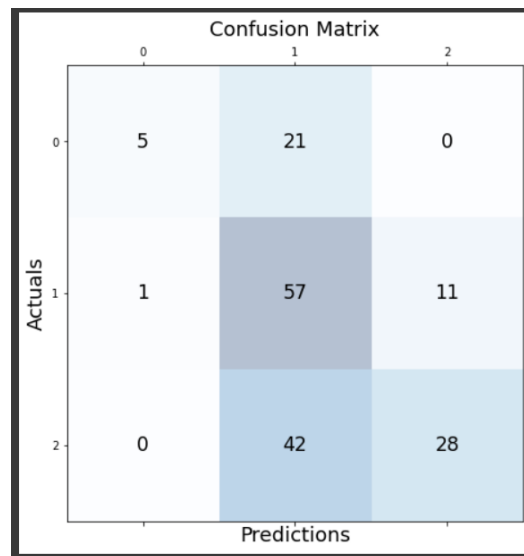


Figure 3.18 Matrice de confusion pour dialecte égyptien

- Test avec dialecte libanais :

Pour une partie de test de 1380 exemples :

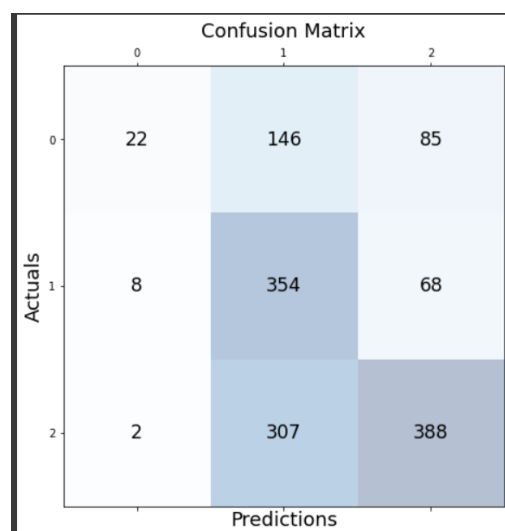


Figure 3.19 Matrice de confusion pour dialecte libanais

- Test avec la data set final (contien la concatenation de toutes les datasets precedement)

Pour une partie de test de 3219 exemples :

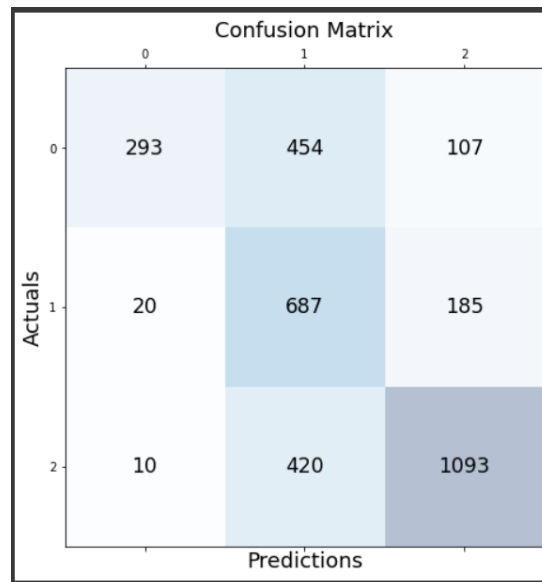


Figure 3.20 Matrice de confusion pour Final dataset

Remarque : Nous constatons d'après les matrices de confusions récentes que notre modèle souffre d'une mauvaise précision. Cela nous prouve qu'il s'adapte trop aux données d'entraînement. Le Tableau ci-dessous montre aussi la même conclusion.

