Université de Paris Cité
UFR Mathématiques et Informatique

Deep learning pour l'analyse de texte

Master 1 Informatique parcours Vision et Machine Intelligente

Ahmad NAJJAR – Lazar ANDJELOVIC

Encadré par Mme Nicole VINCENT et M Baptiste BOHET

Année universitaire 2022-2023

Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué au succès de notre travail et qui nous ont aidé lors de la rédaction de ce rapport.

Tout d'abord, nous tenons à exprimer nos sincères remerciements à nos encadrants, madame **Nicole VINCENT** et monsieur **Baptiste BOHET**, pour leur accueil chaleureux, leur précieuse collaboration et leur expertise partagée tout au long du projet. Leur confiance nous a permis de nous épanouir pleinement dans nos missions et leur précieux soutien a été particulièrement apprécié dans les moments les plus délicats.

Nous tenons à remercier vivement le responsable de notre master, monsieur **Laurent WENDLING**, de l'Université Paris Cité ainsi que tous nos enseignants.

Table des matières

Chapitre 1

Introduction

Pendant des siècles, les scientifiques ont travaillé sur la datation des textes pour comprendre comment l'écriture a évolué au fil des siècles ou des années. Avant l'informatique, la datation de texte était souvent basée sur des méthodes manuelles, mais au cours de la dernière décennie, les scientifiques ont commencé à utiliser différentes approches informatique pour aider à dater les textes avec plus de précision et de rapidité.

Notre projet a été proposé dans le cadre d'une collaboration avec le laboratoire THALIM de l'Université Sorbonne Nouvelle. L'objectif principal de notre projet est la datation de texte en utilisant des méthodes de classification adaptées aux données textuelles.

Nous utiliserons un modèle de deep learning pour le traitement automatique des Langage (TAL), qui est une version distillée du Bert Français (Camembert). Nous espérons que ce modèle nous fournira une méthode plus précise et plus simple pour la classification multi-classes. La quantité de texte lors de l'apprentissage du modèle est une étape importante à étudier ainsi que le pré-traitement des textes pour assurer une classification correcte. Il est à noter que nous considérons les textes comme une suite de mots.

Le rapport est structuré en plusieurs parties. Nous commencerons par une revue de l'état de l'art pour présenter les différentes méthodes existantes de classification adaptées aux données textuelles. Puis nous parlerons de la méthode que nous utilisons. Ensuite, nous décrirons les différentes étapes du projet, la réalisation des apprentissages et la comparaison des résultats. Enfin, nous discuterons des résultats obtenus et leur interprétation. Notre rapport se conclura par une conclusion et une bibliographie.

Chapitre 2

État De L'Art

La datation des textes est le centre d'intérêt de plusieurs chercheurs depuis longtemps.

Des auteurs ont proposé une approche de datation de documents "NeuralDater" [1] qui utilise des réseaux de graphes convolutif.

Avec cette méthode, des dates de document précises peuvent être déduites même pour des documents arbitraires provenant du Web.

Selon les expériences menées NeuralDater surpasse toutes les autres méthodes avec une marge significative sur les deux ensembles de données. Sachant que les autres méthodes sont tirés d'articles dont BurstySimDater (TF-IDF), MaxEnt-Time-NER, MaxEnt-Joint, MaxEnt-Uni-Time, CNN.

Ils ont fait leur teste grâce aux données de l'Associated Press (une agence de presse mondialement connu) et du New York Times (l'un des plus grands journaux américains).

Leur précision est différente par rapport aux données d'apprentissage, sur les données de l'APW ils ont obtenu 64.1% de précision or sur les données du NYT ils ont obtenu 58.9% de précision.

Ils ont réussi à avoir environ 19% de précision en plus pour les deux datasets comparé à BurstySimDater, ce qui n'est pas négligable. Mais aussi 8% de précision en moyenne en plus qu'un réseau classique comme le CNN (réseau de neurones convolutifs).

Des recherches supplémentaires seraient nécessaires pour déterminer la faisabilité et l'efficacité d'une telle adaptation. Cette méthode se rapproche le plus de la méthode que nous utilisons dans notre projet.

Un autre groupe de chercheurs ont proposé un système neuronal de datation de documents qui utilise à la fois des informations contextuelles et temporelles dans les documents "AD3"[2].

