Programmation de Modèles Linguistiques (I)

(L5SOPROG L3 Sciences du Langage)

2024-2025

TD1 Analyse en caractères

Caroline Koudoro-Parfait, Sorbonne Université.
Crédit : Gaël Lejeune, Sorbonne Université

Objectifs

- Faire des manipulations simples sur un texte
- Identifier les propriétés morphologiques d'une langue
- Les comparer à d'autres

Exercice 1 : Distribution des mots selon la taille en caractères

Nous allons implanter le calcul de la distribution des mots d'un texte (son vocabulaire) en fonction de leur taille en caractères. Ainsi, vous devrez calculer le nombre de mots uniques pour une taille donnée comme le montre la Figure 1.

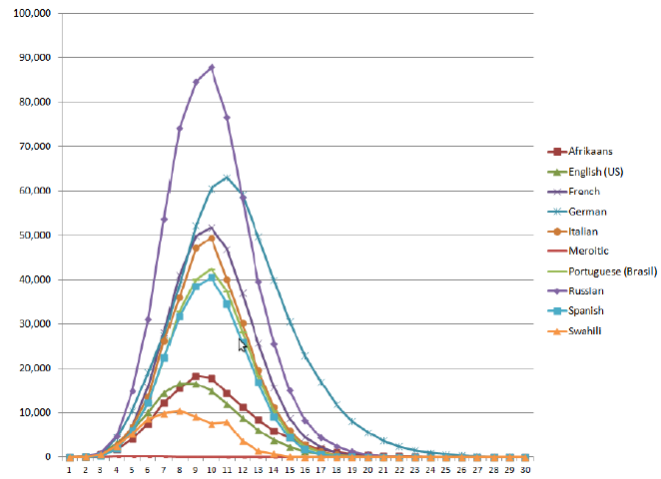

Figure 1: Distribution des mots par rapport à leur taille en caractères dans différentes langues

L'axe des abscisses représente la taille d'un mot en caractères et l'axe des ordonnées représente le nombre de mots uniques correspondant à cette taille.

Pour préparer votre espace de travail sur votre notebook JUPYTER, créez un dossier TD1 dans lequel vous enregistrerez votre code et vos données.

Pous constituer un corpus, vous utiliserez les fichiers textes suivants (choisissez plain text utf-8):

- "Le discours de la méthode" (fr) `http://www.gutenberg.org/ebooks/13846`
- "Ulysses" (en) `http://www.gutenberg.org/ebooks/4300`

Les étapes :

1. **lire** les textes
2. **découper** en mots (ou tokeniser)
3. **compter** le nombre de mots par taille de caractères
4. **observer** les résultats chiffrés
5. **représenter** cela sur une courbe

Etape 1 : lire

On ouvre le fichier en indiquant son chemin (*path*), si vous avez bien enregistré votre fichier au même endroit que votre code, le nom du fichier suffit .

Si ça ne marche pas c'est que :

- Tout n'est pas au bon endroit (`File not Found`), regardez dans l'onglet `files` de JUPYTER pour voir où vous êtes.
- ou que on a un problème d'encoding (`charmap`), il faut ajouter ecoding ='utf-8' dans le open : open("13846-0.txt", encoding ="utf-8")

Nous allons commencer par le "Discours de la Méthode", si vous avez conservé le nom d'origine il devrait s'appeler "13846-0.txt".

```
with open("13846-0.txt") as f:
  chaine = f.read()
```

Et on affiche un bout du texte pour vérifier que ça marche :

```
print(chaine[:100])
```

Etape 2 : découper

On va très simplement découper en mots avec la **méthode** *split*

```
liste_mots = chaine.split()#approximation des occurrences
print("Nombre de mots : %i" %len(liste_mots))
```

Etape 3 : compter

On va utiliser un **dictionnaire** (ou tableau associatif) où l'on va stocker pour chaque longueur en caractères le nombre de mots qu'on a rencontré. Le fonctionnement est le suivant:

