Classification d'entité médicales à partir du langage naturel

Introduction

L'exercice de classification d'entités médicales à partir du langage naturel consiste à reconnaitre automatiquement, à partir de données textuelles, quelles pathologies celles-ci contiennent, sous un format standardise.

Cette problématique est omniprésente dans le domaine médical, tout particulièrement en milieu hospitalier, ou des codeurs humains passent de longues heures à lire et annoter manuellement des comptes rendu médicaux pour nourrir des bases de données (à des visées de facturation, typiquement).

Il vous dans le cadre de ce projet demande d'utiliser les notions d'apprentissage profond que vous avez acquises au cours de vos heures d'enseignement pour construire l'algorithme de classification d'entité médicales le plus performant possible, à partir d'un jeu de données fourni.

Description du jeu de données

Le jeu de données fourni est constitué de 3 fichiers csv contenant des données textuelles médicales. Chaque ligne contient une phrase en français et sa pathologie médicale correspondante exprimée sous format CIM-10, comme suit :

RawText	ICD-10
Thrombose veineuse profonde cuisse gauche	I802
Hémiplégie post-traumatique	S099

Masculinisation avec hyperplasie surrénale	E250
Hyperammoniémie cérébrale	E722

Il vous est demande dans ce projet de créer un modèle qui, à partir de la colonne RawText de votre csv, apprenne à prédire la colonne ICD-10.

Les 3 fichiers csv mis à votre disposition correspondent à vos jeux de données d'entrainement, de validation et de test. Ils contiennent 180k observations pour le jeu de données d'entrainement, et 30k observations pour les jeux de données de validation et de test. Vous êtes libre d'effectuer tout prétraitement que vous souhaitez sur la colonne RawText, tant que vous renseignez précisément vos actions dans votre compte rendu.

La classification CIM-10 (pour classification internationale des maladies, version 10), est une classification hiérarchique médicale développée par l'OMS est utilisée intensément à l'international pour nourrir des bases de données médicolégales. Elle est constituée d'un peu plus de 10000 codes distincts composes au maximum de 4 caractères et présente un aspect hiérarchique, au sens où les codes partageant le même préfixe (ie les mêmes premiers, deuxièmes, ou troisième caractères) sont typiquement associés à des pathologies qui présentent des similarités, comme par exemple :

- Tous les codes commençant par la lettre C, par exemple, sont associées a des pathologies de cancer
- Tous les codes commençant par le préfixe E11 sont associes à des variantes de diabète sucre de type 2

A partir de cette remarque, il peut semblant particulièrement intéressant d'exploiter le caractère séquentiel et hiérarchique de cette classification dans l'élaboration du modèle.

Exercice 1 : Description du jeu de données

Commencez par fournir une description du jeu de données. Vous répondrez entre autre aux questions suivantes :

- Combien de mots différents sont présents dans vos données textuelles ?
- Combien de codes CIM différents sont-ils présents dans le jeu de données ?

- Le jeu de données est-il équilibré ? (Au sens de « y a-t-il autant d'exemples de chaque code CIM dans les jeux de données ? »)
- Si oui, est ce que cela doit influencer votre choix de métriques ?

N'hésitez pas à fournir des visualisations afin d'illustrer vos propos.

Exercice 2 : Classification du premier caractère du code CIM-10 à partir du langage naturel

Dans cette partie, vous commencerez à vous familiariser avec l'exercice de reconnaissance d'entité médicales avec une version simplifiée du problème. Construisez un jeu de données ou votre label à prédire n'est pas le code CIM correspondant en entier, mais seulement son premier caractère (réduisant le nombre de classes à prédire d'une dizaine de milliers à 26, puisqu'un code CIM commence par une lettre de l'alphabet).

Cette exercice peut être considéré comme un simple exercice de classification de séquences, comme il a déjà été vu en cours.

Vous constaterez en regardant leur description dans l'API tensorflow que les fonctions et objets suivant vous seront probablement utiles :

- tf.keras.Sequential
- tf.keras.layers.TextVectorisation (attention à l'initialiser correctement avant de l'utiliser dans tf.keras.Sequential. Les tutoriels indiqués sur la page de description de cette fonction peuvent être utiles)
- tf.keras.layers.Embedding
- tf.keras.layers.LSTM
- tf.keras.layers.GlobalAveragePooling1D
- tf.keras.layers.Dense

Note : Pour vous donner une idée, des mesures d'accuracy (évaluées sur le jeu d'entrainement) de l'ordre de 80% sont (relativement) facilement atteignables pour cet exercice.