Ils se sont inspirés du modèle "NeuralDater". C'est un mixe entre un réseau Attentive Context Model (AC-GCN) et un Ordered Event Model (OE-GCN), tous les deux font partie des réseaux de graphe convolutif.

Grâce à une expérimentation approfondie sur plusieurs ensembles de données du monde réel, l'efficacité de l'AD3 est démontrée par rapport aux autres méthodes existantes.

AD3 exploite deux principaux types de signaux du document - syntaxique et heure de l'événement - pour prédire l'horodatage du document.

Contrairement à NeuralDater, AD3 traite ces deux modèles complètement séparément et les combine ultérieurement.

De plus, AD3 utilise des mécanismes d'attention dans chacun de ces deux modèles, c'est ce qu'utilise les réseau de type transformers comme BERT. Donc le type de réseau est proche de celui que nous utilisons dans ce projet.

Parlons maintenant de résultat, alors si on compare le NeuralDater vu précédemment et de l'AD3, on obtient que la précision de l'AD3 est 4% plus grande que le NeuralDater sachant que les deux sont entrainés sur les mêmes données.

Des chercheurs ont testé une autre solution au problème de la détermination de l'horodatage d'anciens documents numérisés[3], où l'horodatage peut être corrompu ou indisponible.

En utilisant l'analyse de la rafale, l'approche proposée peut estimer avec précision l'horodatage d'un document en fonction de son contenu.

L'analyse de la rafale aide à déterminer l'horodatage d'un document en capturant les tendances d'utilisation du vocabulaire au cours de la période correspondante.

L'intuition est que les rafales de fréquence des termes peuvent être utilisées pour identifier la période pendant laquelle un terme particulier a été fréquemment utilisé, et ces informations peuvent être utilisées pour estimer l'horodatage d'un document.

Cette approche est basée sur l'idée que les rafales de fréquence des termes capturent les tendances d'utilisation du vocabulaire au cours de la période correspondante et peuvent donc s'avérer utiles dans la datation des documents.

Cette méthode est basée sur la similarité lexicale et la rafale, qui sont des caractéristiques indépendantes de la langue.

Par conséquent, il est possible que cette approche puisse également être étendue à d'autres langues.

Néanmoins, des recherches supplémentaires seraient nécessaires pour valider l'efficacité de cette approche pour les documents non anglais.

Ensuite dans un autre article, des chercheurs ont présenté un outil permettant de déterminer l'horodatage d'un document non horodaté à l'aide de modèles de langage temporel[4], ce qui peut être extrêmement utile pour rechercher des pages Web ou des documents.

Les modèles de langage temporel sont utilisés pour analyser le langage utilisé dans un document et déterminer la période de temps la plus probable au cours de laquelle il a été créé.

Cet outil présenté prend en entrée un document non horodaté et présente une estimation des temps/périodes de création possibles.

L'outil utilise des modèles de langage temporel pour analyser le langage utilisé dans le document et déterminer quelle période est la plus susceptible d'être basée sur cette analyse.

L'outil présenté peut être particulièrement utile dans les situations où l'heure de création d'un document est inconnue ou contestée, mais il est important de déterminer quand le document a été créé. Par exemple, cet outil pourrait être utilisé par des historiens ou des chercheurs qui essaient de dater un document ou un écrit particulier.

Il pourrait également être utilisé par des journalistes ou des vérificateurs de faits qui tentent de vérifier l'authenticité d'un document qui a été partagé en ligne.

De plus, cet outil pourrait être utile aux organismes chargés de l'application de la loi qui enquêtent sur des crimes impliquant des preuves numériques.

Concernant les grands modèles pré-formés basés sur les Transformers dans le traitement du langage naturel, des chercheurs ont proposé de générer des modèles plus petits[5] qui gèrent moins de langues selon les corpus ciblés, les auteurs visent pour réduire le nombre total de paramètres et faciliter le déploiement dans des applications de production réelles.

La taille des grands modèles est souvent un inconvénient pour leur déploiement dans des applications de production réelles.

En effet, la plupart des paramètres sont situés dans la couche des représentations vectorielles continues.

Par conséquent, la réduction de la taille du vocabulaire devrait avoir un impact important sur le nombre total de paramètres.