- pour chaque mot de la liste de mots, on calcule sa longueur
- on vérifie si on a déjà rencontré un mot de cette longueur:
  - Si c'est le premier mot pour cette longueur on crée une **clé** pour cette longueur à laquelle on affecte la **valeur** 1
  - Sinon, on **incrémente** de 1 la valeur existante

```
dic_longueurs = {} #un dictionnaire vide

for mot in liste_mots:
  longueur = len(mot)#la longueur du mot
   #on a jamais vu cette longueur de mot
  if longueur not in dic_longueurs:
    dic_longueurs[longueur]=1 #
  else: #on a vu cette longueur de mot
    dic_longueurs[longueur]+=1

print(dic_longueurs)#pour avoir une vue de ce qu'on a fait
```

NB: si le processus ne vous semble pas clair, ajoutez au début de la boucle *for* deux lignes (avec l'indentation) pour suivre le processus pas à pas :

```
  print(dic_longueurs)
  dd=input("Appuyez sur Enter pour passer a la suite")
```

Etape 4: observer

Un dictionnaire n'est pas une structure de données ordonnée, pour vérifier que'on trouve des résultats proche de l'attendu, on va afficher le nombre d'occurences enregistré dans `dic_longueurs` pour toutes les longueurs de 1 à 30 en utilisant **l'itérateur** *range*. Dans le *print* on utilise du **formatage de chaînes de caractères**[1].

```
for toto in range(1, 31):#de 1 à 30 (31 est exclu)
  nbr_occurences = dic_longueurs[toto]
  print("%s : %s"%(toto, nbr_occurences))
  #ou
  print(f"{toto} : {nbr_occurences}")
```

Vous verrez que le code plante car on a des longueurs qui ne sont pas dans le dictionnaire, on va donc améliorer le code de la façon suivante:

```
for toto in range(30):
  if toto in dic_longueurs:
    nbr_occurences = dic_longueurs[toto]
    print("%s : %s"%(toto, nbr_occurences))
  else:
    nbr_occurences = 0
    print("%s : %s"%(toto, nbr_occurences))
```

Etape 5 : représenter

Et maintenant c'est magique, on va créer une courbe grâce à la librairie `matplotlib`. On va importer cette librairie et la renommer pour que ça soit plus court à écrire. Puis pour avoir les valeurs à mettre sur la courbe on va lire les valeurs dans l'ordre croissant pour les ranger dans une liste nommée *liste_effectifs*. Pyplot prend entrée un **vecteur**, une liste de valeurs ordonnées.

```
import matplotlib.pyplot as pyplot #import avec alias

liste_effectifs = []
```

[1] Voir par exemple `https://stackoverflow.com/questions/5082452/string-formatting-vs-format`

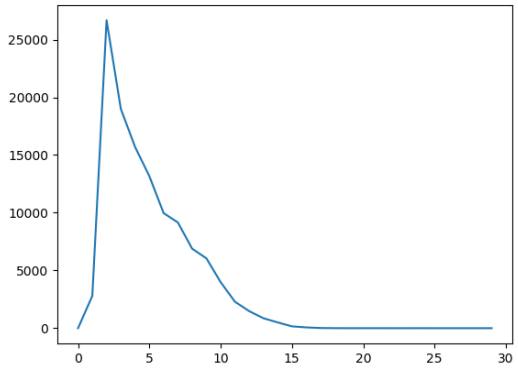

Figure 2: "Discours de la Méthode" : nombre de mots par longueur (en abscisse), en ordonnée l'effectif

```
for toto in range(30):
  if toto in dic_longueurs:#on a donc vu des mots de cette longueur
    liste_effectifs.append(dic_longueurs[toto])
  else:#on en n'a pas vu de cette longueur, on ajoute donc un 0
    liste_effectifs.append(0)
pyplot.plot(liste_effectifs)#on "dessine"
pyplot.show()#"on affiche"
```

Maintenant si on veut faire le même calcul pour l'autre texte on a juste à changer le nom du fichier dans l'étape 1 et à relancer toutes les cellules. Mais si on avait 100 textes à faire ça ne serait pas très pratique. Nous allons donc voir dans l'exercice suivant comment améliorer le code.

