Utilisation de l'outil FastText, pour améliorer la recherche sur un site web

Introduction:

En janvier 2024, au cours de ma deuxième année en BTS SIO, j'ai effectué un stage d'une durée d'un mois au sein du groupe de presse Jeune Afrique Media Group. Au cours des premières semaines de mon stage, on m'a confié la tâche de reprendre le travail exploratoire entrepris par un développeur, axé sur la recherche avec ***FastText,*** sous Python.

Qu'est ce que FastText ?

FastText est une bibliothèque open source développée par Facebook pour le traitement du langage naturel (NLP) et l'apprentissage automatique de ce langage. Elle est principalement utilisée pour la classification de texte et la création de modèles de représentation de mots. FastText se distingue par sa capacité à entraîner des modèles rapidement, même sur de grands ensembles de données.

NLP et modèles de représentation de mots:

Le *Natural Language Processing* (ou *Traitement de Langage Naturel* en français) est une branche de l'intelligence artificielle, axée sur l'interaction entre les ordinateurs et le langage humain. L'objectif de ce traitement est de permettre aux machines d'apprendre, de comprendre et de pouvoir générer un langage similaire à celui de l'Homme, et de manière compréhensible par l'Homme.

Ces modèles capturent ***les nuances sémantiques***(variations subtiles de sens ou de signification qui existe entre des termes dû au contexte, connotations) permettant d'utiliser ***les subtilités morphologiques des mots*** (changements liés à sa construction grammaticale), permettant d'enchérir la compréhension des relations entre des mots, ici on parle "**d'embeddings**".

FastText déploie des réseaux de neurones,afin de générer des embeddings de mots. Ces représentations numériques positionnent les mots dans un espace multidimensionnel, capturant ainsi leurs **nuances** et **similarités** sémantiques et morphologiques. Cette approche permet **d'enrichir** la compréhension du langage par le modèle, révélant des **nuances subtiles** essentielles dans la **compréhension** de contextes linguistiques ardus. On parle alors de : " ***classification de mots*** "

Comment utiliser fasttext?: Installation

Il suffit d'installer fasttext en suivant les indications de la documentation officielle : https://fasttext.cc/docs/en/support.html

Ou simplement, créer une copie locale du dépôt GitHub de fasttext:

```
stephane@OptiPlex-7070:~/Téléchargements/fasttextInstall$  git clone https://github.com/facebookresearch/fastText.git
Clonage dans 'fastText'...
remote: Enumerating objects: 3986, done.
remote: Counting objects: 100% (1045/1045), done.
remote: Compressing objects: 100% (182/182), done.
remote: Total 3986 (delta 920), reused 882 (delta 859), pack-reused 2941
Réception d'objets: 100% (3986/3986), 8.29 Mio | 13.05 Mio/s, fait.
Résolution des deltas: 100% (2527/2527), fait.
```

Il suffit ensuite d'entrer dans le dossier créer, puis de lancer la commande make (pour automatiser le processus de compilation,avec un répertoire contenant un fichier appelé "Makefile", cette commande exécute les instructions définies dans ce fichier):

```
stephane@OptiPlex-7070:~/Téléchargements/fasttextInstall$ cd fastText/
stephane@OptiPlex-7070:~/Téléchargements/fasttextInstall/fastText$ make
c++ -pthread -std=c++17 -march=native -O3 -funroll-loops -DNDEBUG -c src/args.cc
c++ -pthread -std=c++17 -march=native -O3 -funroll-loops -DNDEBUG -c src/autotune.cc
c++ -pthread -std=c++17 -march=native -O3 -funroll-loops -DNDEBUG -c src/matrix.cc
c++ -pthread -std=c++17 -march=native -O3 -funroll-loops -DNDEBUG -c src/dictionary.cc
```

Puis, il suffit d'installer le module de Python pour fasttext:

```
stephane@OptiPlex-7070:~/Téléchargements/fasttextInstall/fastText$ sudo python3 setup.py install
/usr/lib/python3/dist-packages/setuptools/dist.py:723: UserWarning: Usage of dash-separated 'description-fi
nstead
  warnings.warn(
running install
/usr/lib/python3/dist-packages/setuptools/command/install.py:34: SetuptoolsDeprecationWarning: setup.py ins
  warnings.warn(
/usr/lib/python3/dist-packages/setuptools/command/easy_install.py:158: EasyInstallDeprecationWarning: easy_
  warnings.warn(
```

A noter qu'il faudra au préalable installer Python3 avec :" ***sudo apt install python3*** " L'installation est maintenant terminée !

