inalco
Institut national
des langues
et civilisations orientales

Institut National des Langues et Civilisations Orientales

Département Textes, Informatique, Multilinguisme

Devoir-Examen du premier semestre

Licence

Traitement Automatique des Langues

par

Perrine Quennehen
n°étudiant : 21900363

Année universitaire 2022/2023

Table des matières

1 Mise au format des données

Code utilisé pour encoder les textes en utf-8 :

```
data_folder = '../DATA/'
filenames = ["80jours_unitex_gramlab_v3_2.txt", "103-0
    _jv80_english_gutenberg_prj.txt", "
    jv80jours_ABU_unformated_iso88591.txt"]

for filename in filenames:
  with open(data_folder+filename, 'rb') as infile:
    encoding = chardet.detect(infile.read())['encoding']

  if encoding != 'utf-8':
     content = ""
     with open(data_folder+filename, 'r', encoding=encoding) as
         infile:
        content = infile.read()
     with open(data_folder+filename, 'w', encoding='utf-8') as
         outfile:
        outfile.write(content)
```

Explications du code étape par étape :

On commence par initialiser les variables à utiliser : la variable *data_folder* à laquelle on attribue la chaine de caractère contenant le chemin du dossier *DATA* puis la variable *filenames* à laquelle on attribue une liste des 3 textes à étudier.

Avec la boucle *for*, on parcourt chaque nom des fichiers contenu dans la variable *filenames*. Avec *with open()* on ouvre le fichier et le lis, grâce au mode *rb* (lecture binaire afin de lire le fichier tel quel) et à la concaténation des deux variables qui me permet d'avoir le chemin du fichier, on lit le fichier comme étant l'objet *infile*.

Ensuite, on utilise la fonction *chardet.detect()*, afin d'analyser le contenu du fichier lu dans l'objet *infile*, et de détecter l'encodage. L'encodage est détecté est affecté à la variable *['encoding']*.

Si l'encodage détecté n'est pas en *utf-8*, le code exécute la boucle *if*. Dans cette boucle, on initialise une variable *content*, à laquelle on attribue une chaine de caractère vide. Un autre *with open* ouvre le fichier en mode lecture, mais cette fois en utilisant l'encodage détecté.

Puis le contenu du fichier est lu et affecté à la variable content. Un troisième *with open* ouvre le fichier en mode écriture et en utilisant l'encodage *utf-8*, qu'on affecte à la variable *outfile*. Enfin, le contenu du fichier est écrit dans le fichier en utilisant la méthode *outfile.write()*. La boucle *for* passe à l'itération suivante et recommence les étapes pour chaque nom de fichier dans la liste *filenames*.

En résumé, ce code permet de vérifier l'encodage de chaque fichier de la liste "filenames" et de les convertir en "utf-8" si nécessaire.

Code pour l'organisation des données du premier texte :

```
with open(data_folder+filenames[0], 'r', encoding='utf-8') as infile:
  data0 = []
  metadata0 = {}
  data_found0 = False
  for line in infile:
      if line.startswith('Chapitre I'):
          data_found0 = True
      elif data_found0 and line.startswith('FIN'):
          data_found0 = False
      elif data_found0:
          data0.append(line)
print('data :', filenames[0], data0)
print('metadata :', filenames[0], metadata0)
```

Explications du code étape par étape :

Avec le *with open()* on ouvre le fichier qui se trouve au chemin *data_folder+filenames[0]* en mode lecture.

On initialise ensuite les différentes variables, la variable *data0* est une liste vide qui sera utilisé pour stocker le texte de l'auteur, la variable *metadata0* est un dictionnaire vide qui sera utilisé pour stocker les métadonnées, puis la variable *data_found0* est un booléen qui permet d'indiquer si le code parcourt des données ou pas.

Avec la boucle *for*, on parcours chaque ligne du fichier. La variable *line* contient chaque ligne du fichier au fur est à mesure qu'elle est parcouru.

Ensuite, le *if* vérifie si la ligne commence par *Chapitre I*. Si c'est le cas, *data_found0* est défini sur *True*.

Si *data_found0* est *True* et si la ligne commence par *FIN*, *data_found0* est défini sur *False*.

Si *data_found0* est *True*, la ligne est ajoutée à la liste *data0*.

La boucle *for* passe à la ligne suivante et recommence les étapes précédente jusqu'à ce qu'elle ait parcouru toutes les lignes du fichier.

Les instructions *print* permettent d'imprimer les valeurs de *data0*, et *metadata0* dans la console.

En résumé, ce code permet de lire un fichier et de séparer son contenu en deux parties : les données et les métadonnées. Les données sont stockées dans la liste *data0* et les métadonnées dans le dictionnaire *metadata0*. Les données sont toutes les lignes du fichier qui se trouvent entre les lignes qui commencent par *Chapitre I* et *FIN*. Il n'y a pas de métadonnée pour ce texte.

Structuration des données :

Afin de mettre le texte de l'auteur dans la variable *data0*, j'ai analysé le texte pour savoir par quoi le texte commence et par quoi il se termine, ensuite j'ai utilisé des chaines de caractères comportant ces expressions afin de le stocker dans la variable *data0*.

Pour les métadonnées, en parcourant le fichier texte, on remarque qu'il n'y en a pas.

Code pour l'organisation des données du second texte :

```
with open(data_folder+filenames[1], 'r', encoding='utf-8') as infile:
  data1 = []
  metadata1 = {}
  data_found1 = False
  for line in infile:
    if line.startswith('Title: '):
      metadata1['titre'] = line[7:].strip()
    elif line.startswith('Author: '):
      metadata1['auteur'] = line[8:].strip()
    elif line.startswith('Translator: '):
      metadata1['traducteur'] = line[12:].strip()
    elif line.startswith('Release Date: '):
      metadata1['date de realisation'] = line[14:].strip()
    elif line.startswith('Language: '):
      metadata1['langue'] = line[10:].strip()
    elif line.startswith('*** START OF THE PROJECT GUTENBERG EBOOK'):
      data_found1 = True
    elif data_found1 and line.startswith('*** END OF THE PROJECT
      GUTENBERG EBOOK'):
      data_found1 = False
    elif data_found1:
      data1.append(line)
print('data : ', filenames[1], data1)
print('metadata : ', filenames[1], metadata1)
```

Explications du code étape par étape :

Le *with open()* ouvre le fichier au chemin *data_folder+filenames[1]* en mode lecture et en utilisant l'encodage *utf-8*. L'objet *infile* est un objet de fichier qui est utilisé pour lire le contenu du fichier.

On initialise ensuite les différentes variables, la variable *data1* est une liste vide qui sera utilisé pour stocker le texte de l'auteur, la variable *metadata1* est un dictionnaire vide qui sera utilisé pour stocker les métadonnées, puis la variable *data_found1* est un booléen qui permet d'indiquer si le code parcourt des données ou pas.

