Université de Montréal

IFT6285 (Automne 2025) — Devoir1
BPE

Contact :
Philippe Langlais
RALI/DIRO
Université de Montréal

+1 514 343 61 11 ext: 47494
felipe@iro.umontreal.ca
http://www.iro.umontreal.ca/~felipe/

dernière compilation : 8 septembre 2025 (20:18)

Données

Dans ce devoir, vous allez manipuler le *dataset* BioMedTok/Wikipedia, un extrait de pages wikipedia touchant au domaine bio-médical. Il s'agit d'un dataset de taille raisonnable (357Mb une fois téléchargé) qui contient 228 938 exemples (page Wikipedia). Notez qu'un exemple est une `string` qui peut contenir plusieurs phrases (qui ne sont pas segmentées). Le code suivant en python affiche les 500 premiers caractères du premier exemple :

```
ds = "BioMedTok/Wikipedia"
ds = load_dataset(ds)
ds_train = ds['train']
print(ds_train[0]['text'][:500])
```

L'exécution de ce code devrait donc produire :

```
Antoine Meillet  Paul Jules Antoine Meillet, né le à Moulins (Allier)
et mort le à Châteaumeillant (Cher), est le principal linguiste français
des premières décennies du . Il est aussi philologue. Biographie.
Enfance et formation. D'origine bourbonnaise, fils d'un notaire de
Châteaumeillant (Cher), Antoine Meillet fait ses études secondaires
au lycée de Moulins. Étudiant à la faculté des lettres de Paris à
partir de 1885 où il suit notamment les cours de Louis Havet, il
assiste également à ceux
```

À faire

Vous devez faire les choses suivantes :

1. Écrire un programme `count.py` qui compte les mots dans les exemples du *dataset*. Comme cette commande peut prendre du temps, vous pouvez spécifier le nombre d'exemples du *dataset* à considérer. Les exemples du *dataset*, étant des chaînes qui ne sont pas découpées en mots (*tokénisées*), vous implémenterez différentes stratégies de découpage en mots : aux espaces et (possiblement) ponctuations et considérerez différentes opérations de pré-traitement comme la mise en minuscule ou le remplacement des chiffres par un caractère spécial

(ex : @) ; je vous laisse être créatifs sur les opérations de pré-traitement considérées qui auront un impact sur le vocabulaire de mots réuni.

Le programme `count.py` doit produire le vocabulaire (le nombre de mots différents) et la fréquence de ces mots dans la partie lue du *dataset* mais devra également afficher le nombre d'exemples dans le *dataset* considérés, le temps mis pour compter les mots une fois le découpage en mot effectué, le type de découpage effectué, la taille du vocabulaire (nombre de mots différents) et le temps total en secondes pour effectuer ce traitement. Avec cette information, vous devriez être en mesure de faire la question qui suit.

Voici un exemple de sortie de votre programme où chaque mot est accompagné de sa fréquence (nombre de fois où il apparaît ici dans les 1 000 premiers exemples du *dataset*) ; la virgule étant le mot le plus fréquent (vu 177684 fois) :

```
, 177684
de 155872
. 122570
’ 121472
la 97193
le 74951
...
#lower-punct-digit examples.: 1000 tokens: 3359999  types:
100315 time: 9.78 (s)
```

On observe dans la dernière ligne que de traiter les 1 000 premiers exemples du *dataset* en découpant aux espaces et ponctuations, en mettant le texte en minuscule et en remplaçant les chiffres par `@` a nécessité environ 10 secondes et qu'un vocabulaire de plus de 100 000 formes a été identifié à partir d'environ 3.4M occurrences de mots.

En lançant différentes variantes de découpage en mot, et faisant varier le nombre d'exemples lus, vous obtiendrez l'information nécessaire à la question suivante.

2. Tracer une courbe qui montre comment évolue le nombre de **types** (le nombre de mots différents) en fonction du nombre de mots lus dans le *dataset*, et ce pour les différentes configurations de *tokénisation* étudiées. Vous pouvez par exemple utiliser matplotlib pour cela.