Création et utilisation d'un modèle:

Le but de cette classification de mots est d'associer à un document (articles de journaux, leurs résumés et leurs titres) à une ou plusieurs catégories, et d'ainsi créer des "classificateurs" regroupant l'entièreté des données (catégories + contenu). Ainsi, l'apprentissage de l'outil fasttext permet la création de ces classificateurs (contexte, connotation, …), mais leur création nécessite des ***données labellisées de manière précise***, ici une catégorie est signifiée par l'appellation ***label***.

Ainsi, avant de créer un modèle, il est essentiel de labelliser les données, et pour cela il suffit d'écrire une ligne, en commençant par les labels, puis par le contenu qui sera labellisé:

-On labellise les catégories en ajoutant le préfixe : "**__label__**", suivit de la catégorie, on répète ce format pour chaque label d'un document.

"" __label__Numéro1 __label__Numéro2 Contenu_du_Document ""

Maintenant que nos données sont en forme, il nous suffit de créer un modèle sur python, en utilisant le fichier contenant nos données:

```
1   import fasttext
2   model = fasttext.train_supervised(input="JA_labels.txt")
```

On importe la librairie fasttext, et ici le “**input**” indique le fichier contenant nos données labellisées.
En exécutant le code :

```
stephane@OptiPlex-7070:~/Bureau/elasticsearch_nlp$ /bin/python3 /home/stephane/Bureau/elasticsearch_nlp/src/pythonDocumentationFAstText.py
Read 0M words
Number of words:  32516
Number of labels: 1425
Progress: 100.0% words/sec/thread:   91290 lr:  0.000000 avg.loss: 11.705429 ETA:   0h 0m 0s
```

On obtient des informations sur notre modèle créé, avec le nombre de mots et le nombre de labels se trouvant dans le fichier.
On peut ensuite poser une question à fasttext en utilisant la méthode :
“**model.predict**”, suivie de notre question:

```
print(model.predict("Qui est le président de la Côte D'Ivoire?"))
```

```
Progress: 100.0% words/sec/thread:   88298 lr:  0.000000 avg.loss: 11.957835 ETA:   0h 0m 0s
(('__label__18',), array([0.35248792]))
```

Cette méthode prédit un label, ici le label 18 équivaut à la catégorie: “Afrique de L’Ouest”.

```
"18": "Afrique de l'Ouest",
```

La Côte d’Ivoire est bel et bien un pays d’Afrique de L’Ouest.

Paramètres:

Il est également possible d’ajouter des paramètres à notre modèle comme:

- ***k***: allows to define the number of labels we want in return,
- ***epoch*** : permet de choisir le nombre de fois où fasttext s'entraîne sur des données (de base à 5, on peut le monter pour permettre un meilleur apprentissage),
- ***lr***: le taux d’apprentissage, peut être réglé via ce paramètre, entre 0 et 1.0, où 0 signifie que le modèle n’apprend pas,
- ***wordNgrams***: ce paramètre permet d’apprendre des tokens par un nombre prédéfinis. Dans fasttext, un token est un mot, dans notre cas nous mettrons la valeur 2 à ce paramètre, ce qui signifie que fasttext apprendra par groupe de 2 tokens, 2 mots dans le cas de fasttext

La notion des paramètres est importante, car elle permet d’obtenir des résultats beaucoup plus concluant qu’avec le modèle initial (sans paramètre)

Autotune(Automatic Hyperparameter optimization):