Data set	Test_precision	Test_recall	Test_accuracy	Test_F1
Data set marocain	0.57	0.46	0.46	0.42
Data set égyptien	0.67	0.44	0.52	0.43

Data set libanais	0.59	0.40	0.49	0.45
Data set multidialecte	0.72	0.60	0.61	0.59

Tableau 3.4 Métriques de test pour les différents dialectes

Pour améliorer notre modèle, nous allons en premier lieu d'augmenter notre jeu de donnée et après nous le allons changer pour qu'il soit diversifié en plus

❖ Augmentation des données et l'amélioration du modèle :

L'augmentation des données est le processus d'augmentation de la quantité et de la diversité des données. Nous ne collectons pas de nouvelles données, nous transformons plutôt les données déjà présentes.

```
def augmentMyData(df, augementer, repetitions=1, sample0=200, sample1=200):
    for j in [0,1] :
        if j==0 :
            samples=sample0
        else :
            samples=sample1
        augmented_texts = []
        # select only the minority class samples
        spam_df = df[df['Label'] == j].reset_index(drop=True) # removes unnecessary index column
        for i in tqdm(np.random.randint(0, len(spam_df), samples)):
            # generating 'n_samples' augmented texts
            for _ in range(repetitions):
                augmented_text = augementer.augment(spam_df['commentaire'].iloc[i])
                augmented_texts.append(augmented_text)

        data = {
            'Label': j,
            'commentaire': augmented_texts
        }
        aug_df = pd.DataFrame(data)
        df = shuffle(df.append(aug_df).reset_index(drop=True))
    return df
```

Figure 3.21 Fonction AugmentMyData

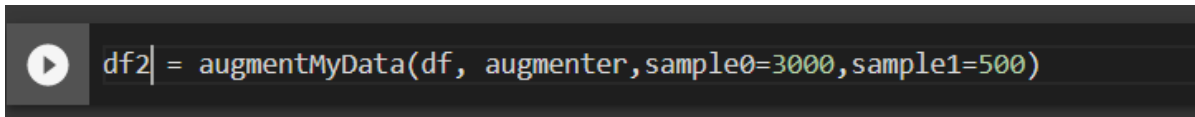


Figure 3.22 Augmentation des données

Repartition du nouveau data set :

```
[ ] train_texts, temp_texts, train_labels, temp_labels = train_test_split(df2['commentaire'], df2['Label'], random_state=42,
                                                                    test_size=0.3)

train_texts=train_texts.apply(preprocess)
temp_texts=temp_texts.apply(preprocess)

val_texts, test_texts, val_labels, test_labels = train_test_split(temp_texts, temp_labels, random_state=42,
                                                                    test_size=0.5)

len(train_texts), len(val_texts), len(test_texts)

(6622, 1419, 1419)
```

Figure 3.23 Repartition du nouveau dataset

Train Data	70% ⇔ 6622
Validation Data	15% ⇔ 1419
Test Data	15% ⇔ 1419
Total Data	100% ⇔ 6460

Tableau 3.5 Partion du nouveau donnée

Re-ajustement du modèle avec le nouveau data set et les memes hyperparameter :

Hyperparameter	Options
N° epochs	5
Batch size per device during training	16
Batch size per device during evaluation	32
Warump_steps	500
Weight decay	0.01

Tableau 3.6 Parametre d'entrainement

```
[ ] training_args = TrainingArguments(
    output_dir='./results',          # output directory
    num_train_epochs=5,              # total number of training epochs
    per_device_train_batch_size=16,  # batch size per device during training
    per_device_eval_batch_size=32,   # batch size for evaluation
    warmup_steps=500,                # number of warmup steps for learning rate scheduler
    weight_decay=0.01,               # strength of weight decay
    logging_strategy='epoch',
    evaluation_strategy='epoch'
)

dziribert_apres_augmentation = Trainer(
    model=model,                    # the instantiated 🤗 Transformers model to be trained
    args=training_args,             # training arguments, defined above
    train_dataset=train_dataset,    # training dataset
    eval_dataset=val_dataset,       # evaluation dataset
    compute_metrics=compute_metrics
)

dziribert_apres_augmentation.train()
```

Figure 3.24 Re-entrainement du modèle

Validation du modèle :