La boucle "for" parcourt chaque ligne du fichier. La variable *line* contient la ligne courante au fur et à mesure qu'elle est parcourue.

Si la ligne commence par *Title :* , le titre est extrait de la ligne et stocké dans le dictionnaire *metadata1* sous la clé *titre*.

Si la ligne commence par *Author :* , l'auteur est extrait de la ligne et est stocké dans le dictionnaire *metadata1* sous la clé *auteur*.

Si la ligne commence par *Translator :* , le traducteur est extrait de la ligne et est stocké dans le dictionnaire *metadata1* sous la clé *traducteur*.

Si la ligne commence par *Release Date :* , la date de publication est extraite de la ligne et est stockée dans le dictionnaire *metadata1* sous la clé *date de réalisation*.

Si la ligne commence par *Language :* , la langue du texte est extraite de la ligne et est stockée dans le dictionnaire *metadata1* sous la clé *langue*.

Si la ligne commence par **** START OF THE PROJECT GUTENBERG EBOOK*, cela signifie que le début du texte est atteint. La variable *data_found1* est donc définie sur *True* pour indiquer que nous sommes maintenant dans le texte et que les lignes suivantes doivent être ajoutées à la liste *data1*.

Si la ligne commence par **** END OF THE PROJECT GUTENBERG EBOOK*, cela signifie que la fin du texte est atteinte. La variable *data_found1* est donc définie sur *False* pour indiquer que nous ne sommes plus dans le texte et que les lignes suivantes ne doivent pas être ajoutées à la liste *data1*.

En fin de boucle, les variables *data1* et *metadata1* sont imprimées pour afficher leur contenu.

En résumé, ce code permet de lire un fichier et de séparer son contenu en deux parties : les données et les métadonnées.

Les données sont stockées dans la liste *data1* et les métadonnées dans le dictionnaire *metadata1*.

Les données sont toutes les lignes du fichier qui se trouvent entre les lignes qui commencent par **** START OF THE PROJECT GUTENBERG EBOOK* et **** END OF THE PROJECT GUTENBERG EBOOK*.

Les métadonnées sont toutes les lignes qui se trouvent avant la première ligne qui commence par **** START OF THE PROJECT GUTENBERG EBOOK* ou après la dernière ligne qui commence par **** END OF THE PROJECT GUTENBERG EBOOK*. Si une ligne commence par *Title :* , *Author :* , *Translator :* , *Release Date :* ou *Language :* , alors elle est ajoutée à "metadata1" en utilisant comme clé le mot qui précède le *" : "* et comme valeur la partie de la ligne qui vient après le *" : "*.

Structuration des données :

Pour les métadonnées, en parcourant le fichier texte, on remarque qu'il y a le titre, l'auteur, le traducteur, la date de publication et la langue, d'indiqués. Afin de les extraire du texte et de les stocker dans le dictionnaire *metadata1*, j'utilise des chaines de caractère : la chaîne de caractères *Title* qui est précédée du titre lui-même sur la même ligne ; pour trouver l'auteur, j'utilise la chaîne de caractères *Author* qui est précédée de l'auteur lui-même sur la même ligne ; pour trouver le traducteur, j'utilise la chaine de caractères *Translator*, qui est précédée du traducteur lui-même sur la même ligne ; pour trouver la date de réalisation, j'utilise la chaine de caractères *Release Date*, qui est précédée de la date de réalisation ; puis pour la langue, j'utilise la chaine de caractères *Language*, qui est précédée de la langue du texte.

Afin de mettre le texte de l'auteur dans la variable *data1*, j'ai analysé le texte pour savoir par quoi le texte commençait et par quoi il terminait, ensuite, j'utilise la méthode *startswith()* avec les chaînes de caractères par quoi le texte commence et se termine, afin de stocker le texte dans la variable *data1*.

Code pour le troisième texte :

```
with open(data_folder+filenames[2], 'r', encoding='utf-8') as infile:
  data2 = []
  metadata2 = {}
  data_found2 = False
  for line in infile:
    if line.startswith('<AUTEUR'):
      metadata2['auteur'] = line[8:].strip().replace('>','')
    elif line.startswith('<IDENT_AUTEURS '):
      metadata2['identification de l\'auteur'] = line[14:].strip().
          replace('>','')
    elif line.startswith('<TITRE '):
      metadata2['titre'] = line[6:].strip().replace('>','')
    elif line.startswith('<IDENT '):
      metadata2['identification'] = line[6:].strip().replace('>','')
    elif line.startswith('<COPISTE '):
      metadata2['copiste'] = line[9:].strip().replace('>','')
    elif line.startswith('<IDENT_COPISTES '):
      metadata2['identification du copiste'] = line[15:].strip().
          replace('>','')
    elif line.startswith('<ARCHIVE '):
      metadata2['adresse de l\'archive'] = line[8:].strip().replace('
          >','')
    elif line.startswith('<VERSION '):
      metadata2['version'] = line[9:].strip().replace('>','')
    elif line.startswith('<DROITS '):
      metadata2['droits'] = line[8:].strip().replace('>','')
    elif line.startswith('<GENRE '):
      metadata2['genre'] = line[6:].strip().replace('>','')
    elif line.startswith('----------------------------- DEBUT DU FICHIER
        tdm80j2 -----------------------------'):
      data_found2 = True
    elif data_found2 and line.startswith('
        ----FIN \n'):
      data_found2 = False
    elif data_found2:
      data2.append(line)
print('data : ', filenames[2],  data2)
print('metadata : ', filenames[2],  metadata2)
```

Explications du code étape par étape :

On ouvre le fichier du dernier texte en mode lecture (r) avec l'encodage *utf-8* et on assigne le descripteur de fichier à la variable *infile*.

On initialise une liste vide *data2* et un dictionnaire vide *metadata2* qui stockeront respectivement les données du texte et les métadonnées. Puis, on initialise également une variable booléenne *data_found2* sur *False* pour indiquer si nous sommes actuellement dans les données du texte ou non.

Pour chaque ligne dans le fichier, on effectue les actions suivantes :

Si la ligne commence par *<AUTEUR*, l'auteur est extrait de la ligne et est stocké dans le dictionnaire *metadata2* sous la clé *auteur*.

Si la ligne commence par *<IDENT_ AUTEURS* , l'identification de l'auteur est extraite de la ligne et est stockée dans le dictionnaire *metadata2* sous la clé *identification de l'auteur*.

Si la ligne commence par *<TITRE* , le titre est extrait de la ligne et est stocké dans le dictionnaire *metadata2* sous la clé *titre*.

Si la ligne commence par *<IDENT* , l'identification est extraite de la ligne et est stockée dans le dictionnaire *metadata2* sous la clé *identification*.

Si la ligne commence par *<COPISTE* , le copiste est extrait de la ligne et est stocké dans le dictionnaire *metadata2* sous la clé *copiste*.

