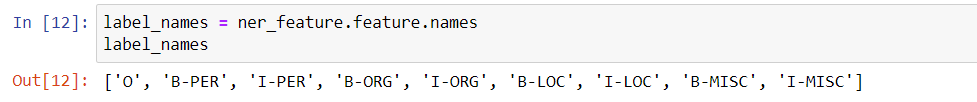
**Projet 1 NLP**

**a)** Le training set continent 14041 phrases.

La validation set représente 23.1% de la training set et la test set représente 24.6% de la training set.

**b)** Les classes sont:



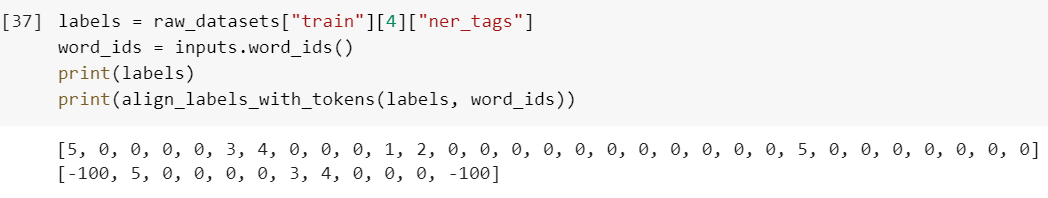
**c)** Voici 5 phrases (15ème jusqu’à la 19ème) avec leurs ner\_tags sous forme de nombre et sous forme de texte :

A picture containing calendar

Description automatically generated

**d)** Les tokens spéciaux reçoivent une étiquette de -100. En effet, par défaut, -100 est un indice qui est ignoré dans la fonction de perte que nous allons utiliser (cross entropy).

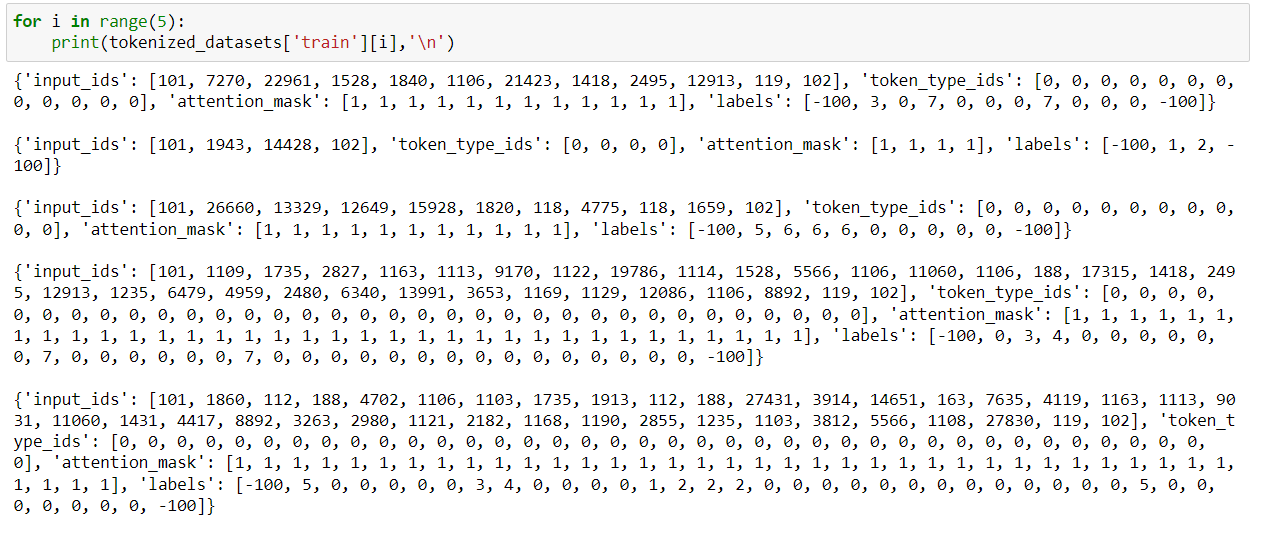
**e)** Dans cette portion de code, label % 2 == 1 vérifie si la valeur de label est impaire ou non. Si c'est le cas, elle est incrémentée de 1. Il semble que ce code modifie la valeur du label en fonction de s'il s'agit d'un label "B" ou d'un label "I". Les labels sont utilisés pour marquer le début et l'intérieur des entités nommées dans le texte. "B" signifie "beginning" (début) et "I" signifie "inside" (intérieur). Par exemple, si un mot est le premier mot d'une entité nommée, il serait étiqueté avec "B-XXX", où XXX est le type d'entité nommée (par exemple "B-PERSON" pour un nom de personne). Si un mot est à l'intérieur d'une entité nommée mais n'est pas le premier mot, il serait étiqueté avec "I-XXX". Par conséquent, ce code vérifie si le label est un label "B" (nombre impair) et, s'il l'est, le change en un label "I" (nombre pair) en l'incrémentant de 1. Cela serait utile si la liste labels et la liste word\_ids ne sont pas déjà alignées et si la fonction doit ajuster les labels pour correspondre aux bons mots.