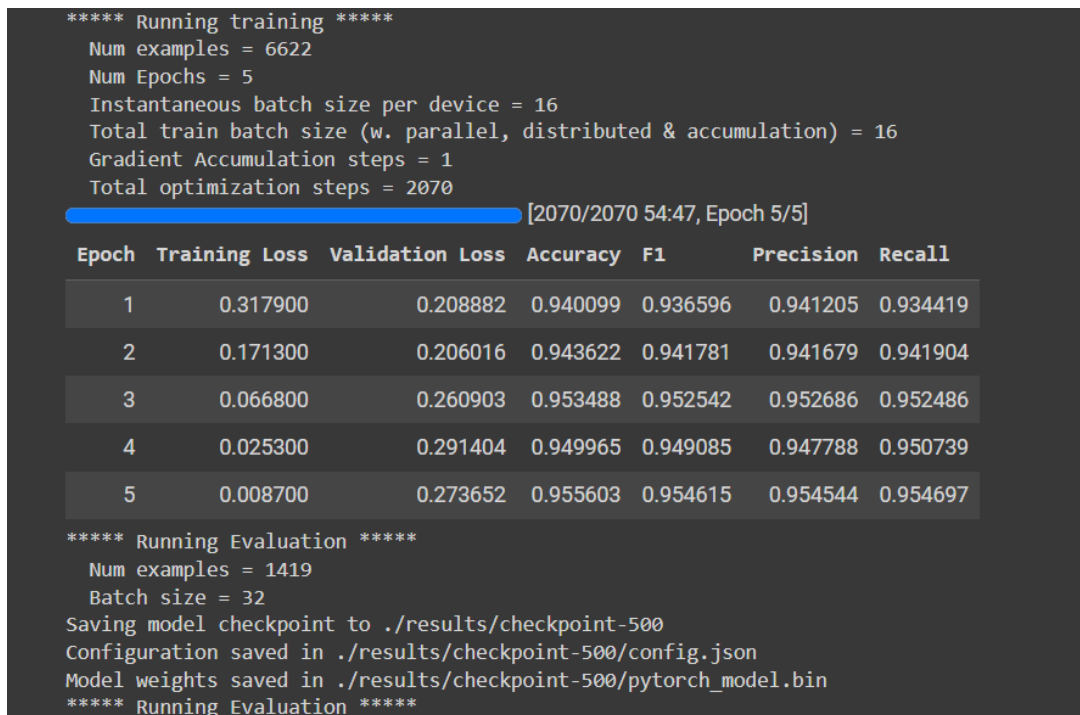


Figure 3.25 Validation du nouveau modèle

Classification report :

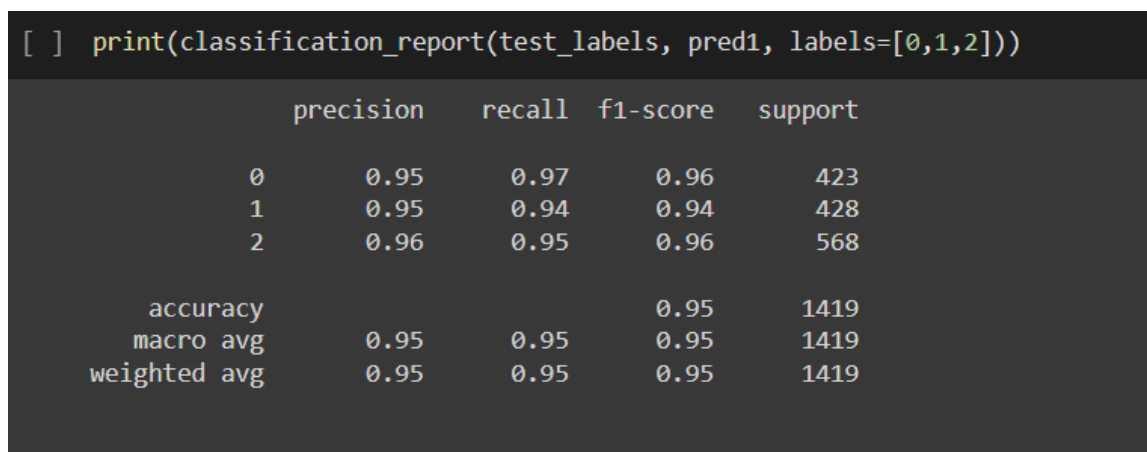


Figure 3.26 Classification report

Matrice de confusion :