Exercice 2 : Factorisation du code

Pour améliorer nous allons construire des **fonctions** pour **factoriser** les traitements et constituer une **chaîne de traitement** fiable.

Etape 1: lire

Ce qui va changer ici c'est qu'on veut traiter plusieurs textes facilement. Bien sûr on pourrait faire :

```
with open("13846-0.txt") as f:#Discours de la Methode
  chaine1 = f.read()
with open("4300-0.txt") as f:#Ulysses
  chaine2 = f.read()
```

La première chose que l'on remarque c'est que sur Windows (le problème ne se pose pas avec Mac ou Linux) on arrive à ouvrir le texte en anglais mais pas celui en français. C'est un problème d'encodage des caractères (`charmap` error). L'encodage c'est la manière dont on stocke les caractères en les changeant en "0" et en "1". Pour faire simple, quand on a des caractères accentués on a besoin d'un encodage (*encoding*) adapté. Ici c'est "utf-8", on va faire la modification suivante :

```
with open("13846-0.txt", encoding='utf-8') as f:#Discours de la Methode
  chaine1 = f.read()

with open("4300-0.txt") as f:#Ulysses
  chaine2 = f.read()
```

On voit que l'on a répété deux fois le même code et aussi que si l'on a des correctiosn à faire, il faudra le faire deux fois.

Si on a 100 textes à traiter il faut 300 lignes de code, pas très pratique.Nous allons voir avec une fonction ça marche mieux. Pour la fabriquer il faut décomposer notre problème, voir **ce qui est constant** (factorisable, les opérations) et **ce qui est variable** (non factorisable, paramètre ou sortie de la fonction). On se rend vite compte que ce que l'on veut c'est à partir d'un chemin de fichier (**input** ou entrant) avoir son contenu sous forme de chaîne de caractères (**output** ou sortant). Ce qui nous donne :

```
def lire_fichier(chemin):
  with open(chemin, encoding = 'utf-8') as f:
    chaine = f.read()
  return chaine

chaine1 = lire_fichier("13846-0.txt")#Discours de la Methode
chaine2 = lire_fichier("4300-0.txt")#Ulysses
```

Etape 2 : découper

Toujours réfléchir en terme d'entrant/sortant : Quel est l'entrant et le sortant qu'il faut ajouter dans le squelette ci-contre à la place des XXX, YYY et ZZZ?

```
def decouper_en_mots(XXX):
  #on decoupe
  liste_mots = YYY
  return ZZZ
```

(à vous de réfléchir, réponse page suivante)

```
def decouper_en_mots(chaine):
  #on decoupe
  liste_mots = chaine.split()
  return liste_mots

liste_mots1 = decouper_en_mots(chaine1)
liste_mots2 = decouper_en_mots(chaine2)
```

Etape 3 : compter
En entrée : la liste de mots
En sortie : les effectifs
NB: vous pouvez mettre des *print* quand vous testez pour bien voir ce qu'il se passe.

```
def get_effectifs(liste_mots):
  dic_longueurs = {}
  for mot in liste_mots:
    longueur = len(mot)#la longueur du mot
    if longueur not in dic_longueurs: #on a jamais vu cette longueur de
      dic_longueurs[longueur]=1 #
    else: #on a vu cette longueur de mot
      dic_longueurs[longueur]+=1
  return dic_longueurs
```

Et cette fonction on va l'utiliser directement dans l'étape 5

Etape 4 : observer (obsolète)

C'était une étape de vérification devenue inutile puisqu'on n'a pas changé les opérations effectuées.