L'automatisation optimale des paramètres, permet de trouver automatiquement les paramètres optimaux pour nos données labellisées. Mais l'utilisation d'Autotune nécessite un fichier de données labellisées dont ont garanti le contenu, qui jouera le rôle de "***autotuneValidationFile***".

```
model = fasttext.train_supervised(input="JA_labels.txt", autotuneValidationFile="JA_labels_autotune.txt")
```

On obtient ce résultat:

```
stephane@OptiPlex-7070:~/Bureau/elasticsearch_nlp$ /bin/python3 /home/stephane/Bureau/elasticsearch_nlp/src/pythonDocumentationFAstText.py
Progress: 100.0% Trials:   17 Best score:  0.112360 ETA:   0h 0m 0s
Training again with best arguments
Read 0M words
Number of words:  32516
Number of labels: 1425
Progress: 100.0% words/sec/thread:  180969 lr:  0.000000 avg.loss: 17.090221 ETA:   0h 0m 0s
```

Ici, le modèle a analysé 17 paramètres optimaux, et s'est exécuté avec les paramètres trouvés.
Il existe également une méthode '***test***', permettant de tester la validité de nos données labellisées, et affichant la précision du modèle, et le nombre de labels correctement prédit (recall) par le modèle:

```
Progress: 100.0% words/sec/thread:  165690 lr:  0.000000 avg.loss: 17.034775 ETA:   0h 0m 0s
(15, 0.2, 0.04054054054054054)
```

Dans notre test

- ➔ précision: 0.2 (pas très probant mais cela nous donne une idée que l'on peut encore améliorer notre modèle, ici il n'y a que 15 labels)
- ➔ recall:0.04

Tous les éléments essentiels à la bonne compréhension de notre utilisation de la librairie fasttext ont été évoqués.

Objectifs et Réalisations

Objectif:

- Reprendre le travail exploratoire sur la partie vectorisation de fasttext (**embeddings**) démarré par un développeur, et utiliser cette base de code pour travailler plus en profondeur la partie label, et autotune.
- Rendre l'outil fasttext viable pour une implémentation future sur le site web

Réalisation:

- Création d'un fichier pour stocker des fonctions permettant de mettre en forme les données initiales (au format JSOn), ainsi que de les exploiter

- Fichier de données labellisées sur tous les articles du journal Jeune Afrique (de 2010 à 2024)
- Utilisation de l'outil autotune + création à la main d'un fichier de données labellisées dont le contenu est garanti véridique (labels et contenu liés)
- Tests de nombreux modèles et de paramètres différents, et obtention d'un modèle potentiellement utilisable par le journal, mais non optimisé, ni entraîné davantage dû à un manque de ressources.

La mise en forme des données:

J'ai débuté ma partie dans ce projet avec un fichier JSON regroupant 5000 articles de Jeune Afrique que je devais exploiter. Mon premier obstacle dans ce projet était de pouvoir exploiter ces données, j'ai donc utiliser différentes fonctions de la librairie " ***json*** " pour rendre ces données exploitables :

- "***json.load***" : permet de convertir un fichier JSON en une liste Python.

```
def fichierJSON_vers_ListePython(cheminFichierJSON):
    with open(cheminFichierJSON, 'r', encoding='utf-8') as fichier_json:
        listePython = json.load(fichier_json)
    return listePython
```

- "***json.loads***" : convertit une chaîne de caractère JSON en un dictionnaire Python.

```
liste_python[indice]['tags'] = json.loads(liste_python[indice]['tags'])
```

Dans notre cas, elle transforme des chaînes de caractères où sont stockés des **labels**, en un **dictionnaire** de la forme ***"clé : valeur"*** ; où la valeur sera l'**intitulé du label**.

Maintenant que nous pouvons extraire nos données d'une liste Python, il ne reste plus qu'à créer une fonction permettant l'ajout automatique du préfixe " ***__label__*** " à nos labels, ainsi que l'ajout du contenu concerné à la suite.