Les chercheurs présentent une évaluation de versions plus petites de BERT multilingue sur l'ensemble de données XNLI, mais ils pensent que cette méthode peut être appliquée à d'autres transformateurs multilingues.

Les résultats obtenus confirment qu'ils peuvent générer des modèles plus petits qui conservent des résultats comparables, tout en réduisant jusqu'à 45% le nombre total de paramètres.

Chapitre 3

Contribution

3.1 Les Transformers

Dans le domaine de l'apprentissage machine, un *transformer*[6] est un modèle basé sur le Deep Learning, conçu spécifiquement pour le Traitement Automatique du Langage (TAL) ou le Natural Language Processing (NLP) en anglais. Il utilise un mécanisme d'attention pour traiter les données d'apprentissage séquentielles. C'est une technique qui permet aux modèles de se concentrer sur des parties bien spécifiques, en fonction de leur importance pour la tâche en cours.

Les transformers permettent à la machine d'apprendre automatiquement à partir de séquences de données sans avoir à être spécifiquement programmées pour elles.

Les transformers fonctionnent de la même manière que les réseaux de neurones récurrents (RNN) lors du traitement des données séquentielles, ce qui les rend particulièrement adaptés au traitement du langage naturel.
Ce sont des modèles très complexes et représentent plusieurs centaines de millions de paramètres comme par exemple BERT (Bidirectional Encoder Representations from Transformers).

Dans le cadre de notre projet, nous avons utilisé le modèle DistilCamemBERT comme modèle de transformers.

3.2 DistilCamemBERT

Le modèle DistilCamemBERT est une version distillée du CamemBERT qui est un modèle de type BERT spécifiquement développé pour le français et est pré-entraîné sur un large corpus de textes français.

Une version distillée est une version qui nécessite moins de paramètres, et permet d'avoir les mêmes résultats de précision que la version non distillée.

3.3 Étude Expérimentale

Comme étant mentionné, le but de notre projet est de dater des textes français, pour cela nous avons utilisé des bases de données fournies par nos encadrants sur lesquelles nous avons appliqué différentes techniques que nous allons expliquer dans la suite de notre rapport.

3.3.1 La Base De Données

Au cours de notre projet, nos encadrants nous ont fourni des corpus sur différentes époques et de tailles différentes.

Les textes des deux premiers corpus étaient repartis sur cinq époques comme suit : $[1800 - 1849]$, $[1850 - 1899]$, $[1900 - 1949]$, $[1950 - 1999]$ et $[2000 - 2022]$.

Et les textes du troisième corpus étaient repartis sur cinq époques comme suit : $[Avant\ 1830]$, $[1830 - 1865]$, $[1866 - 1925]$, $[1926 - 1979]$ et $[1980 - 2022]$.

Le premier corpus que nous avons eu contenait 10 textes par époque.

Le deuxième contenait 21 textes par époque.

Et le dernier contenait 1000 textes au total.

Dans les trois corpus, ils y avaient des textes en *.epub* et en *.docx*, donc nous les avons convertis en fichiers *.txt* pour exécuter des prétraitements afin de lancer notre apprentissage.

3.3.2 Prétraitement : Nettoyage Des Textes

Lorsque nous avons converti les textes *.epub* en *.txt* en utilisant le logiciel *Calibre*, nous avons remarqué que les textes présentaient du bruit (des lettres sans signification, des astérisques et des liens URL). Nous avons donc nettoyé ces textes pour avoir de bons résultats. En plus de ce nettoyage qui était nécessaire, nous en avons fait d'autres pour qu'on puisse savoir si cette étape aurait des impacts sur les résultats. Nous avons regroupé trois catégories de nettoyages :

Nettoyage A

- Nettoyage nécessaire dû à la conversion depuis *.epub*.
- Enlever les tabulations dans le texte.
- Enlever les retours chariots dans le texte.
- Ajouter une espace après chaque ponctuation.

Nettoyage B

- Nettoyage A.
- Minusculisation des lettres.
- Suppression des chiffres.

Nettoyage C

— Nettoyage B.

— Enlever les traits d'unions et les remplacer par des espaces.