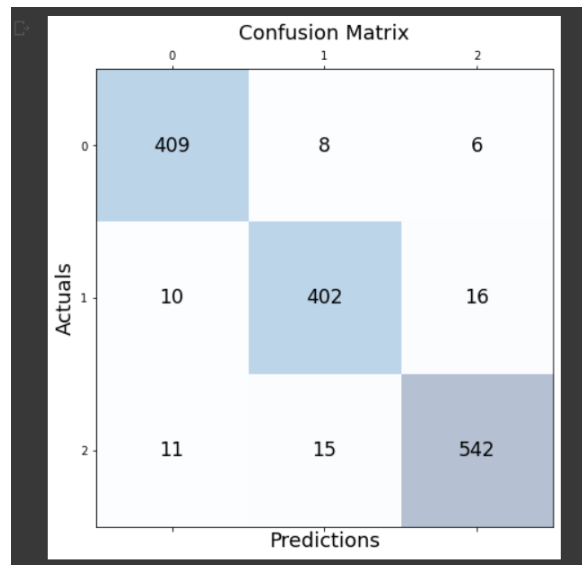


Figure 3.27 Matrice de confusion

Test du nouveau modèle avec les différentes data set :

- Test avec dialecte marocain :

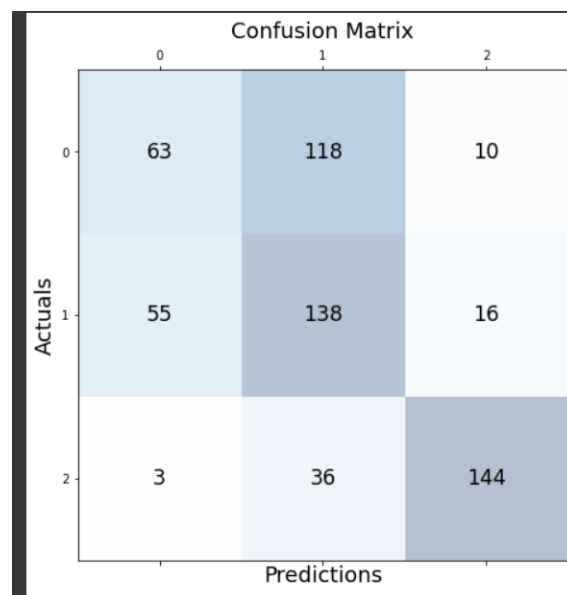


Figure 3.28 Matrice de confusion pour dialecte marocain avec nouveau model

- Test avec multidialecte :

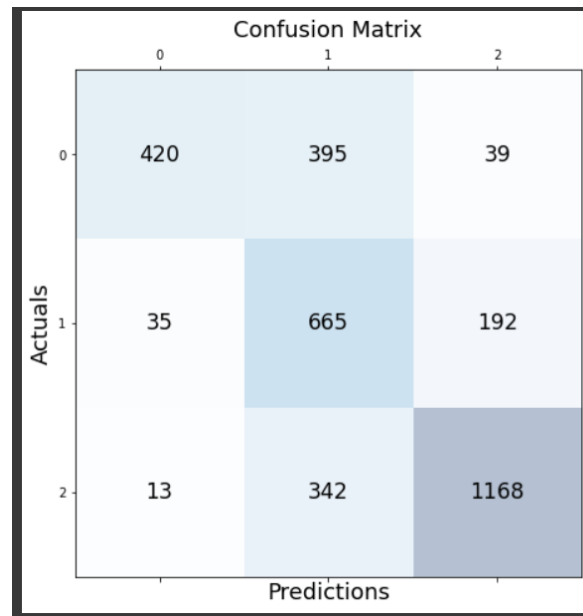


Figure 3.29 Matrice de confusion pour multidialecte avec nouveau model

Nous remarquons une légère amélioration en comparant avec le modèle avant l'augmentation des données d'entraînement. Mais ça ne suffit pas pour obtenir une bonne prediction, donc dans la section suivante nous allons re-ajuster notre modèle avec un nouveau jeu de données plus varié pour gagner en matière de précision et pour être loin du risque que notre modèle s'adapte trop aux données d'entraînement.