Exercice 3 : Classification du code CIM-10 entier à partir du langage naturel

Dans cette partie, vous présenterez une première solution de classification d'entité CIM-10 à partir du langage naturel, en considérant cette fois ci non pas le premier caractère mais le code entier comme une variable catégorielle à expliquer. Votre code ne devrait pas drastiquement changer entre cette partie et la précédente, aussi n'hésitez pas à vous montrer un peu créatif au niveau de l'architecture de votre modèle. Les fonctions suivantes devraient notamment pouvoir vous intéresser :

- tf.keras.layers.Conv1D
- tf.keras.layers.Bidirectional

Quelles performances obtenez-vous ? Sont-elles comparables à celles obtenues dans l'exercice précédent ? Est-ce que cela était prévisible ?

Il est assez courant dans la littérature académique sur le sujet, de ne pas rapporter seulement les métriques à l'échelle de la correspondance entre valeur vraie et prédiction du code en entier, mais de s'intéresser aussi aux premiers caractères.

Recalculez vos métriques à l'échelle du premier, des deux premiers, et des trois premiers caractères. Que pouvez-vous dire sur l'évolution des performances en fonction du nombre de caractères inclus ? Comment se présentent les performances obtenues mesurées au niveau du premier caractère, comparées à celles obtenues dans l'exercice précédent ?

Note : Pour vous donner une idée, des mesures d'accuracy (évaluées sur le jeu d'entrainement) de l'ordre de 60% sont (relativement) facilement atteignables pour cet exercice. Si vous obtenez moins, vous avez probablement un problème avec votre code (ou vous ne laissez pas tourner le modèle suffisamment longtemps. Ou un autre milliard des charmants petits détails cruciaux a la correcte implémentation d'un réseau de neurone vous a probablement échappé).

Exercice 4 : Modèle seq2seq et prédiction gloutonne du code CIM-10 à partir du langage naturel

Dans cette partie, vous exploiterez directement le caractère hiérarchique de la CIM10 pour prédire les codes correspondant au langage naturel, en utilisant un modèle seq2seq comme expose en cours. Vous considèrerez la variable RawText comme une phrase en langage source, et le code CIM (considéré comme une séquence de caractère de la manière suivante ['C', '2', '5', '4']) comme une phrase d'au plus 4 mots, en langage cible.

Partie 1 Ajustement du modèle par apprentissage forcé

Dans cette première partie, vous vous concentrerez uniquement sur l'ajustement du modèle, et non sur la prédiction (que vous verrez en deuxième partie).

Plusieurs sujets sont à traiter dans cette partie :

- Convertir les labels de la colonne ICD-10 en séquences de lettre (de longueur variable, étant donne que certains code CIM ne disposent que de 3 caractères). D'autres étapes de prétraitement sont-elles nécessaires dans cette partie ? Vous aurez probablement besoin de la fonction tf.pad, et de la méthode padded_batch de l'objet tf.data.Dataset pour cette étape. N'hésitez pas à vous inspirer des exemples de codes vu en cours pour cette partie !
- De manière à gérer les séquences de longueur variable, il est de coutume dans tensorflow d'utiliser du padding avec un token spécial réservé à cet effet. Pensez-vous qu'il soit pertinent, lors de l'ajustement du modèle, que les prédictions relatives au padding soient considérées dans la fonction objectif ? N'hésitez pas à vous inspirer pour cette partie de l'exemple de code suivant : https://github.com/tensorflow/models/blob/master/official/nlp/transformer/metrics.py
- De la même manière, pensez-vous que les métriques que vous avez choisi d'utiliser doivent considérer le padding dans leur estimation ?
- De manière à ne pas avoir à ajuster un modèle à chaque fois que l'on désire obtenir de nouvelles prédictions, il est important de savoir comment exporter et importer les poids d'un modèle préalablement ajusté dans tensorflow. Les méthodes « save_weights » et « load_weights » de l'objet keras « Model » vous seront probablement utiles a cet effet. N'hésitez pas à aller regarder leurs définitions sur la page de description de l'objet « Model » dans l'API tensorflow

Note : Pour vous donner une idée, des mesures d'accuracy (évaluées sur le jeu d'entrainement) de l'ordre de plus de 70% sont (relativement) facilement atteignables pour cet exercice. Si vous obtenez moins, vous avez probablement un problème avec votre code (ou vous ne laissez pas tourner le modèle suffisamment longtemps. Ou un autre milliard des charmants petits détails cruciaux a la correcte implémentation d'un réseau de neurone vous a probablement échappé).