3.3.3 Prétraitement : Tokénisation Et Labélisation

Tokénisation

Pour pouvoir utiliser le modèle DistilCamemBERT, nous devions tokéniser les textes c'est-à-dire les découper en morceaux de tokens ou jetons (512 tokens pour notre code). Pour cela, le modèle utilise l'algorithme Sentencepiece[7].

Nous avons remarqué que le mot « éclatant » n'est pas divisé mais «éclatants» est divisé en « éclatant » et « s » (Figure 3.1). En recherchant un peu nous avons trouvé pourquoi l'algorithme faisait cela.

Cela aide le modèle à apprendre que le mot « éclatants » est formé en utilisant le mot « éclatant » avec des significations légèrement différentes, mais le même mot racine.

Nous avons aussi remarqué que les espaces ont été remplacés par des tirets du bas (_).

Pour initialiser le tokeniseur, on a utilisé ce code :

```
from transformers import CamembertTokenizer

tokenizer =
    CamembertTokenizer.from_pretrained("cmarkea/distilcamembert-base")
```

FIGURE 3.1 – Exemple de tokénisation

Labélisation

Ainsi, pour pouvoir comprendre et analyser le contenu textuel nous avons accédé à la labélisation des morceaux de textes, c'est-à-dire leur attribuer des étiquettes suivant l'époque

de chaque texte. Par exemple, les époques [1800 – 1849], [1850 – 1899], [1900 – 1949], [1950 – 1999] et [2000 – 2022] ont été labélisées respectivement par 0, 1 ,2 ,3 et 4.

3.3.4 Prétraitement : Encodage

Pour pouvoir lancer l'apprentissage, il faut transformer les morceaux de 512 tokens sous forme numérique pour que cela soit compréhensible par le modèle.

L'encodage permet d'identifier chaque token présent à l'aide d'un entier correspondant à sa position dans le vocabulaire de l'algorithme mentionné précédemment, c'est ce qu'on appelle les *input_ids* (Figure 3.2). Mais il permet aussi de masquer les jetons de remplissage (padding) et d'indiquer sur quels tokens l'attention doit se porter dans chaque phrase, c'est ce qu'on appelle les *attention_mask* (Figure 3.3).

```
train_input_ids[0]

]: tensor([    5, 18995,  1370,  1190,     8, 30050,   261,  1370,  1190,    55,
         22107,   153,   930,    25,   813,   525,     9,   180,  4438,  1190,
          9898,  1038,     9,    21, 17251,  1991,  1163,   888, 15877, 10197,
         10759, 17100,   403,  3372,   657,   374,  4438,  1163,  1098,     8,
         25708,    35,  6780,     3,    43,   158,   349,    13,  8938,     7,
            22,    21, 17333,  1190,   219,  1370,  1190,  9843,   172,    23,
           510,    32, 25708,    35,  6780,     9,   310,   403,  3372,   657,
           122,   374,     8,   831,    23,   510,     8, 16573,  1370,  1190,
             3,    43,   170,  7605,   273,    23,  1705,   770,  4438,    13,
         16573,  1370,  1190,    20,  1726,     7,  4438,    13,   375,    22,
           563,     7,  4438,    13, 30307,     7,  4438,    13,  3946,  4438,
```

FIGURE 3.2 – Exemple d'input_ids

```
train_attention_masks[0]

]: tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

FIGURE 3.3 – Exemple d'attention_mask

3.3.5 Entrainement Du Modèle

Répartion De La Base De Données

Comme pour tous les modèles de deep learning, nous avons réparti la base de données en données d'apprenstissage, de validation et et de test avec les proportions respectives 80%, 10% et 10%.

Lancement Du Modèle

Tout d'abord, nous avons appelé le modèle avec cette ligne de commande :