(Pour référence, nous avons utilisé 6622 échantillons d'entraînement et 1419 échantillons de validation).

❖ Réajustement du model avec un nouveau jeu données :

Le jeu données choisi pour être le nouveau données d'entraînement est la concaténation de toutes les dialectes. Aussi nous allons augmenter ce jeu de données pour qu'il soit équilibrer de plus. Ci-dessous une légère description.

commentaire	classe	Label
... يكفي انك جبان تسولف بالدايركت مثل الحریم تدور	hate	1
... عشان انت بس مش وسخ زيها ولا وسخ زيهم فده حصل ب	normal	2
... لاغازيتا ياربى لا عزاء لا ما عندكش نتفارضو	hate	1
... فده متخلق شمس ورجلة ياربك وقده لطفي قليل ربا	normal	2
... هو فقط قال الحقيقة ما قدمت قطر للبنان ليس منحة	normal	2

Figure 3.30 Presentation du nouveau data

```
[57] df2['Label'].value_counts()

2    9822
1    6063
0    5906
Name: Label, dtype: int64
```

Figure 3.31 Description de données

```
[58] train_texts, temp_texts, train_labels, temp_labels = train_test_split(df2['commentaire'], df2['Label'], r
                                     test_size=0.3)

train_texts=train_texts.apply(preprocess)
temp_texts=temp_texts.apply(preprocess)

val_texts, test_texts, val_labels, test_labels = train_test_split(temp_texts, temp_labels, random_state=4
                                     test_size=0.5)

len(train_texts), len(val_texts), len(test_texts)

(15253, 3269, 3269)
```

Figure 3.32 Partition de données

Train Data	70% ⇔ 15253
Validation Data	15% ⇔ 3269
Test Data	15% ⇔ 3269
Total Data	100% ⇔ 20891

Tableau 3.7 partitions de jeu de données

Evaluation du nouveau modèle :

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.636900	0.522966	0.787091	0.777276	0.776258	0.778390
2	0.373800	0.564228	0.780055	0.777339	0.774666	0.787522
3	0.202800	0.736002	0.779137	0.774286	0.772440	0.777735
4	0.119000	1.067414	0.793209	0.783227	0.785508	0.782702
5	0.073800	1.201203	0.789538	0.780420	0.782219	0.779033

Saving model checkpoint to ./results/checkpoint-500
Configuration saved in ./results/checkpoint-500/config.json
Model weights saved in ./results/checkpoint-500/pytorch_model.bin
**** Running Evaluation ****
Num examples = 3269

Figure 2.33 Evaluation du nouveau modèle

Classification report :

```
[ ] print(classification_report(test_labels, pred, labels=[0,1,2]))
```

	precision	recall	f1-score	support
0	0.85	0.87	0.86	854
1	0.68	0.67	0.67	892
2	0.84	0.83	0.84	1523
accuracy			0.80	3269
macro avg	0.79	0.79	0.79	3269
weighted avg	0.80	0.80	0.80	3269

Figure 2.34 Classification report

Matrice de confusion :

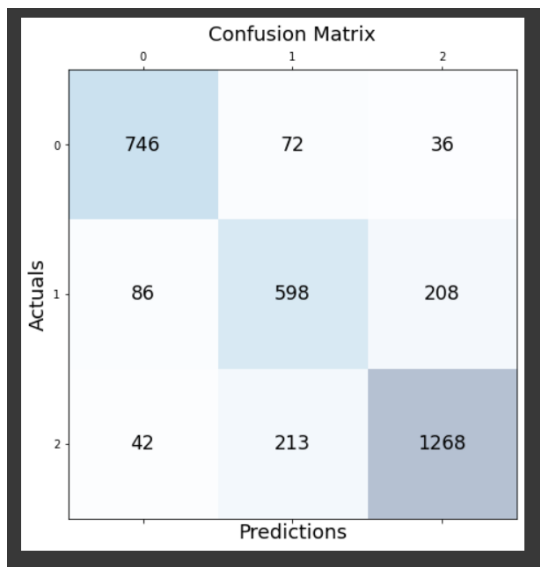


Figure 2.35 Matrice de confusion

NB : Concernant la partie ajustement du modèle , nous avons suivi le meme demarche auparavant. Nous remarquons d'après la figure 2.31 une bonne precision du modèle pour le bien prouver . Maintenant,nous allons passer au partie test avec les differentes dialectes . (Pour référence, nous avons utilisé 15253 échantillons d'entraînement et 3269 échantillons de validation).