Partie 2 Prédictions gloutonnes à partir d'un modèle ajusté

Maintenant que vous disposez d'un modèle correctement ajuste, l'essentiel reste encore à faire. En effet, un modèle prédictif n'a que très peu d'intérêt s'il lui est impossible de prédire ! Vous vous occuperez donc dans cette partie de créer une fonction de prédiction gloutonne utilisant le modèle que vous avez déjà ajusté.

Pour rappel, le model que vous avez ajusté est habitue à recevoir la séquence correcte de caractère de code cim correspondant, à laquelle on a concaténé au début un token « <START> ». En cas de prédictions, vous ne connaissez pas cette dite séquence (puisque vous la cherchez), mais vous connaissez son premier élément. Votre code devrait donc ressembler à quelque chose comme :

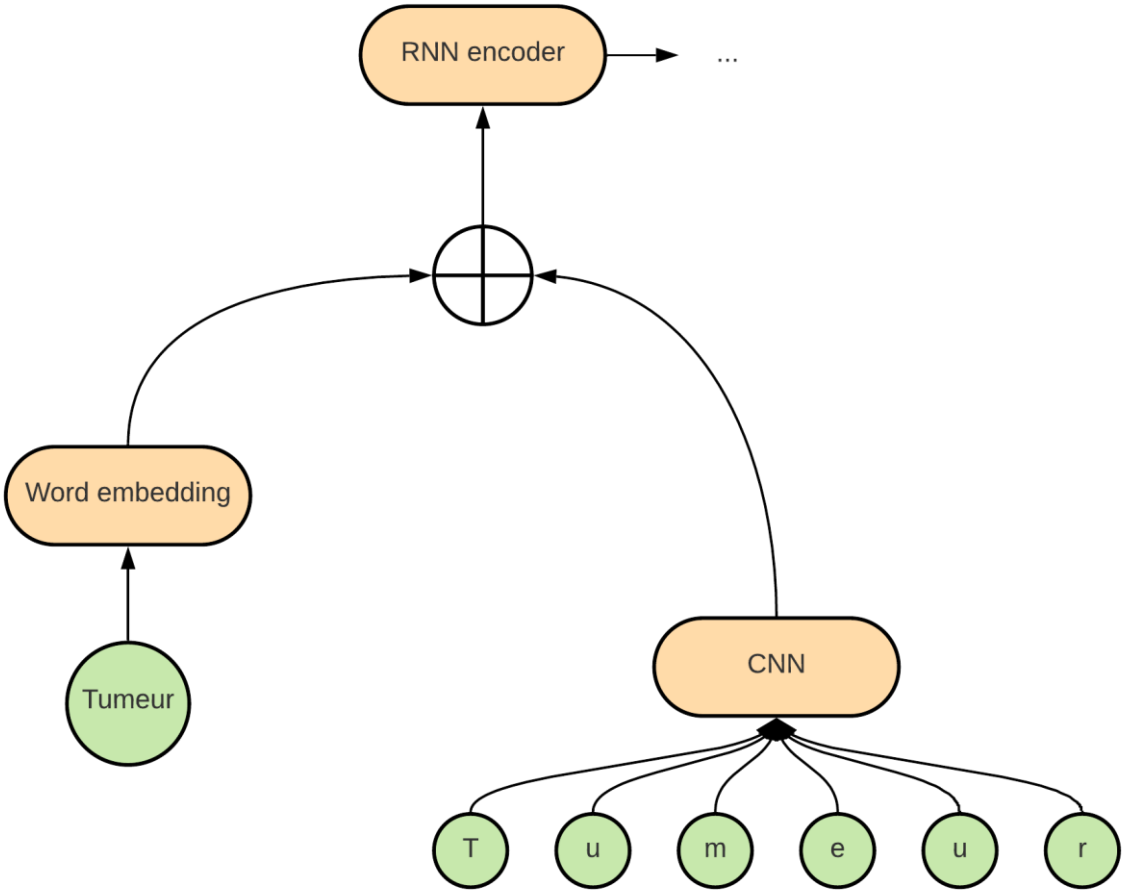
- Etape 1 : Appliquer model.predict() aux séquences en français et a des séquences de labels constituées exclusivement du token <START>. Utiliser la sortie du modèle pour trouver le premier caractère de chaque code correspondant
- Etape 2 : Concaténer le token <START> a ce premier caractère, puis réappliquer le modèle sur ces nouvelles séquences de labels.
- Etape 3 : Réitérer ce processus jusqu'à obtenir des codes CIM entiers

Maintenant que vous êtes capables de prédire avec votre modèle, vous pouvez enfin en évaluer ses performances sur le jeu de données de validation. Comment se comporte-t-il, compare au modèle de classification classique que vous avez ajusté dans l'exercice 3 ?