3. Écrire un programme `bpe.py` qui "entraîne" un *tokéniseur* en sous-mots à l'aide de l'algorithme BPE, un algorithme classique `bottom-up` pour représenter un ensemble de chaînes (mots et leur fréquence, tel que produit par le programme `count.py`) à l'aide d'un ensemble de sous-mots de taille contrôlée. L'algorithme commence par faire de chaque caractère un sous-mot, puis à créer un nouveau sous-mot en regroupant les deux sous-mots apparaissant en séquence dans l'entrée le plus fréquemment, et ce, jusqu'à obtenir un vocabulaire de sous-mots d'une taille désirée. Chaque opération de regroupement est mémorisée de sorte qu'il est possible par la suite d'opérer un découpage en sous-mots.

   Voici à titre indicatif (vous n'êtes pas obligé de reproduire ce format) une trace d'exécution de l'algorithme où le premier regroupement introduit le sous-mot `_d` (`_` étant ici le marqueur de début de mot), et où le 1898è regroupement produit le sous-mot `_particulièrement`. Les deux dernières lignes de la trace indiquent respectivement le vocabulaire de sous-mots produit, et le temps mis pour réunir ce vocabulaire (à partir ici des 100 000 mots les plus fréquents spécifiés en entrée) :

```
INFO 16:02:33 [bpe.py] ## merge 1:  mfp: ('_', 'd')  new token:
_d |tokens|: 822
INFO 16:02:33 [bpe.py] ## merge 2:  mfp: ('_', 'l')  new token:
_l |tokens|: 823
...
INFO 16:03:57 [bpe.py] ## merge 79:  mfp: ('_', 'in')  new
token: _in |tokens|: 900
...
INFO 16:04:42 [bpe.py] ## merge 300:  mfp: ('ti', 'n')  new
token: tin |tokens|: 1121
...
INFO 16:04:56 [bpe.py] ## merge 369:  mfp: ('_p', 'h')  new
token: _ph |tokens|: 1190
...
INFO 16:05:23 [bpe.py] ## merge 509:  mfp: ('el', 'les')  new
token: elles |tokens|: 1330
INFO 16:05:23 [bpe.py] ## merge 510:  mfp: ('_premi', 'er')
 new token: _premier |tokens|: 1331
...
```

```
INFO 16:09:33 [bpe.py] ## merge 1898:  mfp: ('_particuli',
'èrement')  new token: _particulièrement |tokens|: 2719
INFO 16:09:34 [bpe.py] ## merge 1899:  mfp: ('_plut', 'ôt')
 new token: _plutôt |tokens|: 2720
...
INFO 16:15:56 [bpe.py] ## sub-tokens (5000): pin|_aspects|_nota|...
INFO 16:15:56 [bpe.py] voc: 5000 text: 100000 total time (s):
747.07
```

À l'issue de l'exécution de ce programme, les informations nécessaires pour lancer votre *tokéniseur* en sous-mots doivent être sauvées de manière à pouvoir traiter la question suivante. Vous pouvez vous aider d'une implémentation existante pour autant que vous le mentionniez dans votre rapport. J'ai par exemple basé mon programme sur l'implémentation du notebook du livre TheLMBook.

4. Écrire un autre programme (ou modifiiez `bpe.py`) qui permet de découper en sous-mots des phrases. Bien sûr, un *tokéniseur* en sous-mots doit au préalable être "entraîné" puis sauvé (question précédente).

   Voici un exemple de phrase (découpée aux espaces et ponctuations, mise en minuscule et où les chiffres ont été remplacés par `@`) :

   ```
   étudiant à la faculté des lettres de paris à partir de @@@@
   où il suit notamment les cours de louis havet ,  il assiste
   également à ceux de michel bréal au collège de france et de
   ferdinand de saussure à l ' école pratique des hautes études
   ```

   et son découpage en 61 sous-mots séparés ici par `||` pour (peut-être) plus de lisibilité :

   ```
   _étudi ant || _à || _la || _fa cul té || _des || _lettres ||
   _de || _paris || _à || _partir || _de || _@@@@ || _où || _il
   || _suit || _notamment || _les || _cours || _de || _louis ||
   _ha ve t || _, || _il || _ass iste || _également || _à || _ceux
   || _de || _michel || _b ré al || _au || _collège || _de ||
   _france || _et || _de || _fer d in and || _de || _sa us s ure
   || _à || _l || _' || _école || _pratique || _des || _hautes
   || _études
   ```

   Votre programme doit afficher au moins le temps mis pour découper

en sous-mots l'ensemble de phrases en entrée. À titre indicatif, la sortie d'une implémentation de `bpe.py` en mode découpage :

```
INFO 16:26:24 [bpe.py] nb: 1000 sentences tokenized by fast
in  0.1661064624786377 sec.
```

le découpage de 1000 phrases prend moins de 0.2 secondes avec cette implémentation.