Si la ligne commence par *<IDENT_ COPISTES* , l'identification du copiste est extraite de la ligne et est stockée dans le dictionnaire *metadata2* sous la clé *identification du copiste*.

Si la ligne commence par *<ARCHIVE* , l'adresse de l'archive est extraite de la ligne et est stockée dans le dictionnaire *metadata2* sous la clé *adresse de l'archive*.

Si la ligne commence par *<VERSION* , la version est extraite de la ligne et est stockée dans le dictionnaire *metadata2* sous la clé *version*.

Si la ligne commence par *<DROITS* , les droits sont extraits de la ligne et sont stockés dans le dictionnaire *metadata2* sous la clé *droits*.

Si la ligne commence par *<GENRE* , le genre est extrait de la ligne et est stocké dans le dictionnaire *metadata2* sous la clé *genre*.

Si la ligne commence par ——————————- *DEBUT DU FICHIER tdm80j2* ——————————————, la variable booléenne *data_found2* est définie sur *True* pour indiquer que nous sommes maintenant dans les données du texte.

Si *data_found2* est *True* et que la ligne commence par *FIN* , *data_found2* est définie sur *False* pour indiquer que nous avons atteint la fin des données du texte.

À la fin de la boucle, les données du texte et les métadonnées dont imprimées.

En résumé, comme pour les précédents code, ce code permet de lire un fichier et de séparer son contenu en deux parties : les données et les métadonnées.

Les données sont stockées dans la liste *data2* et les métadonnées dans le dictionnaire *metadata2*.

Les données sont toutes les lignes du fichier qui se trouvent entre les lignes qui commencent par ——————————- *DEBUT DU FICHIER tdm80j2* ——————————————— et *FIN* .

Les métadonnées sont toutes les lignes qui se trouvent avant la première ligne qui commence par ——————————- *DEBUT DU FICHIER tdm80j2* ——————————————— ou après la dernière ligne qui commence par *FIN* . Si une ligne commence par *AUTEUR*, *IDENT_ AUTEURS*, *TITRE*, *IDENT*, *COPISTE*, *IDENT_ COPISTES*, *ARCHIVE*, *VERSION*, *DROITS* ou *GENRE*, alors elle est ajoutée à *metadata2* en utilisant comme clé le mot qui précède le " " (espace) et comme valeur la partie de la ligne qui vient après le " " (espace).

Structuration des données :

Pour les métadonnées, en parcourant le fichier texte, on remarque qu'il y a l'auteur, l'identification de l'auteur, le titre, l'identification du titre, le copiste, son identification, l'adresse de l'archive, la version, les droits ainsi que le genre. Afin de les extraire du texte et de les stocker dans le dictionnaire *metadata2*, j'utilise des chaines de caractère :

- Avec la chaîne de caractères *<AUTEUR*, l'auteur est extrait de la ligne en utilisant la sous-chaîne *line[8 :]*, qui correspond à tout le texte après le huitième caractère ;
- Avec la chaîne de caractères *<IDENT_ AUTEURS* , l'identification de l'auteur est extrait de la ligne en utilisant la sous-chaîne *line[14 :]*, qui correspond à tout le texte après le quatorzième caractère ;
- Avec la chaîne de caractères *<TITRE* , le titre est extrait de la ligne en utilisant la sous-chaîne *line[6 :]*, qui correspond à tout le texte après le sixième caractère ;
- Avec la chaîne de caractères *<IDENT* , l'identification du titre est extrait de la ligne en utilisant la sous-chaîne *line[6 :]*, qui correspond à tout le texte après le sixième caractère ;

— Avec la chaîne de caractères *<COPISTE* , le copiste est extrait de la ligne avec la sous-chaîne *line[9 :]*, qui correspond à tout le texte après le neuvième caractère ;
— Avec la chaîne de caractères *<IDENT_ COPISTES*, l'identification du copiste est extrait de la ligne grâce à la sous-chaîne *line[15 :]*, qui correspond à tout le texte après le quinzième caractère ;
— Avec la chaîne de caractères *<ARCHIVE*, l'adresse de l'archive est extrait de la ligne avec la sous-chaîne *line[8 :]*, qui correspond à tout le texte après le huitième caractère ;
— Avec la chaîne de caractères *<VERSION* , la version du texte est extrait de la ligne en utilisant la sous-chaîne *line[9 :]*, qui correspond à tout le texte après le neuvième caractère ;
— Avec la chaîne de caractères *<DROITS* , les droits sont extraits de la ligne avec la sous-chaîne *line[8 :]*, qui correspond à tout le texte après le huitième caractère ;
— Avec la chaîne de caractères *<GENRE*, le genre est extrait de la ligne grâce à la sous-chaîne *line[6 :]*, qui correspond à tout le texte après le sixième caractère.

On voit également que chaque ligne précédente se termine par *>*, afin que ce symbole ne soit pas affiché, j'utilise la fonction *replace*, pour le remplacer par une chaine vide.

Pour extraire le texte de l'auteur et le stocker dans la variable *data2*, j'utilise la méthode *startswith()* pour vérifier si une ligne commence par ——————————- *DEBUT DU FICHIER tdm80j2* ——————————— et se termine par *FIN*

2 Extraction des distributions

Code pour l'extraction des caractères et des tokens :

```
compt = 0
text = 0
caract_dist_dic = {}
token_dist_dic = {}
for file in filenames:
    with open(data_folder + file, 'r', encoding='utf-8') as f:
        current_text_dic = {}
        current_token_dic = {}
        punctuation_marks = ['.', ',', '!', '?', ':', ';', '"', '/',
            '(', ')', '_', '', '“', '”', '[', ']', '«', '»', '°', '<',
             '>']
        for line in f:
            for caract in line:
                if caract in punctuation_marks:
                    continue
                if caract not in current_text_dic.keys():
                    current_text_dic[caract] = 1
                else:
                    current_text_dic[caract] += 1

            token_line = line.split()
            for token in token_line:
                for punctuation_mark in punctuation_marks:
                    token = token.replace(punctuation_mark, '')
                if token in punctuation_marks:
                    continue
                if token not in current_token_dic.keys():
                    current_token_dic[token] = 1
                else:
                    current_token_dic[token] += 1

        current_text_dic = sorted(current_text_dic.items(), key=
            lambda x: x[1], reverse=True)
        current_token_dic = sorted(current_token_dic.items(), key=
            lambda x: x[1], reverse=True)

        caract_dist_dic[text] = current_text_dic
        print('Caractères texte : ', text, '\n', caract_dist_dic[text
            ])
        token_dist_dic[text] = current_token_dic
        print('Token texte : ', text, '\n', token_dist_dic[text])

        tokens, frequencies = zip(*token_dist_dic[text])

        plt.title("Distribution des tokens dans le texte " + str(text
            ))
        plt.bar(tokens, frequencies)
        plt.axis([0, 10, 0, 4000])
```

```
        plt.show()

    with open('../DATA/frequences_texte_{}.txt'.format(text), 'w') as
        f:
        f.write("Fréquences de caractères pour le texte {}:\n".format
            (text))
        for char, freq in current_text_dic:
            f.write("{} : {}\n".format(char, freq))

        f.write("Fréquences de tokens pour le texte {}:\n".format(
            text))
        for token, freq in current_token_dic:
            f.write("{} : {}\n".format(token, freq))

    compt += 1
    text += 1
```

