UNIVERSITÉ DE LORRAINE

Phonétique

NLP Master 1
UE704 EC2

Cécile MACAIRE
Ludivine ROBERT

Décembre 2019

Table des matières

1 Introduction

Nous avons choisi de traiter le sujet se rapportant à la syllabification des mots du français, en accord avec notre intérêt pour le traitement automatique de la langue.
La syllabification, selon Goldsmith, désigne le processus consistant à associer une chaîne linéaire de phonèmes pour obtenir une structure syllabique, c'est à dire la segmentation d'un mot en syllabes.

Dans les parties suivantes, nous décrivons les méthodes et techniques utilisées pour obtenir le fichier nécessaire (forme VC de la forme orthographique et phonétique du mot, syllabification de la forme phonétique et représentation de la forme syllabique sous forme VC) à la réalisation de la deuxième partie du projet. Celle-ci consiste à récupérer les structures syllabiques les plus fréquentes en tenant compte de divers paramètres.

Pour réaliser ce projet, nous avons développé un code python et utilisé la librairie nltk, notamment pour déterminer les fréquences demandées.

Pour lancer le programme, un fichier input avec la forme orthographique et phonétique de chaque mot est nécessaire, ainsi qu'un fichier output vide, qui sera rempli à l'exécution du code.

2 Première étape : création du fichier output

Pour la création du fichier output demandé, nous avions à disposition un fichier input contenant la forme orthographique et la forme phonétique de chaque mot français.

```
abdiquent abdik
abdiquera abdikRa
abêtit abeti
abhorrait abORE
abîmerai abimRE
abjurent abZyR
ablations ablasj1
abolisse abOlis
abominai abOminE
abondamment ab1dam@
```

FIGURE 1 – Fichier input fourni.

Nous avons utilisé le langage python pour réaliser les différentes opérations demandées afin de produire le fichier output.

2.1 Conversion sous forme V (voyelle) et C (consonne)

La première opération que nous avons effectué concernait la conversion des mots sous la forme VC. Par exemple, la forme orthographique suivante *abdiquera* devient *VCCVCVVCV* et la forme phonétique correspondante *[abdikRa]* devient *VCCVCCV*.

```
def ortho_VC(l):
    words_VC = []
    for word in l:
        word_VC = ''
        for letter in word:
            if letter in ortho_c:
                word_VC += 'C'
            else:
                word_VC += 'V'
        words_VC.append(word_VC)
    return words_VC
```

FIGURE 2 – Fonction *ortho_ VC* pour la conversion sous forme VC de la forme orthographique d'un mot.

Cette fonction **ortho_VC** permet de regarder chaque lettre composant le mot. Si cette lettre est une consonne (*p, t, k, b, d, g, f, s, c, v, z, g, j, m, n, r, l, h*), elle est donc labellisée consonne, sinon voyelle (*a, e, i, o, u, y, é, è, ê, ë, ï, à, ù*).

Pour la conversion de la forme phonétique sous forme VC, la méthode est basée sur le même principe.

2.2 Représentation syllabique de la forme phonétique

Il était, par la suite, demandé de représenter la forme phonétique sous forme syllabique, en prenant en compte différentes règles. La fonction **syllabification** de notre code python réalise cette représentation syllabique.

2.2.1 1 consonne entre 2 voyelles (VCV)

Si une consonne se trouve entre deux voyelles, alors la consonne sera en première position de la deuxième syllabe. VCV deviendra donc V-CV. On peut prendre comme exemple le mot *[adOsE]* (VCVCV). La représentation syllabique sera donc *[a-dO-sE]* (V-CV-CV).

```
if word[i] in syl_vow and word[i + 1] not in syl_vow and word[i + 2] in syl_vow:
    string += word[i] + '-' + word[i + 1] + word[i + 2]
    is_add = True
    i += 2
```

FIGURE 3 – Code python de la gestion de la forme VCV.

La première ligne du code regarde si la lettre du mot à l'itération i est une voyelle, puis la suivante une consonne et enfin la dernière une voyelle. Si tel est le cas, alors on applique la règle.

2.2.2 2 voyelles adjacentes (VV)

2 voyelles adjacentes sont deux syllabes distinctes : *[aeROnEf]* devient *[a-e-RO-nEf]* (a et e sont séparées).