5. Écrire un rapport `rapport-<nom>.pdf` d'au plus 4 pages (recto seulement, fonte d'au moins 11pt) en pdf seulement (je ne lirai pas d'autre format) qui contient la ou les courbes demandées, ainsi que toute information (autre à priori que le code) que vous aimeriez amener et qui témoigne de votre curiosité. Par exemple, vous pourriez vous poser des questions comme :
   - influence du mode de découpage sur le nombre de types réunis
   - pourcentage du vocabulaire des sous-mots qui sont des mots
   - caractéristiques qui différencient les vocabulaires de sous-mots obtenus sous différentes conditions (différentes tailles de vocabulaire de sous-mots, différentes entrées, etc.)
   - pourcentage de mots donnés en entrée à `bpe.py` qui sont découpés en n sous-mots (n variant de 1 à 5 par exemple)
   - influence de la fréquence du mot sur sa decomposition
   - comparaison du vocabulaire obtenu par votre implémentation de l'algorithme BPE avec un *tokeniseur* déjà entraîné.

   Vous pouvez en effet facilement récupérer un *tokéniseur* en sous-mots depuis Hugging Face. Le code suivant permet par exemple d'extraire les 100 premiers sous-mots du *tokéniseur* du modèle `FacebookAI/roberta-base` :

```python
checkpoint = "FacebookAI/roberta-base"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

indexes = tokenizer.get_vocab().values()
vocab = list(map(lambda i:
    tokenizer.decode(i).strip(),indexes))

print(f"{len(vocab)} tokens:", " | ".join(vocab[:100]))
```

   et a pour sortie :

```
50265 tokens: plaque | meetings | Females | reinforcement |
1992 | Feld | False | XT | noted | .'" | 700 | objection |
lane | indict | prett | Sung | Die | Developer | fork | NER...
```

À remettre

Vous devez remettre en groupe d'au plus deux personnes sur Studium (activité `devoir1`) un fichier `devoir1-<nom>.tar.gz` ou `devoir1-<nom>.(g)zip` (où `<nom>` doit contenir le nom des personnes ayant réalisé le devoir (ex : `Jean-Paul-Lamarre+Pierre-Poulin-Lemaître`). Une seule remise par binôme doit être effectuée qui contient (une fois décompressée) :

- votre code qui doit être dans un répertoire `code`
- un fichier texte `readme` contenant des informations jugées utiles. À minima, vous devez indiquer :
  - dans les premières lignes de votre fichier le nom des personnes impliquées dans la réalisation du devoir.
    !!! Un oubli de nom dans ce document indique que seule la personne ayant remis sera notée !!!
  - la liste des programmes remis et brièvement leur usage
- votre rapport `rapport-<nom>.pdf` s
- des sorties pertinentes de vos programmes, toutes dans un répertoire `sorties`, votre rapport peut y faire référence.

Attention, vous pouvez remettre une archive d'au plus 5Mb (ce qui devrait largement suffire pour votre rapport, votre code et quelques sorties pertinentes). En aucun cas, je ne souhaite voir de modèle.

Mise en garde

- Si vous développez votre solutionaire sur les machines du DIRO, prenez note que vous ne devez en aucun cas exécuter vos programmes sur la machine d'accueil (`arcade`). Le faire amènerait l'équipe support à bloquer votre compte. Vous pouvez en revanche vous connecter sur une machine `ens`.
- Il se peut que j'ajuste le sujet pour le clarifier, l'heure de compilation est indiqué en première page.

Notation

Ce devoir est noté sur 5 points qui seront donnés après lecture du rapport (et au besoin inspection du code) et qui sanctionnera 1) le respect du cahier des charges (3 points) et 2) votre curiosité (2 points).

Questions

Toute question est à poser de préférence via discord sur le canal `devoir1` et ce, après avoir lu le sujet au complet. Je ferai de mon mieux pour répondre au plus vite (je suis en avance de 8h par rapport à Montréal et occupé la journée, merci de votre compréhension).