```
from transformers import CamembertForSequenceClassification

model =
    CamembertForSequenceClassification.from_pretrained('cmarkea/distilcamembert-base',
    num_labels=5)
```

En cherchant sur le site de **Hugging Face** [8], nous sommes tombés sur la classe *Trainer* qui fournit une API pour entraîner notre modèle.

Ensuite, nous avons passé les paramètres que nos encadrants nous ont conseillé :

— EarlyStopping avec une patience égale à 3, qui aboutit à l'arrêt de l'entraînement du modèle si les performances sur l'ensemble de validation ne s'améliorent pas pendant trois itérations consécutives.

— Learning_rate égale à 10^{-5}, c'est le taux d'apprentissage qui est un paramètre de réglage dans un algorithme d'optimisation qui détermine la taille du pas à chaque itération tout en se déplaçant vers un minimum d'une fonction de perte.

Nous avons aussi un optimisateur qui est intégré dans la clase *Trainer*, c'est l'optimisateur Adam qui est une méthode d'optimisation adaptative qui ajuste le taux d'apprentissage.

Ainsi, *Trainer* inclut la méthode CrossEntropyLoss qui mesure la disparité entre une distribution de probabilités prédite par un modèle et la distribution de probabilité réelle (étiquettes) des données.

Sur l'ordinateur que nos encadrants nous ont prêté, nous avons utilisé un batch_size égale à 4. Alors que sur nos ordinateurs personnels, en codant sur Google colab, nous avons initialisé cette variable à 32. Nous précisons que le "batch size" correspond au nombre d'échantillons d'apprentissage utilisés pour mettre à jour les poids du modèle lors d'une itération d'entraînement.

Nous avons également lancé notre apprentissage sur 5 épochs, et nous avons remarqué que les meilleurs résultats sont celles des trois premières épochs.

3.3.6 Résultats

Comme nous avons mentionné ci-haut, nous avons lancé l'apprentissage avec chacun des trois nettoyages sur la base de données, et nous avons constaté que cette étape avait une forte influence sur les résultats.

Avec le nettoyage C, nous avons pu avoir les meilleurs résultats que nous développeront ci-dessous. Nous avons lancé un apprentissage sur chaque corpus :

— Corpus 1 : 10 textes par époque.

— Corpus 2 : 21 textes par époque.

— Corpus 3 : 1000 textes au total.

Résultats De La Validation

Le tableau 3.1 montre les mesures d'évaluation de la validation de chaque apprentissage.

Corpus	1	2	3
Accuracy	0.485	0.735	0.981
Précision	0.457	0.74	0.893
Rappel	0.487	0.724	0.902

TABLE 3.1 – Les mesures d'évaluation de la validation

L'accuracy représente le pourcentage de prédictions correctes par rapport à toutes les prédictions effectuées.

Comme on peut l'apercevoir l'apprentissage du corpus 3 a le pourcentage de prédiction correctes le plus élevé comparé au corpus 1 et 2 ce qui laisse penser que le nettoyage n'ai pas la chose la plus importante dans notre apprentissage mais aussi la quantité de texte.