Test du modèle avec les differentes dialectes :

- Test avec dialecte tunisien :

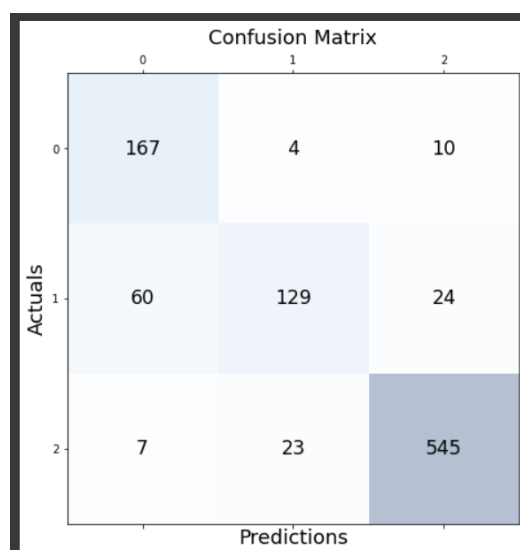


Figure 2.36 Matrice de confusion du dialecte tunisien avec le nouveau modèle

- Test avec dialecte marocain :

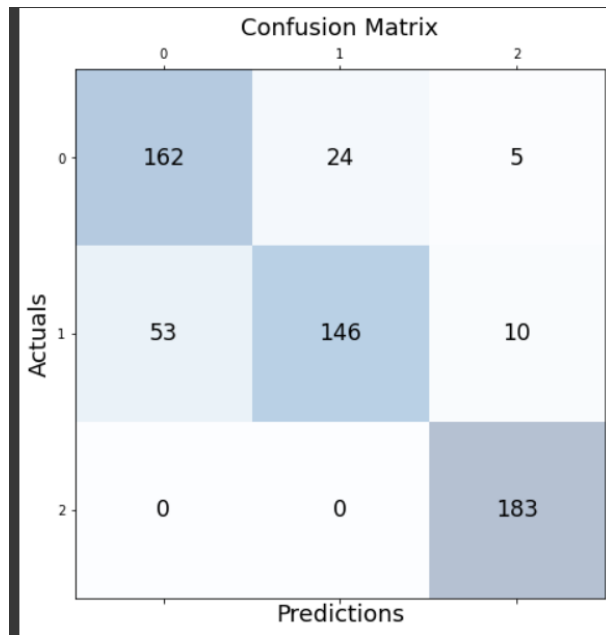


Figure 2.37 Matrice de confusion du dialecte marocain avec le nouveau modèle

- Test avec dialecte égyptien :

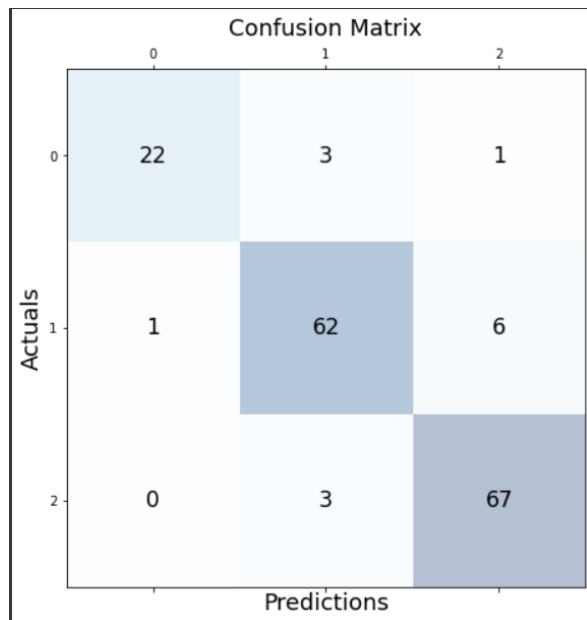


Figure 3.38 Matrice de confusion du dialecte marocain avec le nouveau modèle

- Test avec dialecte libanais :