Explications du code étapa par étape :

On commence par initialiser les variables *compt* et *text*, qui auront comme valeur 0 et deux dictionnaires vides *caract_dist_dic*, *token_dist_dic* utilisés à la fin afin de stocker les dictionnaires de chaque texte. On parcourt ensuite chaque fichier dans la liste *filenames* et on utilise la fonction *open* pour ouvrir chaque fichier en lecture. On définit deux dictionnaires vides, *current_text_dic* et *current_token_dic*, utilisés pour stocker les fréquences de chaque caractère et de chaque token respectivement dans chaque fichier. On définit également une liste de marques de ponctuation que nous allons ignorer lors de la comptabilisation des caractères.

Ensuite, pour chaque ligne dans le fichier en cours de traitement, on parcourt chaque caractère et on vérifie si ce caractère se trouve dans la liste de marques de ponctuation. Si c'est le cas, on l'ignore et on passe au caractère suivant. Si ce n'est pas le cas, on vérifie si le caractère est déjà présent dans le dictionnaire *current_text_dic*. Si c'est le cas, on incrémente la valeur de la clé correspondante de 1. Sinon, on ajoute le caractère au dictionnaire et on lui attribue la valeur 1.

On divise également chaque ligne en tokens en utilisant la fonction *split* et on parcourt chaque token. Pour chaque token, on parcourt la liste *punctuation_marks* qui contient des caractères de ponctuation. Pour chaque marque de ponctuation, il remplace celle-ci par une chaîne vide dans le token en cours d'analyse.

Ensuite, si le token en cours d'analyse est lui-même une marque de ponctuation (c'est-à-dire s'il est présent dans la liste *punctuation_marks*), il passe à l'itération suivante de la boucle externe.

Enfin, si le token en cours d'analyse n'est pas déjà présent dans le dictionnaire *current_token_dic*, il est ajouté avec une fréquence de 1. S'il y est déjà présent, la fréquence du token dans le dictionnaire est incrémentée de 1.

Le dictionnaire *current_token_dic* est utilisé pour compter la fréquence d'apparition de chaque token dans le texte en cours d'analyse.

Enfin, on trie les dictionnaires *current_text_dic* et *current_token_dic*, avec la fonction *sorted()* et en spécifiant que la clé de tri est la valeur du dictionnaire.

Puis, on imprime les dictionnaires.

L'avant-dernière partie de ce code va créer un diagramme à barres qui affiche la distribution des tokens dans un texte donné. Le texte est spécifié avec la variable *text*, utilisée pour accéder à la liste de tuples de tokens et de fréquences d'apparition de ces tokens dans le dictionnaire *token_dist_dic*.

La fonction *zip* permet de créer un itérateur qui agrège les éléments de plusieurs objets en paires. Dans ce cas, la fonction *zip* prend en argument **token_dist_dic[text]*, qui est la liste de

tuples de tokens et de fréquences d'apparition de ces tokens dans le texte en cours d'analyse. Avec l'opérateur de déballage de tuple *, la liste de tuples est déballée et passée en argument à *zip*, qui crée un itérateur qui agrège les éléments des tuples en paires. Ces paires sont assignées aux variables *tokens* et *frequencies*.

Ensuite, la fonction *plt.bar* est utilisée pour créer un diagramme à barres. Les arguments *tokens* et *frequencies* sont utilisés pour spécifier les données à afficher dans le diagramme. La fonction *plt.title* est utilisée pour ajouter un titre au diagramme, qui indique le numéro du texte en cours d'analyse. La fonction *plt.axis* est utilisée pour définir l'intervalle de l'axe des abscisses et de l'axe des ordonnées du diagramme. Enfin, la fonction *plt.show* est utilisée pour afficher le diagramme.

La dernière partie du code va permettre d'enregistrer les résultats dans un fichier texte. On ouvre le fichier *frequences_ texte_ {}.txt* en mode écriture, où {} est remplacé par la valeur de la variable *text*. Par exemple, si *text* vaut 1, le fichier ouvert sera *frequences_ texte_ 1.txt*.

On écrit une ligne de texte dans le fichier indiquant que les fréquences de caractères suivantes sont celles du texte *text*. La valeur de *text* est insérée dans la chaîne de formatage grâce à la syntaxe .

On parcourt ensuite chaque entrée du dictionnaire *current_ text_ dic* avec une boucle *for*, où *char* est la clé (le caractère) et *freq* est la valeur (la fréquence). On écrit dans le fichier une ligne indiquant le caractère et sa fréquence, séparés par " : ".

Ensuite, on fait de même pour les tokens, on écrit une ligne de texte dans le fichier indiquant que les fréquences de tokens suivantes sont celles du texte *text*.

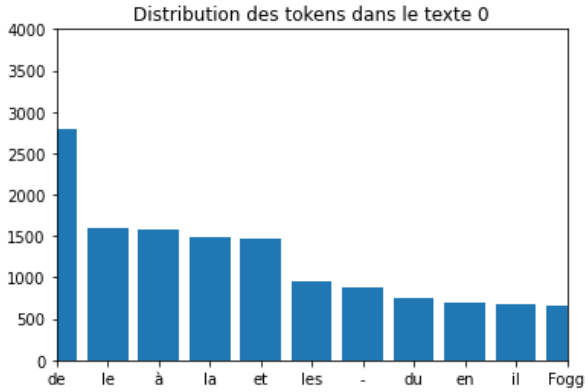
On parcourt chaque entrée du dictionnaire *current_ token_ dic* avec une boucle *for*, où *token* est la clé (le token) et *freq* est la valeur (la fréquence). Puis on écrit dans le fichier une ligne indiquant le token et sa fréquence, séparés par " : ".

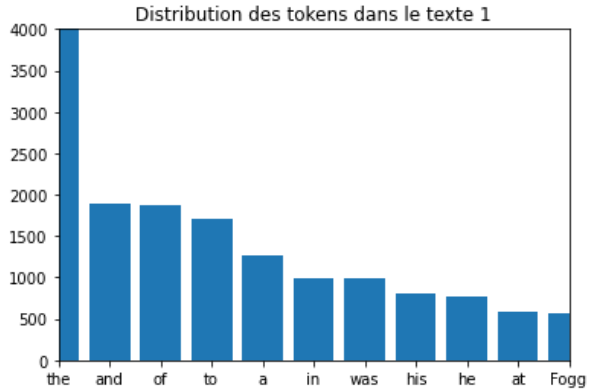
Ainsi, le fichier de sortie contiendra les fréquences d'apparition des caractères et des tokens dans chaque texte analysé.

