**Projet BERT Has Uncommon Sense**

**Mise en place**

**Étape 1)** Télécharger le git + init les submodules

cd "path vers répertoire de notre choix"

git clone "<https://github.com/lgessler/bert-has-uncommon-sense.git>"

cd "bert-has-uncommon-sense"

git submodule init # Initialise les sous-modules

git submodule update # Télécharge et synchronise les fichiers du sous-module

**Étape 2)** Créer un environnement conda pour le projet, le nommer correctement et installer les librairies avec bonnes versions

conda create --name bhus python=3.8

conda activate bhus

pip install -r requirements.txt => Moi il m’a demandé d’installer le compilateur Rust pour installer tokenizers alors j’ai réalisé l’installation ci-dessous à la place car je n’avais pas envie d’installer Rust.

conda install -c conda-forge tokenizers==0.12.1

pip install allennlp==2.10.1 allennlp\_models==2.10.1

pip install pandas click bs4

Si ce n’est pas déjà fait, installer :

pip install torch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1

---------

MODIF 1 de main.py

Quand on run le modèle pour la première fois, il y a une erreur qui apparaît car on n’a pas de GPU.

***Solution 1*** (SPOILER elle n’a pas marché pour moi) : J’ai une puce M1 sur mon Mac donc ce serait bien de l’utiliser pour accélérer les calculs. Pour vérifier qu’on peut utiliser le « mps » de la puce M1, taper dans le terminal :

python

>>> import torch

>>> print("PyTorch version:", torch.\_\_version\_\_)

PyTorch version: 1.12.1

>>> print("MPS available:", torch.backends.mps.is\_available())

MPS available: True 🡨 cool Pytorch prend en compte mps.

Si je fais maintenant « pip check » il me dit «  torch 1.12.1 is not supported on this platform » alors que « MPS Available : True. »

En fait Pytorch==1.12.1 est la première version à supporter MPS donc elle n’est pas optimisée. Cependant, je ne peux pas installer Pytorch>1.12.1 car il y a des problèmes de dépendances avec allennlp==2.10.1

Maintenant, j’ai modifié quelques lignes dans le fichier main.py :

Remplacer

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

par

device = torch.device("cuda" if torch.cuda.is\_available() else "mps" if torch.backends.mps.is\_available() else "cpu")

J’ai aussi modifié dans la function finetune de main.py

Remplacer model.to(‘cuda :0’)

par

model.to(device)où le device est définit comme plus haut.

=> « mps » ne marche pas finalement ! C’est dommage mais je crois que Pytorch==1.12.1 ne gère pas bien « mps » donc je mets du « cpu » partout au final.

***Solution 2*** : (elle a marché pour moi)

Remettre

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

Remplacer model.to(‘cuda :0’) (ligne 79 de main.py)

par model.to(device)où le device est définit comme plus haut. (ligne 434 de main.py)

---------

MODIF 2 de main.py:

Je remplace

df = pd.read\_csv(paths.predictions\_tsv\_path(cfg), sep="\t", error\_bad\_lines=False)

qui est deprecated et qui m’a créé une erreur par

df = pd.read\_csv(paths.predictions\_tsv\_path(cfg), sep="\t", on\_bad\_lines='skip') (ligne 79 et ligne 318 de main.py)

---------

> Ensuite il faudra sans doute installer Jupyter Lab ou Jupyter Notebook + ipykernel correspondant pour pouvoir ouvrir les notebooks, mais je n’ai pas fait pour l’instant.

**Étape 3)** Télécharger les données

* Données **PDEP = Clres :**

Déjà fournies dans le git du projet donc à priori il n’y a rien à télécharger en plus.

* Données **OntoNotes 5.0** :

- Se rendre sur le site : <https://cemantix.org/data/ontonotes.html> : ils disent qu’il faut aller chercher les données auprès de **Linguistic Data Consortium (LDC)**

- Se rendre donc sur le site **Linguistic Data Consortium (LDC) dans la section** OntoNotes 5.0 :

<https://catalog.ldc.upenn.edu/LDC2013T19>. Il faut créer un compte et je ne l’ai pas encore fait. => *Abdellah a réussi donc il suffit de s’inscrire et créer un compte pour télécharger les données*.