Exercice 5 : Pour aller plus loin

Félicitations ! Vous venez d'écrire votre premier modèle de traduction neuronale. Quand on réalise que ce genre de technologies n'existait pas il y a 10 ans, c'est déjà un bel exploit en soi. Cependant, vous savez pertinemment que cette architecture est tout de même assez archaïque, et que de multiples variations sur cette idée existent pour améliorer les performances de votre système. Vous êtes dans cette partie libre de choisir quelles directions explorer pour améliorer votre modèle :

1. Aucun preprocessing (si ce n'est un effacement des ponctuations et une méthode lower()) n'a été appliquée a votre jeu de données avant ajustement. Peut-être qu'un preprocessing un peu plus intelligent (comme une lemmatisation par exemple) vous permettrait de donner un coup de pouce à votre modèle !
2. Des méthodes de tokenizations existent pour profiter du caractère composite (préfixe et suffixe) des mots. Est-ce que ces méthodes entrainent un gain de performance pour le modèle ? Vous regarderez notamment la méthode de Byte Pair Encoding, dont une implémentation tensorflow est trouvable au lien suivant : https://github.com/tensorflow/models/blob/master/official/nlp/transformer/utils/tokenizer.py
3. Comme alternative aux méthodes de tokenizations, certains praticiens il y a de ça quelques années préféraient fournir au modèle les phrase t'entrées décomposées à la fois en séquences de mots et en séquences de caractères, typiquement en injectant dans le LSTM encodeur une combinaison additive du word embedding correspondant au mot (que vous savez déjà construire) et de la sortie d'un réseau à convolution applique à la séquence des caractères composant le mot comme vous pouvez l'observer dans le schéma suivant :



4. Comme vous le savez, les réseaux de neurones récurrents sont récemment tombés dans l'oubli, au profit des convolutions causales dilatées ou des modèles de type Transformer. En vue du faible nombre d'observations disponibles dans votre jeu de données, le Transformer n'est pas forcément la meilleure option dans votre situation, mais les convolutions causales dilatées peuvent en revanche être une piste intéressante d'amélioration. Vous pourrez probablement bénéficier de regarder dans la documentation de tensorflow les différentes valeurs du kward « padding » que peut accepter l'objet « Conv1D » de Keras.
5. Même si le Transformer en lui-même est probablement trop demandeur en données, les modèles RNN encodeurs décodeurs avec attention constituaient une remarquable innovation qui a conduit a de notables améliorations de performances en traduction neuronale. Vous pourrez essayer d'en appliquer un a votre projet, en vous appuyant par exemple sur le tutoriel suivant :

https://www.tensorflow.org/text/tutorials/nmt_with_attention

Ca peut également être l'occasion de montrer des matrices d'attentions apprises par le modèle, afin d'observer sur quels mots de la phrase en français il se base pour prédire chacun des caractères.

6. Encore une fois, le Transformer n'est pas nécessairement applicable sur ce jeu de données. En revanche, les modèles BERT non supervisés ont typiquement été construits pour des situations pareilles ! Vous trouverez facilement sur le net des modèles BERT spécifiques au français (comme CamemBERT et FlauBERT). Essayez de les adapter à votre jeu de données pour obtenir des meilleures performances !

Vous choisirez parmi toutes ces options au moins deux points à développer. La seule combinaison interdite est 1-2 (trop facile de travailler seulement sur du preprocessing. Le but du jeu est tout de même de coder du réseau de neurone ici) et comparerez vos résultats à ceux obtenus dans les questions précédentes. Notez que ces points peuvent également être combinés de manière astucieuse (par exemple, 4 et 5 en implémentant un modèle encoder-décodeur à attention ou encodeur et décodeurs sont des réseaux à convolutions causales dilatées, comme il a été proposé par facebook https://arxiv.org/abs/1705.03122).

Bon courage !