**f)** Voici l’output qu’on a eu :

Pas de 2 qui se répète.

**g)** raw\_datasets est de type datasets.dataset\_dict.DatasetDict.

La fonction map() permet d'accélérer le traitement et d'appliquer une fonction de traitement à chaque exemple de la dataset, indépendamment ou par lots.



Les « input\_ids » sont les identifiants uniques des tokens dans une phrase.

« token\_type\_ids » sont représentés sous la forme d'un masque binaire identifiant les deux types de séquence dans le modèle.

« attention\_mask » est utilisé pour regrouper la séquence d'entrée et indiquer si le token doit être pris en compte par notre modèle ou non. Un token avec une valeur de attention\_mask de 0 signifie que le token sera ignoré et 1 signifie que les tokens sont importants et seront pris pour un traitement ultérieur.

« labels » est une liste dont chaque élément représente l’entité (son numéro) du token.

On n’a pas encore fait le padding pour les inputs, c’est ce qui manque pour faire le training.

**Méthode Trainer:**

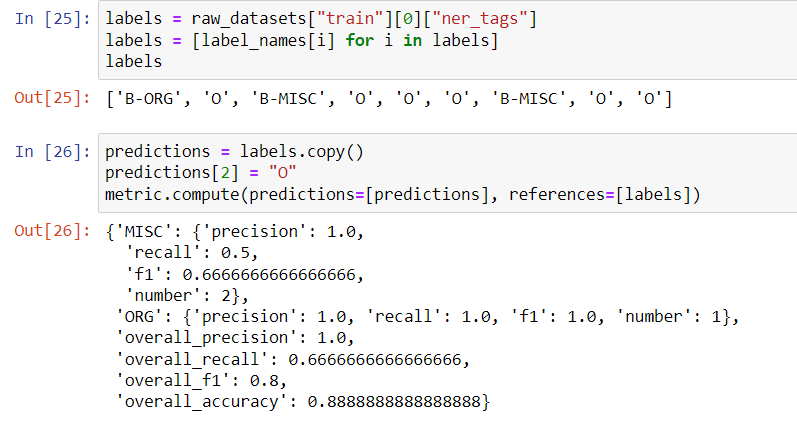
**h)** On utilise DataCollatorForTokenClassification parce qu’on travaille sur une tâche de classification de tokens et que les données d'entrée sont déjà sous la forme de séquences de texte tokenisées avec des étiquettes.

Si on travaille sur une tâche différente ou si nos données d'entrée ne sont pas dans ce format spécifique mais peuvent toujours être remplies jusqu'à la même longueur, on utiliserait DataCollatorWithPadding.

**i)** Seqeval produit des scores d'étiquetage ainsi que ses statistiques suffisantes à partir d'une source par rapport à une ou plusieurs références.

predictions : une liste de listes d'étiquettes prédites, c'est-à-dire les cibles estimées telles que retournées par un tagger.

references : une liste de listes d'étiquettes de référence, c'est-à-dire les ground truth (target).



Overall :

* Accuracy : la précision moyenne, sur une échelle comprise entre 0,0 et 1,0.
* Précision : la précision moyenne, sur une échelle comprise entre 0,0 et 1,0.
* Recall : le recall moyen, sur une échelle comprise entre 0,0 et 1,0.
* F1 : le score F1 moyen, qui est la moyenne harmonique de la précision et du recall. Il a également une échelle de 0,0 à 1,0.

Par type (par exemple, MISC, ORG, ...) :

* Precision : la précision moyenne, sur une échelle entre 0.0 et 1.0.
* Recall : le recall moyen, sur une échelle entre 0.0 et 1.0.
* F1 : le score moyen de F1, sur une échelle comprise entre 0,0 et 1,0.