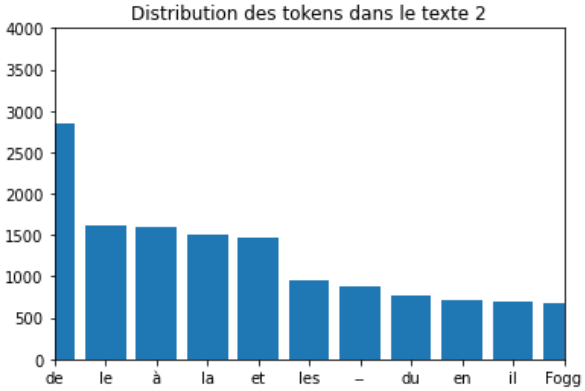
Analyse des distributions :

Concernant cette distribution, on remarque que pour les deux textes (texte 0 et texte 2) en français la distribution est la même, le type de mot que l'on retrouve le plus "de", "le", "à", "la", et "et" sont les tokens les plus fréquents dans le texte 0, "the", "of", "and", "to", et "a" sont les tokens les plus fréquents dans le texte 1.

On remarque également que les tokens les plus fréquents dans chaque texte sont des articles et des prépositions, ce qui est cohérent avec leur usage en français et en anglais.







3 Filtrage des données

Code pour l'extraction du dictionnaire et des noms des villes :

```
data_folder = '../DATA/'
filenames = ["80jours_unitex_gramlab_v3_2.txt", "103-0
    _jv80_english_gutenberg_prj.txt", "
   jv80jours_ABU_unformated_iso88591.txt"]
lexique = ["dimaju-4.1.1_utf8.txt"]

with open(data_folder+lexique[0], 'r', encoding='utf-8') as f:
    mot_sep = "\t"
    etiquette_sep = "\t"
    lexique = {}
    for ligne in f:
        index_etiquette = ligne.find(etiquette_sep)
        mot = ligne[0:index_etiquette]
        etiquettes = ligne[index_etiquette:].strip().split(
            etiquette_sep)
        lexique[mot] = etiquettes

proper_nouns_dic = {}
for text, token_freq in token_dist_dic.items():
    with open("../DATA/SBP_text_%d.txt" % text, "w") as etiquetage:
        printed_words = set()
        for mot, _ in token_freq:
            if mot in lexique:
                if "SBP:sg" in lexique[mot] or "SBP:pl" in lexique[
                    mot] and mot not in printed_words:
                    mot_etiq = '%s\n' % mot
                    etiquetage.write(mot_etiq)
                    printed_words.add(mot)
        proper_nouns_dic[text] = sorted(printed_words, key=lambda x:
            x.lower())
print(proper_nouns_dic)
```

Explication du code étape par étape :

Dans un premier temps, on définit des variables :

— *data_folder* est une chaîne de caractères qui contient le chemin d'accès au dossier de données.

— *filenames* est une liste de chaînes de caractères qui contient les noms de fichiers de texte à ouvrir.

— *lexique* est une liste de chaînes de caractères qui contient le nom du fichier de lexique.

On ouvre le fichier de lexique avec la fonction *open* et en spécifiant l'encodage *utf-8*. Le contenu du fichier est lu et stocké dans une variable appelée *f*.

On définit deux variables de chaîne de caractères : *mot_sep* et *etiquette_sep*. Ces variables sont utilisées pour séparer les mots et les étiquettes dans le fichier de lexique.

On définit une variable appelée *lexique*, qui sera utilisée pour stocker les mots et les étiquettes du fichier de lexique.

On parcourt chaque ligne du fichier de lexique grâce à une boucle *for*. Pour chaque ligne, le code trouve l'index de la première occurrence de *etiquette_sep*, ce qui permet de déterminer la position du séparateur d'étiquettes dans la ligne.

On définit la variable mot en utilisant une sous-chaîne de la ligne allant de l'index 0 jusqu'à l'index de *etiquette_sep*. La variable *etiquettes* est définie en prenant une partie de la ligne qui va de l'index de *etiquette_sep* jusqu'à la fin de la ligne, puis en enlevant les espaces en début et fin de chaîne à l'aide de la méthode *strip*, et enfin en séparant les étiquettes avec *etiquette_sep* comme séparateur grâce à la méthode *split*.

Le code ajoute le mot et ses étiquettes dans le dictionnaire *lexique* en utilisant le *mot* comme clé et les *étiquettes* comme valeur.

Une fois que le fichier de lexique a été entièrement lu et que le dictionnaire *lexique* a été rempli, on parcourt la liste *filenames* en utilisant une boucle *for*.

On crée d'abord un dictionnaire vide nommé *proper_nouns_dic*. Ensuite, on parcourt chaque entrée du dictionnaire *token_dist_dic*, précédemment créé, qui associe un numéro de texte à la liste de tuples de tokens et leurs fréquences dans ce texte. Pour chaque entrée, on ouvre un fichier nommé *SBP_text_%d.txt* où *%d* est remplacé par le numéro de texte en cours d'analyse.

On initialise ensuite un ensemble vide nommé *printed_words*. Pour chaque token dans le texte en cours d'analyse, on vérifie si le token se trouve dans un dictionnaire nommé *lexique*. Si c'est le cas et si le token a l'étiquette *SBP :sg* ou *SBP :pl* dans ce dictionnaire, on vérifie si le token n'a pas déjà été imprimé (c'est-à-dire s'il ne se trouve pas dans l'ensemble *printed_words*). Si ces deux conditions sont remplies, le token est ajouté à une chaîne de caractères nommée *mot_etiq*, puis cette chaîne est écrite dans le fichier ouvert. Enfin, le token est ajouté à l'ensemble *printed_words*.

Une fois que tous les tokens ont été traités, le code ajoute l'ensemble *printed_words*, trié par ordre alphabétique, au dictionnaire *proper_nouns_dic* associé au numéro de texte en cours d'analyse. Enfin, il affiche le dictionnaire *proper_nouns_dic*.

Pour résumer, ce dictionnaire *lexique* est un dictionnaire qui associe à chaque token une ou plusieurs étiquettes. Dans ce cas précis, le code est utilisé pour extraire les noms propres du texte en cours d'analyse et les écrire dans un fichier. Les noms propres extraits sont stockés dans le dictionnaire *proper_nouns_dic*.

Comparaison des personnages extraits avec ceux du roman _ :

Les personnages principaux sont : Phileas Fogg, Jean Passepartout, Fix, Aouda et le Colonel Proctor.