2.2.3 2 consonnes entre 2 voyelles (VCCV)

Plusieurs cas de figure sont présents :

- VCCV devient V-CCV si :
  - C_1 n'est ni une liquide (R, l), ni une semi-voyelle (w, j, 8) et C_2 est une liquide ou semi-voyelle. Par exemple, le mot *[al@djE]* devient *[a-l@-djE]*.
- VCCV devient VC-CV si :
  - C_1 et C_2 sont liquides. Par exemple, le mot *[SaRlat@]* devient *[SaR-la-t@]* (R et l sont séparées).
  - C_1 est une liquide et C_2 est ni une liquide, ni une semi voyelle. Par exemple, le mot *[defORme]* devient *[de-fOR-me]* (R et m sont séparées).
  - C_1 est une semi-voyelle et C_2 est n'importe quelles consonnes. Par exemple, le mot *[f9jte]* devient *[f9j-te]* (j et t sont séparées).
  - C_1 et C_2 ne sont ni liquides, ni semi-voyelles. Par exemple, le mot *[Zimnaz]* devient *[Zim-naz]* (m et n sont séparées).

```
elif word[i + 1] not in (syl_vow + liquids + semi_vowels) and word[i + 2] in (
        liquids + semi_vowels):
    string += word[i] + '-' + word[i + 1] + word[i + 2] + word[i + 3]
    is_add = True
    i += 3
```

FIGURE 4 – Code python de l'une des règles pour la gestion de la forme VCCV.

Après avoir en premier regardé si la première lettre phonétique et la 4ème lettre phonétique à partir de l'itération i sont une voyelle, le code regarde si la première consonne n'est ni une voyelle, ni une liquide, ni une semi-voyelle, et si la suivante est soit, une liquide ou une semi-voyelle. Alors l'output sera de la forme : *voyelle - consonne consonne voyelle*.

2.2.4 3 consonnes entre 2 voyelles (VCCCV)

Dans le cas où 3 consonnes sont situées entre deux voyelles, la règle suivante s'applique. En effet, si :

— C_1 n'est ni une liquide, ni une semi-voyelle,
— C_2 est une liquide,
— C_3 est une semi-voyelle,

alors la forme VCCCV devient V-CCCV. Par exemple, le mot *[@plwa]* devient *[@-plwa]*.

```
if (word[i] and word[i + 4]) in syl_vow:
    if word[i + 1] not in (syl_vow+liquids+semi_vowels) and word[i + 2] in liquids and word[i + 3] in semi_vowels:
        string += word[i] + '-' + word[i + 1] + word[i + 2] + word[i + 3] + word[i + 4]
        is_add = True
        i += 4
```

FIGURE 5 – Code python pour la gestion de la forme VCCCV.

Le code python suivant regarde si les lettres entourant le groupe sont des voyelles, puis regarde si les lettres correspondent bien aux règles énoncées précédemment. Si tel est le cas, on ajoute une séparation entre la première voyelle et le reste du groupe.

2.2.5 3 consonnes ou plus entre 2 voyelles (VCC..CV)

Si les 3 consonnes ne sont pas des liquides ni des semi-voyelles, alors la forme VCC..CV devient VC-..CCV. Par exemple, le mot *[OpskyRsi]* est découpé comme ceci : *[Op-skyR-si]* (le groupe *[Opsky]* devient *[Op-skY]*). Un autre exemple lorsque le mot contient 4 consonnes consécutives : *[sypstRa]* devient *[syp-stRa]*.

```
if (word[i] and word[i + 4]) in syl_vow:
    if word[i + 1] not in syl_vow and word[i + 2] not in syl_vow and word[i + 3] not in syl_vow:
        string += word[i] + word[i + 1] + '-' + word[i + 2] + word[i + 3] + word[i + 4]
        is_add = True
        i += 4
```

FIGURE 6 – Code python pour la gestion de la forme VCCCV.

Pour gérer la forme VCCCV dans le code python, nous regardons d'abord si la première lettre syllabique et la dernière lettre syllabique du groupe sont des voyelles. Si c'est bien le cas, on regarde tout simplement si les 3 consonnes ne sont pas des voyelles. Rappelons ici que ce cas de figure est visible que si la règle de la partie 2.2.4 ne s'applique pas.