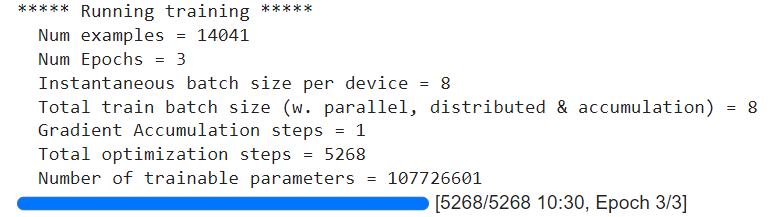
**j)** En apprentissage automatique, les "logits" et les "probabilités" sont deux manières de représenter les prédictions faites par un modèle.

Les "logits" sont les sorties brutes d'un modèle avant qu'elles n'aient été transformées en probabilités. On peut les considérer comme des probabilités non mises à l'échelle, avec des valeurs qui peuvent aller de l'infini négatif à l'infini positif.

Les "probabilités", en revanche, sont les sorties d'un modèle après avoir été transformées à partir de logits. Ce sont des valeurs entre 0 et 1 qui représentent la probabilité d'un résultat particulier. Pour transformer des logits en probabilités, nous appliquons la fonction « softmax » aux logits.

Donc, quand l'expression dit "as usual, the logits and the probabilities are in the same order, we don’t need to apply the softmax," cela signifie que les logits et les probabilités sont déjà dans l'ordre correct et qu'il n'est pas nécessaire de les transformer avec la fonction softmax.

**k)** Le training avec le Trainer est fait sur Google Colab avec succès.



**l)** Non, parce que c’est à Google Colab de distribuer le calcul sous plusieurs GPU.

Si nous augmentons la taille du batch, alors TensorFlow a le plus grand taux d'utilisation des GPU et PyTorch le plus faible. Avec l'augmentation de la taille du batch, la consommation des GPU augmente de façon spectaculaire. Donc c’est l’application cloud-based qui fait le meilleur choix de batch size tout seule.

D’après l’affichage du training sur Colab :

Instantaneous batch size per device = 8

Total train batch size (w. parallel, distributed & accumulation) = 8

Cela veut dire que Google a alloué des GPUs, et chacune d’elles a un batch size de 8.

**m)** AutoModelForTokenClassification est une classe de la bibliothèque transformers qui peut être utilisée pour entraîner un modèle de classification de tokens de manière automatisée. Elle utilise le "transfer learning" pour pré-entraîner un modèle sur un grand corpus de données et le fine-tuner ensuite sur des données spécifiques à la tâche de classification de tokens souhaitée. Cela permet d'obtenir de meilleures performances que si l'on entraînait le modèle de manière "from scratch" sur des données de tailles limitées.

En utilisant AutoModelForTokenClassification, on n’a pas besoin de choisir manuellement le modèle de pré-entraînement à utiliser ni les paramètres de fine-tuning. La classe s'occupe de tout cela automatiquement en utilisant un algorithme de recherche automatisée. On peut simplement lui fournir nos données d'entraînement et laisser la classe s'occuper du reste. Cela peut nous faire gagner du temps et nous assurer qu’on utilise le modèle le plus approprié pour notre tâche de classification de tokens.

**n)** notebook\_login() lance un widget dans notre notebook à partir duquel on pourra entrer nos informations d'identification pour Hugging Face. Cela illustre un concept de connexion à notre Colaboratory notebook avec un "token".

Un "token" est généralement un jeton unique qui peut être utilisé pour s'authentifier auprès d'un système ou d'une application plutôt qu'un nom d'utilisateur et un mot de passe, ce qui peut être pratique et plus sécurisé que de stocker un nom d'utilisateur et un mot de passe en clair.

**o)** Les métriques obtenues sont affichées après le training.

Table

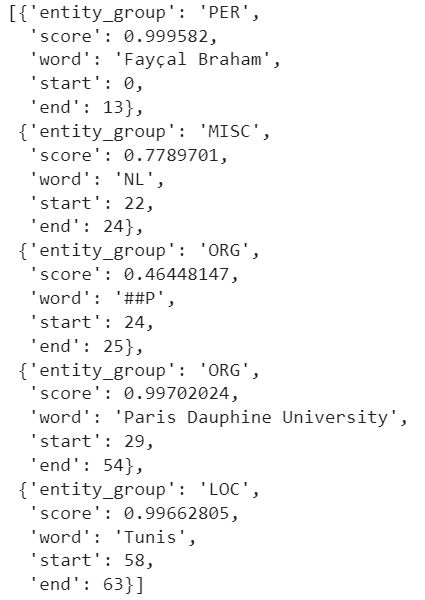
Description automatically generated with medium confidence