* Données **streusle** :

Normalement si on a init les submodules git, ce dataset est présent déjà dans le git du projet donc il n’y a rien de plus à télécharger.

**Étape 4)** Essayer de run l’experiment du README.md du projet pour voir si tout est installé correctement. Pour cet experiment, on peut skip l’étape 3 car on n’a pas besoin de télécharger les données **OntoNotes 5.0**.

mkdir models

# finetune the model on a small number of STREUSLE instances

python main.py finetune --num\_insts 100 distilbert-base-cased models/distilbert-base-cased\_100.pt

=> je vois maintenant que j’ai le modèle distilbert-base-cased\_100.pt enregistré dans mon répertoire models/

# run the trials using the finetuned model we just created on pdep--note `clres` is an alias we use for pdep

python main.py trial \

--embedding-model distilbert-base-cased \

--metric cosine \

--query-n 1 \

--bert-layer 5 \

--override-weights models/distilbert-base-cased\_100.pt \

clres

=> Lors de la première exécution, le code reste bloqué un moment sur « reading split train » mais il faut juste attendre. Si on run une deuxième fois le même trial, ce sera plus rapide. Le code crée un répertoire cache/ et mets des trucs dedans. Attention, ils recommandent sur le README.md du projet de vider le cache/ avant chaque nouvel experiment.

# analyze results by bucket

python main.py summarize \

--embedding-model distilbert-base-cased \

--metric cosine \

--query-n 1 \

--bert-layer 5 \

--override-weights models/distilbert-base-cased\_100.pt \

clres

=> Le code m’enregistre les résultats au format .tsv directement dans le répertoire de travail bert-has-uncommon-sense.

**Étape 5)** Maintenant, ce serait bien de comprendre ce qu’il y a dans le répertoire cache/ et de comprendre les fichiers .tsv qui ont été créés.

**Dans le répertoire cache/clres-cosine-q1-predictions**.

Les fichiers enregistrés dans le répertoire « clres\_cosine\_q1\_predictions » sont des résultats d’évaluation, d’analyse ou de calcul des précisions et rappels basés sur les prédictions générées lors de l’entraînement et du test du modèle.

Chaque fichier suit une structure de nommage standard, par exemple :

`distilbert-base-cased\_models\_\_distilbert-base-cased\_100.pt\_5\_200-50000\_0.0-0.05.rec`

=> Déchiffrage de la ligne ci-dessus à l’aide de la fonction **bucketed\_metric\_at\_k\_path** dans le script **path.py**

- distilbert-base-cased\_models 🡪 cfg.embedding\_model

- models\_\_distilbert-base-cased 🡪 models fait référence au dossier dans lequel le modèle est sauvegardé. distilbert-base-cased\_100.pt fait référence au fichier des poids du modèle après fine-tuning avec 100 instances d'entraînement.

- 5 🡪 Indique la couche BERT utilisée pour extraire les embeddings.

- (rien ici mais ça peut exister) 🡪 pos

- (rien ici mais ça peut exister) 🡪 query\_category

- 200-50000 🡪 = {min\_train\_freq}-{max\_train\_freq} Spécifie la plage de fréquences d'entraînement (train frequency) des étiquettes dans le corpus : entre 200 et 50,000 occurrences.

- 0.0-0.05 🡪 {min\_rarity}-{max\_rarity} Définit la rareté (rarity) des étiquettes dans le corpus d'entraînement. Ici, les étiquettes avec une rareté entre 0.0 (très fréquentes) et 0.05 (assez rares) sont considérées.

- .rec 🡪 {ext} L'extension .rec indique qu'il s'agit sans doute d'un fichier de récapitulatif (probablement des résultats ou métriques calculées).

Types de fichiers

- `.count` : Contient des informations sur les comptages, comme la fréquence des labels ou des instances dans le jeu de données d’entraînement.

- `.lemmas` : Contient des données sur les lemmes associés aux prédictions du modèle.

- `.oprec` : Fichier contenant les précisions oracle (idéales) pour différentes configurations.

- `.orec` : Fichier contenant les rappels oracle (idéaux).