Mais on remarque qu'il n'y a aucun nom de personnage qui a été extrait des textes.

Comparaison des pays extraits avec ceux du roman :

Le roman mentionne ces différents pays (réels car il y a aussi plusieurs pays imaginaires) : Angleterre, France, Inde, Chine, Grande-Bretagne, Japon, Brésil, Allemagne, Scandinavie, Malaisie, Mexique, Pérou, Chili, Brésil, Israël, Cambodge, Hollande, États-Unis et Égypte.

On remarque que la majorité des pays ont été extraits : Japon, Angleterre, Chine, France, Grande-Bretagne, Inde, Hollande, Cambodge, Malaisie, Mexique, Pérou, Chili, Brésil, Israël, Scandinavie, Allemagne.

Code pour retrouver la liste des pays traduit dans le texte en anglais :

```
english_country_nouns = ["Japan", "England", "China", "France", "
    Britain", "India", "Holland", "Cambodia", "Malaysia", "Mexico", "
    Peru", "Chili", "Brazil", "Israel", "Scandinavia", "Germany"]
printed_country = set()
with open("../DATA/frequences_texte_1.txt", "r") as f:
    for line in f:
        for mot in line.split():
            if mot in english_country_nouns and mot not in
                printed_country:
                 printed_country.add(mot)

with open("../DATA/english_country.txt", "w") as english_country:
    for mot in printed_country:
        english_country.write(mot + "\n")

print(printed_country)
```

Explication du code étape par étape :

On commence par créer une liste de noms de pays en anglais nommée *english_country_nouns*.

Puis, on crée un ensemble vide nommé *printed_country* qui contiendra les noms de pays de la liste trouvé dans le texte.

On ouvre le fichier *frequences_texte_1.txt* qui contient la liste des tokens du texte en anglais, ici le texte 1, en mode lecture et le stocke dans une variable f.

On parcourt chaque ligne du fichier f avec une boucle *for*.

Puis pour chaque ligne, divise la ligne en mots avec la méthode *split()* et parcourt chaque mot avec une boucle *for*.

Si le mot est dans la liste *english_country_nouns* et n'est pas déjà dans l'ensemble *printed_country*, ajoute le mot à l'ensemble.

On ouvre le fichier *english_country.txt* en mode écriture et le stocke dans une variable *english_country*.

Enfin, on parcourt chaque mot de l'ensemble *printed_country* avec une boucle *for* et écrit chaque mot dans le fichier *english_country.txt*.

Comparaison avec la liste anglaise :

Après avoir traduit la liste du français à l'anglais, voici les pays que l'on retrouve dans la liste traduite et celle en anglais : France, Allemagne, Pérou, Brésil, Inde, Mexico, Israël, Japon, Hollande, Chine, Angleterre, Chili.

La liste n'est pas exactement la même que la liste en français, certains pays comme la Grande-Bretagne n'y figure pas. D'autres pays comme la Scandinavie ont été remplacés par Suède et Norvège dans le texte en anglais.

4 Annotation

J'ai utilisé l'annotation morpho-syntaxique de l'Universal Dependencies pour annoter manuellement ces 3 phrases :

PHRASE 1 : "On ne l'avait jamais vu ni à la Bourse, ni à la Banque, ni dans aucun des comptoirs de la Cité. "

manuellement	Programme
On (PRON, pronom)	On (PRV :sg)
ne (PART, particule négative)	ne (ADV PUL)
l' (DET, déterminant)	l' (DTN :sg PRV :sg)
avait (AUX, auxiliaire "avoir")	avait (ACJ :sg)
jamais (ADV, adverbe)	jamais (ADV)
vu (VERB, verbe)	vu (ADJ2PAR :sg ADJ1PAR :sg VPAR :sg SBC :sg PREP)
ni (CCONJ, conj. coordination)	ni (COO)
à (ADP, préposition)	à (PREP)
la (DET, déterminant)	la (DTN :sg PRV :sg PRO :sg SBC :sg SBC :pl)
Bourse (PROPN, nom propre)	Bourse (SBP :sg)
, (PUNCT, ponctuation)	, (,)
ni (CCONJ, conj. coordination)	ni (COO)
à (ADP, préposition)	à (PREP)
la (DET, déterminant)	la (DTN :sg PRV :sg PRO :sg SBC :sg SBC :pl)
Banque (PROPN, nom propre)	Banque (INCONNU)
, (PUNCT, ponctuation)	, (,)
ni (CCONJ, conj. coordination)	ni (COO)
dans (ADP, préposition)	dans (PREP SBC :pl)
aucun (DET, déterminant)	aucun (DTN :sg PRO :sg ADJ :sg)
des (DET, déterminant)	des (DTC :pl DTN :pl)
comptoirs (NOUN, nom commun)	comptoirs (SBC :pl)
de (ADP, préposition)	de (PREP DTN :pl DTN :sg)
la (DET, déterminant)	la (DTN :sg PRV :sg PRO :sg SBC :sg SBC :pl)
Cité (PROPN, nom propre)	Cité (INCONNU)
. (PUNCT, ponctuation)	. (.)

Taux de précision : On a un taux de précision de 88%. Car il y a 22 tokens de correctement étiquetés sur 25.

PHRASE 2 : "Ni les bassins ni les docks de Londres n'avaient jamais reçu un navire ayant pour armateur Phileas Fogg."

manuellement	Programme
Ni (CCONJ, conj. coordination)	Ni (COO)
les (DET, déterminant)	les (DTN :pl PRV :pl PRO :pl)
bassins (NOUN, nom commun)	bassins (SBC :pl)
ni (CCONJ, conj. coordination)	ni (COO)
les (DET, déterminant)	les (DTN :pl PRV :pl PRO :pl)
docks (NOUN, nom commun)	docks (SBC :pl)
de (PREP, préposition)	de (PREP DTN :pl DTN :sg)
Londres (PROPN, nom propre)	Londres (SBP :sg)
n' (PART, particule négative)	n' (ADV PUL)
avaient (AUX, auxiliaire "avoir")	avaient (ACJ :pl)
jamais (ADV, adverbe)	jamais (ADV)
reçu (VERB, verbe)	reçu (ADJ2PAR :sg ADJ1PAR :sg VPAR :sg SBC :sg)
un (DET, déterminant)	un (DTN :sg PRO :sg CAR)
navire (NOUN, nom commun)	navire (SBC :sg)
ayant (VERB, verbe "avoir")	ayant (ANCNT)
pour (PREP, préposition)	pour (PREP)
armateur (NOUN, nom commun)	armateur (SBC :sg)
Phileas (PROPN, nom propre)	Phileas (INCONNU)
Fogg (PROPN, nom propre)	Fogg (INCONNU)
. (PUNCT, ponctuation)	. (.)

Taux de précision : pour cette phrase, on a un taux de précision de 90%. Car on a 18 tokens correctement annotés sur 20.