**p)** Voici l’output de la prédiction/classification des tokens de cette phrase en utilisant la méthode du Trainer : « My name is Sylvain and I work at Hugging Face in Brooklyn. » :

Text

Description automatically generated

**q)** Voici l’output de la prédiction/classification des tokens de cette phrase en utilisant la méthode du Trainer : « Fayçal Braham teaches NLP at Paris Dauphine University at Tunis » :



Le modèle a bien detecté « Fayçal Braham » comme une Personne (99.9% de probabilité), « Paris Dauphine University » comme une Organisation (99.7% de probabilité) et « Tunis » comme une Localisation (99.6% de probabilité). Mais il a commis une erreur de tokenization. « NLP » devrait être un seul token, alors qu’il l’a séparé en 2 : « NL » et « ##P ». Néanmoins, puisque le modèle de classification de tokens est bien entraîné, il a bien classé « NL » comme MISC quand même (au lieu de classer le token « NLP » comme MISC).

**Méthode boucle spécifique:**

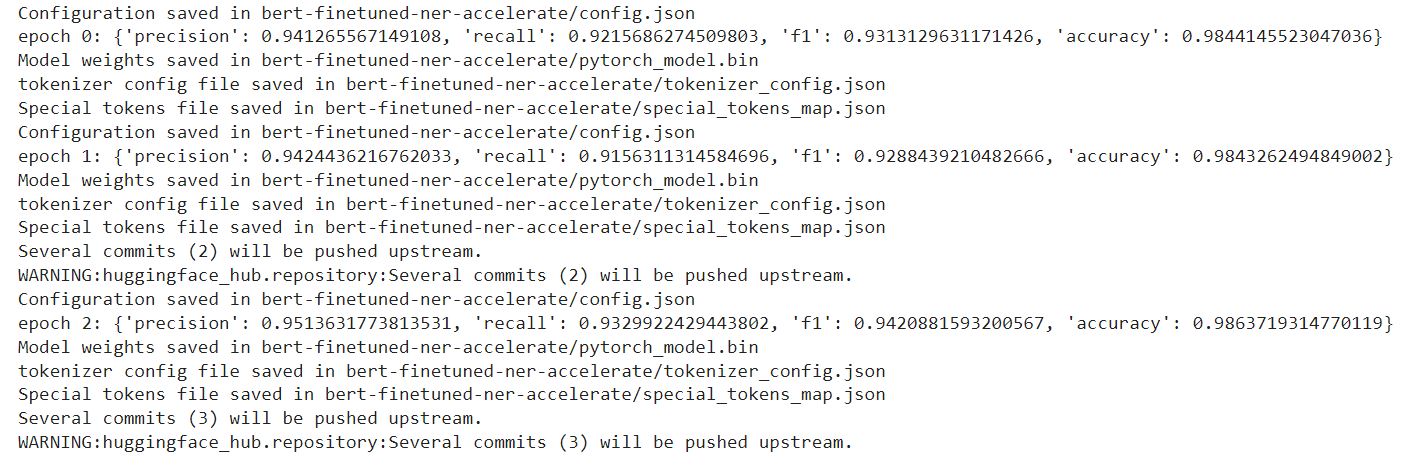
**r)** Le Dataloader de PyTorch est une classe qui permet de charger des données à partir d'un dataset et de les préparer pour l'apprentissage avec un réseau de neurones. Il effectue plusieurs fonctions utiles, telles que le mélange des données à chaque époque pour éviter le overfitting et le groupement des données en batchs pour faciliter l'apprentissage.

**s)** Pour exécuter le training, on peut réduire la taille des batchs, c’est-à-dire réduire la quantité de mémoire nécessaire pour stocker les gradients et les tenseurs intermédiaires.

**t)** La classe Accelerator () détecte automatiquement le type de configuration distribuée et initialise tous les composants nécessaires à la formation. On n’a pas besoin de placer explicitement notre modèle sur un appareil.