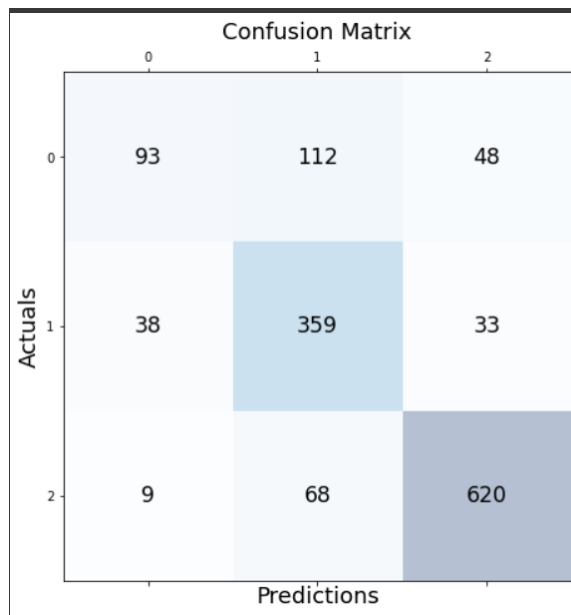


Figure 3.39 Matrice de confusion du dialecte libanais avec le nouveau modèle

Remarques :

Finalement , nous pouvons arriver à ajuster notre modèle pour obtenir une meilleure prediction et ceci apres avoir augmenter notre jeu de données , le rendre variés et équilibré en terme de nombre d'exemples par classes .

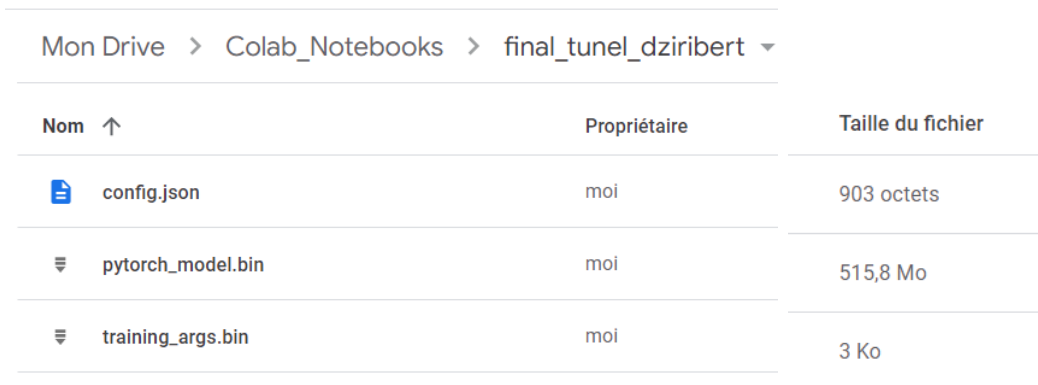
Nous remarquons aussi , que certainement , notre modèle n'arrive pas toujours à differencier entre la classe 1 (hate) et la classe 0 (abusive) et c'est à cause que ces deux classes sont très proches contrairement au classe 2 (normal) .

Nous avons suivi les memes approche et demarches avec les deux autres modèles **Arabertv2** et **bert-base-arabic** et nous avons presque les memes resultats que ce modèle .