La précision (équation 3.1) permet de connaître le nombre de prédictions positifs bien effectuées.

Le deuxième corpus a une précision légèrement supérieure (0.74) , indiquant qu'il y a moins de faux positifs dans les prédictions positives. Le troisième corpus a la précision la plus élevé 0.893 donc elle minimise à presque 90% le nombre de Faux Positifs. Quant à la précision du corpus 1, elle est assez faible comparée aux autres mais reste cohérente.

$$Precision = \frac{Vrai\_Positif}{Vrai\_Positif + Faux\_Positif} \tag{3.1}$$

Le rappel (équation 3.2) permet de savoir le pourcentage de positifs bien prédit par notre modèle.

Le troisième corpus a le rappel le plus élevé (0.902), indiquant qu'il maximise le nombre de Vrai Positif à 90% mais cela ne dit pas que ça ne fait pas de fautes. Le premier corpus a un faible rappel (0.487) par rapport au deuxième corpus (0.724) et au troisième (0.902).

$$Rappel = \frac{Vrai\_Positif}{Vrai\_Positif + Faux\_Negatif} \tag{3.2}$$

En résumé, l'apprentissage sur le troisième corpus semble avoir les meilleures performances globales en termes d'accuracy, de précision et de rappel, tandis que le premier corpus a les performances les plus faibles. La taille du corpus et le nombre de textes

par époque peuvent influencer les résultats et même peut être que cela est le plus important pour les performances de notre apprentissage . La phase de test nous le confirmera.

Nous avons réaliser la fonction de perte pour l'apprentissage fait sur le corpus 3 (Figure 3.4), elle montre que le modèle généralise bien les données.

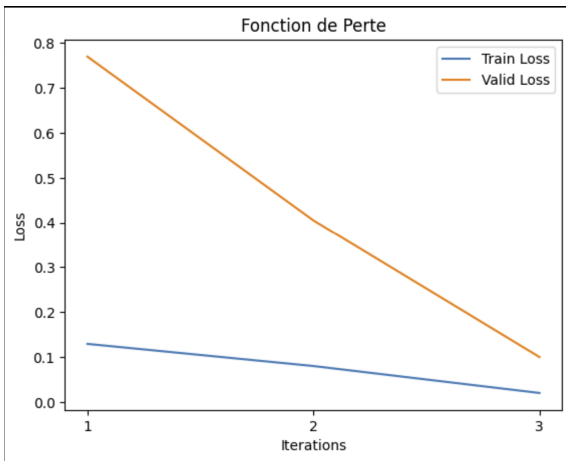


FIGURE 3.4 – Courbe de loss pour le corpus 3

Résultats De La Phase Test

Après les entrainements du modèle sur les différents corpus, nous les avons sauvegardés. À ce stade, nous avons lancé chaque modèle sur l'ensemble des test du corpus correspondant (sur 10% de chaque époque du corpus). Le tableau 3.2 représente l'accuracy du test.

Corpus	1	2	3
Accuracy	0.419	0.792	0.98

TABLE 3.2 – L'accuracy de la phase test

L'accuracy de la phase de test vient de nous confirmer les résultat de l'entrainement, c'est-à-dire que le test sur le corpus3 est assez élévé comparé aux autres. Donc on peut à ce stade confirmer l'importance de la quantité de texte pour la datation de textes avec le model de transformers.

Ci-dessous, les matrices de confusion pour la phase de test des differents corpus (Figures 3.5, 3.6 et 3.7).

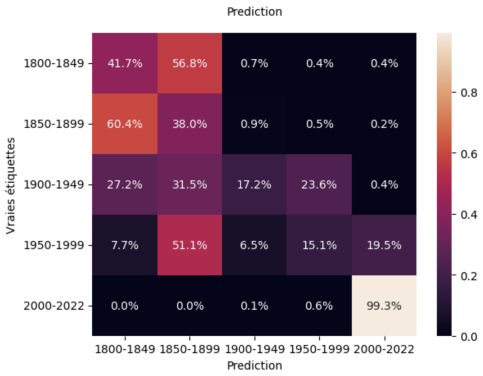


FIGURE 3.5 – Matrice de confusion pour le corpus 1

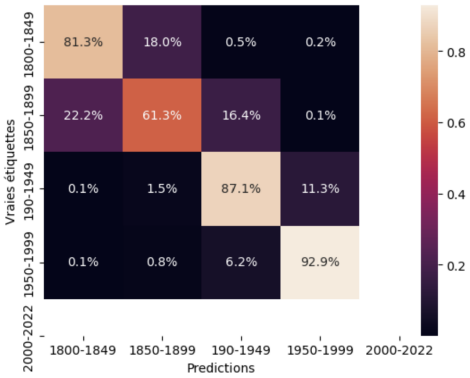


FIGURE 3.6 – Matrice de confusion pour le corpus 2

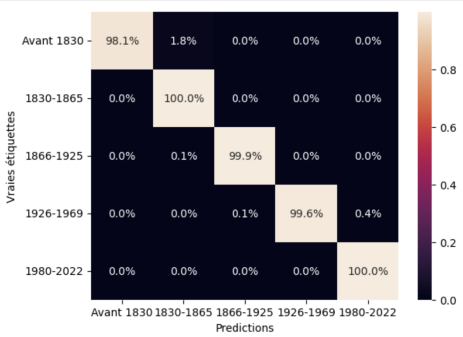


FIGURE 3.7 – Matrice de confusion pour le corpus 3

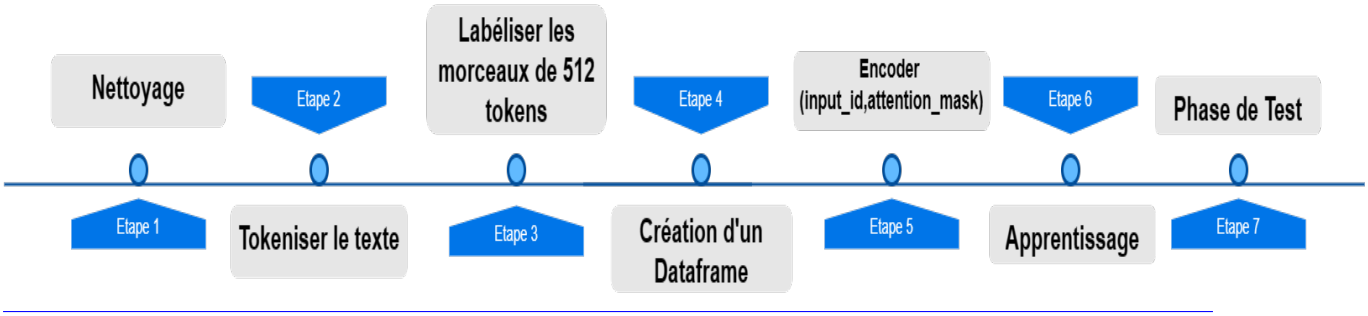