**u)** Le training avec cette méthode de boucle spécifique et Dataloader est fait sur Google Colab avec succès.



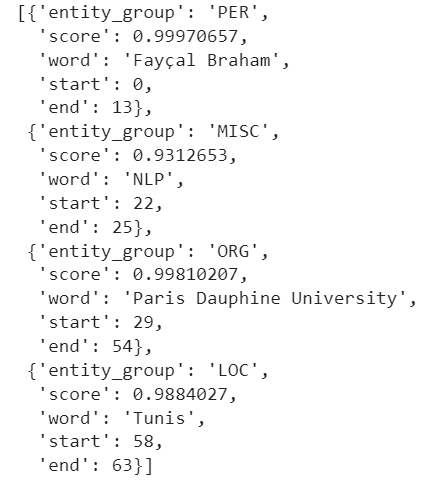
**v)** 

**w)** Voici l’output de la prédiction/classification des tokens de cette phrase en utilisant la méthode de boucle spécifique et Dataloader : « My name is Sylvain and I work at Hugging Face in Brooklyn. » :

Text

Description automatically generated

**x)** Voici l’output de la prédiction/classification des tokens de cette phrase en utilisant la méthode de boucle spécifique et Dataloader : « Fayçal Braham teaches NLP at Paris Dauphine University at Tunis » :



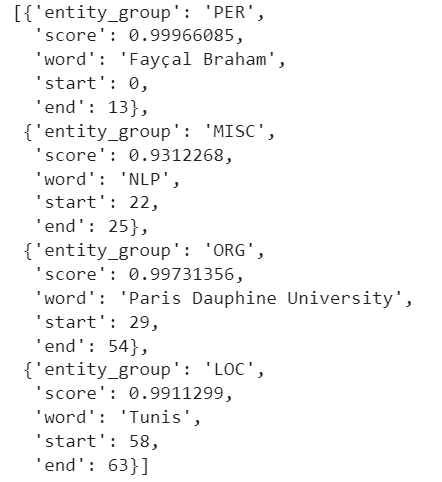
Le modèle a bien detecté « Fayçal Braham » comme une Personne (99.9% de probabilité), « Paris Dauphine University » comme une Organisation (99.8% de probabilité) et « Tunis » comme une Localisation (98.8% de probabilité). Cette fois-là, le modèle a bien tokenizé la phrase en considérant « NLP » comme un seul token, et en lui donnant la classe MISC avec 93% de probabilité, ce qui est un bon résultat.

**Pipeline :**

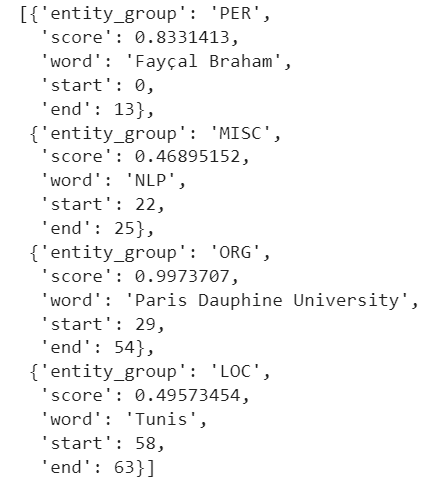
**y)** « aggregation\_strategy » définit la stratégie à utiliser pour agréger les prédictions de la classification de tokens sur plusieurs segments. La valeur "simple" indique que les prédictions sur chaque segment doivent être concaténées pour former une seule liste de prédictions. Cette stratégie est souvent utilisée lorsque les segments ont la même longueur et que l'ordre des tokens dans chaque segment est important.

On peut utiliser d’autres aggregation strategies comme : ‘’first’’, ‘’average’’, et ‘’max’’.

Avec aggregation\_strategy = ‘’first’’:



Avec aggregation\_strategy = ‘‘average’’ :



Avec aggregation\_strategy = ‘’max’’ :

Text

Description automatically generated

« first » et « max » donnent presque le même résultat que « simple », alors que « average » donne un résultat acceptable mais avec une plus basse probabilité (ce la veut dire que le modèle n’est pas très sûr).

**z)** La méthode « Trainer » est utilisée pour configurer et lancer l'entraînement d'un modèle de transformateur, tandis que le « Dataloader » est utilisé pour charger et préparer les données d'entraînement ou de test pour l'entraînement d'un modèle.

Dans notre cas, la méthode du « Dataloader » a donné de meilleurs résultats. Son modèle a fait une bonne tokenization, et une bonne classification de tokens avec des probabilités de 99% (et 93% pour le token « NLP »). C’est-à-dire, le modèle est très certain de ce qu’il fait.