- `.prec` : Fichier contenant les précisions calculées pour les prédictions.

- `.rec` : Fichier contenant les rappels calculés pour les prédictions.

**Les fichiers results.tsv enregistrés dans le répertoire principal bert-has-uncommon-sense**

=> Déchiffrage des fichiers ci-dessous à l’aide de la fonction **summarize** dans le script **main.py**

**high\_freq\_all\_rarity\_results.tsv** => l > 500, r >= 0.25 (/•\ le nom « all\_rarity » prête à confusion mais en lisant la fonction summarize de main.py il semble que ça correspond bien à r >= 0.25 et non pas à r >= 0.25 et r < 0.25 comme pourrait le laisser entendre le nom « all\_rarity »)

Dans ce fichier .tsv on a les chiffres suivants

clres distilbert-base-cased 5 100 0.8753940686365718 0.030033018299907156 0.057463427911094024

clres 🡪 Le corpus utilisé pour l'évaluation

distilbert-base-cased 🡪 Le modèle d'embedding utilisé pour extraire les représentations.

5 🡪 La couche BERT utilisée pour extraire les embeddings.

100 🡪 Le nombre d'exemples utilisés pour le fine-tuning du modèle.

0.8753940686365718 🡪 Précision moyenne ( mean\_average(prec) ) pour le modèle sur les labels réels dans les conditions données. Ici le modèle a une précision moyenne de ~87,5 %.

0.030033018299907156 🡪 Moyenne du rappel/recall ( mean\_average(rec) ) pour le modèle sur les labels réels dans les conditions données. Ici le modèle a un rappel moyen de ~3 %.

0.057463427911094024 🡪 Moyenne du F1-score ( mean\_average(f1d) ) pour le modèle sur les labels réels dans les conditions données. Ici le modèle a un score F1 moyen de ~5,7 %.

**high\_freq\_low\_rarity\_results.tsv** => l > 500, r < 0.25

clres distilbert-base-cased 5 100 0.644459590169371 0.020404955817222455 0.03914302066972436

**low\_freq\_all\_rarity\_results.tsv** => l <= 500, r >= 0.25 (/•\ le nom all\_rarity prête à confusion mais en lisant la fonction summarize de main.py il semble que ça correspond bien à r >= 0.25)

clres distilbert-base-cased 5 100 0.8304195667819353 0.08820309023196257 0.1549017487136242

**low\_freq\_low\_rarity\_results.tsv** => l <= 500, r < 0.25

clres distilbert-base-cased 5 100 0.5763486433163372 0.08708674403780188 0.14500724008581958

**Étape 6)** Je crée à présent un nouveau script intitulé one\_experiment.sh qui s’inspire de all\_experiment.sh mais pour run seulement un experiment de mon choix.

/•\ **Ne pas oublier avant de lancer le script de suivre les recommandations du README.md :** In between runs, clear cache/, models/, and \*.tsv

**Remarque** : je ne suis pas sure qu’on ait besoin de clear les fichiers.tsv, je crois que ça écrit les résultats en dessous des résultats existants.

Peut-être qu’il n’est pas utile de clear cache/datasets/ car sinon il faudra recharger les données dans le chache/

Voici toutes les configurations qu’il nous faudra tester, qui sont inscrites dans le fichier all\_experiment.sh :

- **Dataset** : "ontonotes" "clres"

- **Modèle** : "bert-base-cased" "distilbert-base-cased" "roberta-base" "distilroberta-base" "albert-base-v2" "xlnet-base-cased" "gpt2"

- **Fine-tuning** : 0 100 250 500 1000 2500

Pour lancer le script one\_experiment.sh :

- Placer le script one\_experiment.sh dans le répertoire principal (bert-has-uncommon-sense)

- Sur l’invité de commande, se placer également dans le répertoire principal avec « cd »

- Lancer le script depuis le terminal : sh one\_experiment.sh

Se souvenir à chaque nouveau run : (issu du README.md du projet)

Type bash scripts/all\_experiments.sh to yield the numbers used in the paper's tables, which will appear in TSV files by bucket. For individual results on each trial, see .tsv files under cache/. In between runs, clear cache/, models/, and \*.tsv