PHRASE 3 : "En tout cas, il n'était prodigue de rien, mais non avare, car partout où il manquait un appoint pour une chose noble, utile ou généreuse, il l'apportait silencieusement et même anonymement. "

manuellement	Programme
En (PREP, préposition)	En (PREP PRV :++)
tout (DET, déterminant)	tout (PRO :sg DTN :sg ADV SBC :sg ADJ :sg)
cas (NOUN, nom commun)	cas (SBC :sg SBC :pl)
, (PUNCT, ponctuation)	, (,)
il (PRON, pronom)	il (PRV :sg)
n' (PART, particule négative)	n' (ADV PUL)
était (AUX, auxiliaire "être")	était (ECJ :sg)
prodigue (ADJ, adjectif)	prodigue (ADJ :sg VCJ :sg)
de (PREP, préposition)	de (PREP DTN :pl DTN :sg)
rien (PRON, pronom)	rien (PRO :sg SBC :sg ADV)
, (PUNCT, ponctuation)	, (,)
mais (CCONJ, conj. coordination)	mais (COO SBC :pl)
non (ADV, adverbe)	non (ADV)
avare (ADJ, adjectif)	avare (SBC :sg ADJ :sg)
, (PUNCT, ponctuation)	, (,)
car (SCONJ, conj. subordination)	car (COO SBC :sg)
partout (ADV, adverbe)	partout (ADV)
où (PRON, pronom)	où (REL ADV)
il (PRON, pronom)	il (PRV :sg)
manquait (VERB, verbe)	manquait (VCJ :sg)
un (DET, déterminant)	un (DTN :sg PRO :sg CAR)
appoint :(NOUN, nom commun)	appoint (SBC :sg)
pour (PREP, préposition)	pour (PREP)
une (DET, déterminant)	une (DTN :sg PRO :sg CAR)
chose (NOUN, nom commun)	chose (SBC :sg VCJ :sg)
noble (ADJ, adjectif)	noble (ADJ :sg SBC :sg)
, (PUNCT, ponctuation)	, (,)
utile (ADJ, adjectif)	utile (ADJ :sg SBC :sg)
ou (CCONJ, conj. coordination)	ou (COO)
généreuse (ADJ, adjectif)	généreuse (ADJ :sg SBC :sg)
, (PUNCT, ponctuation)	, (,)
il (PRON, pronom)	il (PRV :sg)
l' (PRON, pronom)	l' (DTN :sg PRV :sg)
apportait (VERB, verbe)	apportait (VCJ :sg)
silencieusement (ADV, adverbe)	silencieusement (ADV)
et (CCONJ, conj. coordination)	et (COO)
même (ADV, adverbe)	même (ADJ :sg ADV PRO :sg)
anonymement (ADV, adverbe)	anonymement (ADV)
, (PUNCT, ponctuation)	. (.)

Taux de précision : pour cette phrase, on a un taux de précision de 100%. Parce que l'on a 39 tokens correctement annotés sur 39.

Voici le programme utilisé :

```
for phrase in phrases:
    annotations = []
    mots = []
    mot_temp = ""
    for caractere in phrase:
        if caractere.isalpha() or caractere.isdigit() or caractere ==
            "'" or caractere == "-":
            mot_temp += caractere
        else:
            if mot_temp:
                if "'" in mot_temp:
                    index_apostrophe = mot_temp.index("'")
                    mot1 = mot_temp[:index_apostrophe+1]
                    mot2 = mot_temp[index_apostrophe+1:]
                    mots.append(mot1)
                    mots.append(mot2)
                else:
                    mots.append(mot_temp)
            mot_temp = ""
            mots.append(caractere)

    for mot in mots:
        if mot.isspace():
            continue
        if mot in lexique:
            etiquettes = lexique[mot]
            annotations.append((mot, etiquettes))
        else:
            annotations.append((mot, ["INCONNU"]))

    annotations_str = ["{} ({})".format(annotation[0], " ".join(
        annotation[1])) for annotation in annotations]
    annotations_str = "\n".join(annotations_str)
    with open('../DATA/phrases_annotations.txt', 'a') as f:
        f.write("\n" + phrase + "\n")
        f.write(annotations_str + "\n")
```

Explication du code étape par étape :

Pour chaque phrase du texte, on définit une liste vide *annotations* qui va contenir les annotations de chaque mot de la phrase, et une liste vide *mots* qui va contenir les mots de la phrase. On initialise une variable *mot_temp* à vide, qui va servir à stocker temporairement un mot en cours de traitement.

Pour chaque caractère de la phrase :

Si le caractère est une lettre, un chiffre, une apostrophe ou un tiret, on ajoute le caractère à *mot_temp*.

Sinon (c'est-à-dire si le caractère est un autre type de caractère, comme un espace ou une ponctuation) :

Si *mot_temp* n'est pas vide, cela signifie qu'un mot est en cours de traitement. Dans ce cas :

Si l'apostrophe est présente dans *mot_ temp*, on sépare le mot autour de l'apostrophe en deux morceaux et on ajoute ces deux morceaux dans la liste *mots*.

Sinon, on ajoute le mot complet dans la liste mots.

On réinitialise *mot_ temp* à vide et on ajoute le caractère dans la liste mots.

Pour chaque mot de la liste mots :

Si le mot est une chaîne vide ou un espace, on passe au mot suivant.

Si le mot est présent dans le lexique, on récupère les étiquettes associées au mot dans le lexique et on ajoute le mot et ses étiquettes dans la liste *annotations*.

Sinon, on ajoute le mot et l'étiquette *INCONNU* dans la liste annotations.

On crée une chaîne de caractères *annotations_ str* à partir de la liste *annotations* en formatant chaque élément de la liste sous la forme *mot (étiquettes)* et en séparant chaque élément par une nouvelle ligne.

On ouvrir le fichier *../DATA/phrases_ annotations.txt* en mode *ajout* et on écrit la phrase et les annotations dans le fichier, séparées par une nouvelle ligne.