Etape 5 : représenter

Ici on va pouvoir afficher les deux courbes sur la même figure et cerise sur le gâteau on va la sauvegarder.

```
import matplotlib.pyplot as pyplot #import avec alias

for liste in [liste_mots1, liste_mots2]: #on a une liste de liste pour
  dic_longueurs = get_effectifs(liste)
  liste_effectifs = []
  for toto in range(30):
    if toto in dic_longueurs:#on a donc vu des mots de cette longueur
      liste_effectifs.append(dic_longueurs[toto])
    else:#on en n'a pas vu de cette longueur, on ajoute donc un 0
      liste_effectifs.append(0)
  pyplot.plot(liste_effectifs)#on "dessine" mais dans la boucle

pyplot.show()#"on affiche" mais hors de la boucle (pour avoir tout)
```

On se rend compte que la figure est difficile à interpréter, en effet on travaille en valeur absolue alors que les textes sont de taille différente. On va donc utiliser la taille de chaque texte en mots (avec la fonction `len`) pour avoir cette fois une figure avec la proportion de mots de chaque longueur :

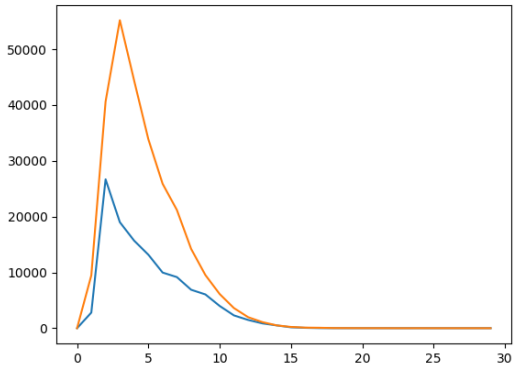

Figure 3: "Discours de la Méthode" et "Ulysses" : nombre de mots par longueur (en abscisse), en ordonnée l'effectif

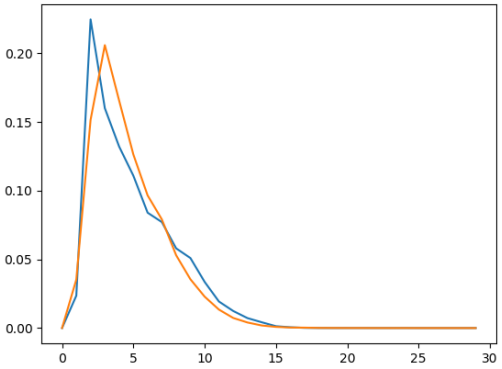

Figure 4: "Discours de la Méthode" et "Ulysses" : nombre de mots par longueur (en abscisse), en ordonnée la fréquence

```
#On remplace la ligne :
        liste_effectifs.append(dic_longueurs[toto])
#Par:
        liste_effectifs.append(dic_longueurs[toto]/len(liste))
```

Voir page suivante pour un bilan

Après une dernière étape de factorisation voici où nous en sommes :

```
def lire_fichier(chemin):
  with open(chemin, encoding = "utf-8") as f:
    chaine = f.read()
  return chaine

def decouper_en_mots(chaine):
  liste_mots = chaine.split()
  return liste_mots

def get_effectifs(liste_mots):
  dic_longueurs = {}
  for mot in liste_mots:
    longueur = len(mot)
    if longueur not in dic_longueurs:
      dic_longueurs[longueur]=1
    else:
      dic_longueurs[longueur]+=1
  return dic_longueurs

def vecteur_longueurs(dic_longueurs, liste_mots):
  liste_effectifs = []
  for toto in range(30):
    if toto in dic_longueurs:
      liste_effectifs.append(dic_longueurs[toto]/len(liste_mots))
    else:
      liste_effectifs.append(0)
  return liste_effectifs

import matplotlib.pyplot as pyplot

for chemin in ["13846-0.txt", "4300-0.txt"]:
  chaine = lire_fichier(chemin)
  liste_mots = decouper_en_mots(chaine)
  dic_longueurs = get_effectifs(liste_mots)
  liste_effectifs = vecteur_longueurs(dic_longueurs, liste_mots)
  pyplot.plot(liste_effectifs)

pyplot.savefig("frequences.png")#le bonus: on sauvegarde
pyplot.show()
```

C'est pas mal, au prochain TD on améliorera le rendu de la figure (légende, échelle) et on travaillera sur plus de langues.