Alors comme vous pouvez le constater, la meilleure matrice de confusion est celle de la figure 3.7 sur le corpus 3, je tiens a preciser que ces matrices ont été réalisées sur des morceaux de 512 tokens. Sur la matrice de confusion du corpus 3 pour l'époque "Avant 1830" on remarque qu'il n'y a que 98% de vrai positifs et plus de 1% de faux négatifs, ce qui est normal car il n'y a que 36 textes pour cette époque comparé à la derniere époque ou il y avait 310 textes.

La matrice de confusion du corpus 2 n'est pas mauvaise mais présente un pourcentage élévé de faux négatifs pour certaines époques.

Quant à la matrice de confusion du corpus 1, comme l'accuracy l'indique on voit très bien qu'il faut plus de textes pour chaque époque.

Donc en regardant ces matrices, on peut conclure que la quantité de textes et leur répartitions sur les époques est importante pour avoir des bons résultats.

Schéma Descriptif De Notre Travail



Chapitre 4

Conclusion

En conclusion, la datation des textes français par la méthode de deep learning en utilisant le modèle de transformers DistilCamemBERT a montré une sérieuse efficacité.

Les résultats que nous avons obtenus peuvent illustrer cette efficacité, surtout en les comparants avec d'autres méthodes, par exemples celles mentionnées dans l'état de l'art.

Cette méthode est influencée par plusieurs paramètres, tels que le prétraitement des textes (nettoyage, tokénisation, encodage) et la taille de la base de données.

De point de vue scientifique, aucune recherche peut s'arrêter, nous pouvons toujours l'améliorer et la perfectionner au cours du temps. Et cette règle s'applique sur notre travail.

Nous devons toujours utiliser les résultats et les modèles de recherche dans le but d'en tirer des bénéfices pour l'ensemble de la communauté.

Quant à notre modèle, on peut l'intégrer dans un site web pour permettre aux utilisateurs de dater des textes.

Références

[1] Shikhar VASHISHTH, Shib Sankar DASGUPTA, Swayambhu Nath RAY et Partha TALUKDAR. « Dating documents using graph convolution networks ». In : *arXiv preprint arXiv :1902.00175* (2019) (page 2).

[2] Swayambhu Nath RAY, Shib Sankar DASGUPTA et Partha TALUKDAR. « Ad3 : Attentive deep document dater ». In : *arXiv preprint arXiv :1902.02161* (2019) (page 2).

[3] Dimitrios KOTSAKOS, Theodoros LAPPAS, Dimitrios KOTZIAS, Dimitrios GUNOPULOS, Nattiya KANHABUA et Kjetil NØRVÅG. « A burstiness-aware approach for document dating ». In : *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 2014, p. 1003-1006 (page 3).

[4] Nattiya KANHABUA et Kjetil NØRVÅG. « Using Temporal Language Models for Document Dating. » In : *ECML/PKDD (2)*. 2009, p. 738-741 (page 3).

[5] Amine ABDAOUI, Camille PRADEL et Grégoire SIGEL. « Load what you need : Smaller versions of multilingual bert ». In : *arXiv preprint arXiv :2010.05609* (2020) (page 4).

[6] https://www.journaldunet.fr/web-tech/guide-de-l-intelligence-artificielle/1508983-transformer-deep-learning/ (page 5).

[7] https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0 (page 7).

[8] https://huggingface.co/docs/transformers/main_classes/trainer (page 9).