Mainetenant , nous allons sauvegarder notre modèle :

```
dziribert.save_model("final_tunel_dziribert")  
Saving model checkpoint to final_tunel_arabert_v2  
Configuration saved in final_tunel_arabert_v2/config.json  
Model weights saved in final_tunel_arabert_v2/pytorch_model.bin
```

Figure 3.40 Sauvegarde du modele



Mon Drive > Colab_Notebooks > final_tunel_dziribert ▾




Nom ↑	Propriétaire	Taille du fichier
 config.json	moi	903 octets
 pytorch_model.bin	moi	515,8 Mo
 training_args.bin	moi	3 Ko

Figure 3.41 Sauvegarde du modèle dans le Drive

3.5 Conclusion

Dans ce dernier chapitre, nous avons illustré au premier lieu le schéma général de notre projet ainsi nos différents environnements de travail et outils qui nous ont permis de réaliser les fonctionnalités de notre projet. Et en second lieu, nous avons détaillé l'ensemble des étapes ainsi les approches suivies pour atteindre l'objectif de notre projet et parvenir aux résultats souhaités.

Conclusion générale

Le contenu haineux et toxique généré par une partie des utilisateurs dans les médias sociaux est un phénomène croissant qui a incité les chercheurs à consacrer des efforts substantiels à l'identification du contenu haineux. Nous avons besoin non seulement d'un modèle efficace de détection automatique des discours haineux basé sur l'apprentissage automatique avancé et le traitement du langage naturel, mais aussi d'une quantité suffisamment importante de données annotées pour entraîner un modèle. L'absence d'une quantité suffisante de données annotées sur les discours haineux, ainsi que les biais existants, ont été le principal problème dans ce domaine de recherche. Pour répondre à ces besoins, nous avons introduire dans ce projet une nouvelle approche d'apprentissage par transfert basée sur un modèle de langage pré-entraîné existant appelé BERT. Plus précisément, nous avons étudié la capacité de BERT à capturer le contexte haineux dans le contenu des médias sociaux en utilisant de nouvelles méthodes de réglage fin basées sur l'apprentissage par transfert. Pour évaluer l'approche que nous avons proposée, nous avons utilisé des ensembles de données en dialecte arabe accessibles au public qui ont été annotés pour le racisme, le sexisme, la haine ou le contenu offensant. Les résultats montrent que notre solution obtient des performances considérables sur ces jeux de données en termes de précision et de rappel par rapport aux approches existantes.

Nous avons mené le projet en suivant la méthodologie de gestion CRISP-DM, qui couvrait tous les piliers de la durée de vie d'un projet de science des données. Nous avons commencé par une compréhension large de notre problème, puis une recherche approfondie des jeux de données de balayage disponibles. Ensuite, nous sommes passés à la phase de modélisation, où nous avons testé nos modèles existants avec les différents dialectes.

Compte tenu du temps et des ressources informatiques limités dont nous disposions, les résultats obtenus sont considérés comme très convenables et ils ont définitivement répondu à nos attentes et au résultat souhaité.

Il est toujours possible d'améliorer le projet. On peut prévoir des travaux et des perspectives d'avenir, y compris, mais sans s'y restreindre :

- Développer une interface pour bien visualiser notre résultat.
- Appliquer Les méthodes ensemblistes sur nos trois modèles pour avoir une meilleur prédictions

Netographie

1. CCK , <https://cck.rnu.tn/>
2. CRISPDM, <https://www.datascience-pm.com/crisp-dm-2/>,
3. NLP, <https://www.lebigdata.fr/traitement-naturel-du-langage-nlp-definition/>,
4. ALGORITHMES NLP, <https://www.analyticssteps.com/blogs/top-nlp-algorithms>,
5. [python-nlp-libraries-features-us-cases-pros-and-cons/](#),
6. JUPYTER, <https://jupyter.org/>,
7. <https://paperswithcode.com/task/sentiment-analysis>
8. <https://www.kaggle.com/code/ejmejim/commonlit-bert-submission-video-tutorial?scriptVersionId=64939932>
9. <https://inside-machinelearning.com/bert-enfin-un-tutoriel-simple-et-rapide/>
10. <https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-check-if-your-deep-learning-model-is-underfitting-or-overfitting.md>
11. https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/#h2_7
12. https://www.researchgate.net/publication/335409647_Use_of_Natural_Language_Processing_to_Identify_Inappropriate_Content_in_Text
13. <https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html>
14. <https://huggingface.co/>
15. <https://neovision.fr/data-augmentation-solutions-manque-donnees/>