5 Annotation II

Voici le code utilisé pour annoter le texte et calculer la fréquence de chaque étiquette

```
with open(data_folder + filenames[0], 'r') as f:
    texte = f.read()

phrases = texte.split('\n')
frequences = {}

for phrase in phrases:
    annotations = []
    mots = []
    mot_temp = ""
    for caractere in phrase:
        if caractere.isalpha() or caractere.isdigit() or caractere ==
            "'" or caractere == "-":
            mot_temp += caractere
        else:
            if mot_temp:
                if "'" in mot_temp:
                    index_apostrophe = mot_temp.index("'")
                    mot1 = mot_temp[:index_apostrophe+1]
                    mot2 = mot_temp[index_apostrophe+1:]
                    mots.append(mot1)
                    mots.append(mot2)
                else:
                    mots.append(mot_temp)
            mot_temp = ""
            mots.append(caractere)

    for mot in mots:
        if mot.isspace():
            continue
        if mot in lexique:
            etiquettes = lexique[mot]
            annotations.append((mot, etiquettes))
        else:
            annotations.append((mot, ["INCONNU"]))

    annotations_str = ["{} ({})".format(annotation[0], " ".join(
       annotation[1])) for annotation in annotations]
    annotations_str = "\n".join(annotations_str)

    with open('../DATA/texte0_annotations.txt', 'a') as f:
        f.write("\n" + phrase + "\n")
        f.write(annotations_str + "\n")

    for mot, etiquettes in annotations:
        for etiquette in etiquettes:
            if etiquette not in frequences:
                frequences[etiquette] = 1
```

```
            else:
                frequences[etiquette] += 1

with open("../DATA/etiquettes_frequence.txt", "w") as f:
    f.write("Étude de fréquence des étiquettes :\n")
    for etiquette, frequence in frequences.items():
        print("- {} : {} occurrences\n".format(etiquette, frequence))
        f.write("- {} : {} occurrences\n".format(etiquette, frequence
            ))

frequences_triees = sorted(frequences.items(), key=lambda x: x[1],
    reverse=True)
```

Explication du code étape par étape :

Pour la partie étiquetage :

On ouvre le fichier au chemin *data_folder + filenames[0]* en lecture et on stocke son contenu dans la variable texte.

On sépare le texte en phrases en utilisant le caractère de nouvelle ligne *(\n)* comme délimiteur et on stocke le résultat dans la variable *phrases*.

Pour chaque phrase dans phrases :

On crée une liste vide appelée *annotations*.

On sépare la phrase en mots en utilisant tous les caractères qui ne sont pas des lettres, des chiffres, des apostrophes ou des tirets comme délimiteurs et on stocke le résultat dans la variable mots.

Pour chaque mot dans mots :

Si le mot est un espace, passe au mot suivant (continue).

Si le mot se trouve dans le dictionnaire lexique, on ajoute une annotation à la liste annotations sous la forme *mot, étiquettes*, où *étiquettes* est la valeur associée au mot dans le dictionnaire.

Si le mot n'est pas dans le dictionnaire, on ajoute une annotation à la liste annotations sous la forme *mot, ['INCONNU']*.

On convertit la liste d'annotations en une chaîne de caractères et on l'enregistre dans la variable *annotations_str*.

On ouvre le fichier *../DATA/texte0_annotations.txt* en mode *ajout* et on y ajoute la phrase et les annotations sous forme de chaîne de caractères, en insérant des sauts de ligne entre chaque élément.

En résumer cette partie, le code lit un fichier texte, le découpe en phrases, puis sépare chaque phrase en mots et ajoute une annotation pour chaque mot en utilisant un dictionnaire de mots et leurs étiquettes. Puis, il enregistre les phrases et les annotations dans un autre fichier.

Pour la partie fréquence :

Pour chaque mot et ses étiquettes dans la liste annotations, puis pour chaque étiquette de ce mot :

Si l'étiquette n'est pas dans le dictionnaire *frequences*, on ajoute une entrée au dictionnaire en lui associant la valeur 1.

Sinon, on incrémente la valeur associée à l'étiquette de 1.

On ouvre le fichier *../DATA/etiquettes_frequence.txt* en mode écriture et on y écrit la chaîne de caractères *Étude de fréquence des étiquettes :* suivie d'un saut de ligne.

Pour chaque étiquette et sa fréquence dans le dictionnaire *frequences* :

On affiche la chaîne de caractères - *étiquette : frequence occurrences* suivie d'un saut de ligne.

On écrit la même chaîne de caractères dans le fichier *'../DATA/etiquettes_frequence.txt*.

On trie le dictionnaire *frequences* par ordre décroissant de fréquence et on stocke le résultat dans la variable *frequences_ triees*.

Pour résumer, ce code compte la fréquence d'apparition de chaque étiquette dans les annotations, puis affiche et enregistre ces fréquences dans un fichier. Il trie également le dictionnaire de fréquences par ordre décroissant de fréquence.

Voici les 10 annotations les plus fréquentes :

- INCONNU : 5267 occurrences
- CAR : 2459 occurrences
- SUB : 2123 occurrences
- PRV :++ : 2221 occurrences
- PRV :sg : 9031 occurrences
- PRV :pl : 2946 occurrences
- DTN :sg : 11459 occurrences
- , : 6940 occurrences
- PREP : 9141 occurrences
- SBC :sg : 17440 occurrences
- PRO :sg : 8245 occurrences
- SBC :pl : 7589 occurrences

Commentaires :

L'étiquette *INCONNU* est la plus fréquente, ce qui signifie qu'il y a de nombreux mots dans les annotations qui ne sont pas présents dans le dictionnaire de mots et de leurs étiquettes utilisé. En effet, on remarque, par exemple, que pour les noms de personnage du texte, aucun n'est ressorti lorsque l'on a extrait tous les mots ayant pour étiquette *SBP :sg* dans la partie *Filtrage de données*.

L'étiquette *CAR* (cardinal) est également assez fréquente, ce qui indique que des nombres cardinaux sont couramment utilisés dans les phrases.

Les étiquettes *SUB* (subordonnée) et *PREP* (préposition) sont également assez fréquentes, ce qui indique que ces mots sont couramment utilisés dans les phrases pour relier différentes parties de la phrase ou pour introduire des propositions subordonnées ou pour lier un nom ou un pronom à un autre élément de la phrase.

Les étiquettes *PRV :++*, *PRV :sg* et *PRV :pl* sont liées aux pronoms verbaux et sont toutes assez fréquentes, ce qui indique que ces mots sont couramment utilisés dans les phrases pour remplacer un verbe et ses compléments.

Les étiquettes *DTN :sg* et *PRO :sg* sont liées aux noms et aux pronoms et sont toutes assez fréquentes, ce qui montre que ces mots sont également couramment utilisés dans les phrases pour remplacer un nom et ses compléments ou pour introduire une proposition subordonnée.

Bibliographie

[1] Essai de remise a jour de la doc pour BRILL14-JL5. fr. URL : https://studylibfr.com/doc/1601086/essai-de-remise-a-jour-de-la-doc-pour-brill14-jl5.

[2] Quick start guide — Matplotlib 3.6.2 documentation. URL : https://matplotlib.org/stable/tutorials/introductory/quick_start.html#sphx-glr-tutorials-introductory-quick-start-py.

[3] Recension et présentation comparative d'étiqueteurs pour le français et l'anglais. URL : http://www.revue-texto.net/1996-2007/Corpus/Publications/Poudat_Taggers.html.

[4] Universal POS tags. URL : https://universaldependencies.org/u/pos/index.html.

[5] Usage — chardet 5.0.0 documentation. URL : https://chardet.readthedocs.io/en/latest/usage.html#example-using-the-detect-function.