⚠ Attention ! ⚠: Il faut au préalable nettoyer nos données, dans mon cas, les articles sont tirés du site web et possèdent de nombreuses balises HTML, il faut donc penser à retirer les éléments "parasites" de votre contenu.

```
def ecriture_label(ID_Tag, fichier):
    print(f"__label__{ID_Tag} ", end="")
    fichier.write("__label__" + str(ID_Tag) + " ")
```

Cette fonction nous permet d'écrire dans un fichier nos préfixes ! Elle écrit dans "***fichier***", avec la méthode "***write***", cette méthode n'est utilisable que si nous avons ouvert ce fichier en mode écriture avec : " **with open (fichier, 'w')** "

La fonction “***fasttestpython***” prend en paramètre notre liste python contenant nos données, ainsi que le fichier cible où nos données labellisées seront écrites. Cette fonction ne renvoie rien, mais modifie le fichier entré en paramètre.

```
def fasttestpython(liste_python, fich
    transforme_str_en_liste(liste_pyt
    with open(fichiertxt, 'w') as fic
        for indice in range(len(liste
            lister_elements_fichier(l
            lister_elements_fichier(l
            lister_elements_fichier(l

            ecriture_titre(liste_pyth

            ecriture_resume(liste_pyt
            print("")
            fichier_ja.write("\n")
```

Les fonctions “***ecriture_titre***” et “***ecriture_resume***” permettent d’écrire dans le fichier, de la même façon que “***ecriture_label***” mais pour les résumés et titres des articles.

La fonction “***lister_elements_fichier***” fera appel à la fonction “***ecriture_label***” pour chaque label associé à un contenu.

On remarque également que nous avons ouvert le fichier en mode écriture, ainsi les différents appels à la méthode write s’exécuteront.

Ainsi, suite à l’utilisation de la fonction “***fasttestpython***”, nous obtenons un fichier texte exploitable par fasttext, qui se nomme : “***JA_labels_2010_to_2024.txt”.*** Il s’agit du fichier entré en paramètre, qui a été modifié par cette fonction.

J’ai également réalisé un fichier texte nommé : “***JA_labels_autotune.txt***”, tapé à la main et qui joue le rôle de “***autotuneValidationFile***”.

Tests effectués:

- un modèle supervised et des paramètres entrés à la main
    - réponses peu précises, avec 2 labels /5 correspondant à notre recherche
- Un modèle avec le mécanisme autotune (automatisation de nos paramètres)
    - Réponses plus précises, 3 labels sur 5 sont corrects, avec une durée d’apprentissage de 1 heure
    - Impossible d’augmenter la durée d’apprentissage du modèle : manque de ressources (RAM insuffisante)

Pensée sur le projet:

Ce projet a été spécialement conçu pour Jeune Afrique, ce qui signifie que les fonctions présentées en amont ne seront compatibles, en l'état, uniquement avec des fichiers extraits des bases de données Jeune Afrique.

Concernant les articles, et la véracité des tags associés à ces articles, nous prenons le parti de nous dire que les articles sont bien taggués, et qu'ils nous servent de référence pour notre modèle. Il n'empêche qu'il est possible de trouver une coquille, comme lors de la rédaction de mon fichier JA_labels_autotune.txt, où j'ai remarqué qu'un article était mal tagué (ici le label "Coup d'État au Gabon", était associé avec un article traitant de l'architecture du Tati de Barbès, à Paris).

Continuité du projet:

Les résultats obtenus étant encourageants, ce projet est destiné à être repris et approfondi sur la partie apprentissage, avec une machine possédant une plus grande mémoire RAM (une RAM supérieur à 8Go est nécessaire).

Il serait même envisageable de l'ajouter à l'outil de recherche du site web Jeune Afrique, ce qui permettrait de rechercher les articles directement dans la base de données, selon les tags/labels retenus par fasttext lors de la recherche.

Ce projet m'a permis de découvrir la notion de Machine Learning (NLP, embeddings) et m'a permis d'extraire des données, ainsi que de les manipuler, pour mener à bien ma mission d'approfondissement sur les labels et le mécanisme autotune.

Étant actuellement en BTS SIO, je souhaite orienter la suite de mes études vers L'IA, le machine learning et le traitement des données, travailler sur ce projet a donc été une chance, et une ouverture vers ses notions.