2.3 Création du fichier output

Après avoir récupéré la forme VC de la forme orthographique et phonétique, et appliqué les règles pour avoir la forme syllabique de la forme phonétique, et la forme VC de la forme syllabique obtenue, nous créons le fichier output. Il résulte donc 6 colonnes.

```
desservies CVCCVCCVVC desERvi CVCVCCV de-sER-vi CV-CVC-CV
dessinerai CVCCVCVCVV desinRE CVCVCCV de-sin-RE CV-CVC-CV
dessineront CVCCVCVCVCC desinR1 CVCVCCV de-sin-R1 CV-CVC-CV
dessoûla CVCCVVCV desula CVCVCV de-su-la CV-CV-CV
destin CVCCVC dest5 CVCCV des-t5 CVC-CV
destitution CVCCVCVCVVC destitysj1 CVCCVCVCCV des-ti-ty-sj1 CVC-CV-CV-CCV
déstocké CVCCVCCV destOke CVCCVCV des-tO-ke CVC-CV-CV
destructive CVCCCVCCVCV destRyktiv CVCCCVCCVC des-tRyk-tiv CVC-CCVC-CVC
détachant CVCVCCVCC detaS@ CVCVCV de-ta-S@ CV-CV-CV
```

FIGURE 7 – Fichier output obtenu.

3 Deuxième étape : extraction des fréquences

En ce qui concerne l'analyse des fréquences, nous avons utilisé les formes du fichier output précédemment créées, et plus particulièrement les formes phonétiques et VC syllabées. Nous avons également utilisé le langage python ici.

3.1 Extraction des différentes structures syllabiques

Dans un premier temps, la fonction **cut_syll** de notre code permet de séparer toutes les structures syllabiques de nos listes contenant les formes phonétiques et VC syllabées.

```
def cut_syll(l):
    cut = [a.split('-') for a in l]
    return [x for z in cut for x in z if x]
```

FIGURE 8 – Fonction *cut_syll* pour l'extraction des structures syllabiques.

Cette fonction nous permet de récupérer une liste, uniquement composée de syllabes, nécessaire pour la prochaine étape.

3.2 Analyse des fréquences

Dans un second temps, nous avons utilisé nltk qui contient une fonction permettant l'extraction de fréquences : *FreqDist*.

Notre fonction **Freq_VC** utilise alors la fonction importée de nltk pour nous afficher les valeurs les plus fréquentes de notre liste de structures syllabiques. Nous avons le choix de définir le nombre que l'on souhaite, ici 15. Si nous ne faisons pas ce choix, nous pouvons obtenir un très grand éventail de valeurs.

```
def freq_VC(l):
    frequency = nltk.FreqDist(l)
    return frequency.most_common(15)
```

FIGURE 9 – Fonction *freq_VC* pour le calcul de fréquences.

Cette fonction a été appliquée afin d'obtenir les 15 plus fréquentes syllabes sous les formes VC et phonétiques (i.e. consonnes et voyelles).

Il était aussi demandé de regarder les syllabes de par leurs caractéristiques phonétiques. Pour cela, nous ne pouvions pas utiliser la même fonction directement. Il nous a d'abord fallu rechercher pour un élément sa caractéristique. Nous avons donc au départ accordé les sons à leurs labels, tel qu'on peut le voir ci-dessous.

```
stop_V = ["b", "d", "g"]
stop_U = ["p", "t", "k"]
fric_V = ["v", "z", "Z"]
fric_U = ["f", "s", "S"]
liquids = ["R", "l"]
nasals = ["m", "n", "N", "G"]
semi_vowels = ["w", "j", "8"]
vowels_O = ["a", "e", "i", "u", "o", "y", "E", "9", "2", "O", "*"]
vowels_N = ["@", "1", "5"]
```

FIGURE 10 – Listes des consonnes et voyelles assignées à leur caractéristique.

Ensuite, nous avons défini une nouvelle fonction : **freq_label** qui s'applique sur la liste de fréquences des formes phonétiques préalablement obtenues. Par contre, pour récupérer les 15 structures les plus fréquentes sous ces formes labellisées, il a fallu augmenter le nombre de formes phonétiques recherchées qui, auparavant, était limité à 15.

```
def freq_label(l):
    freq_label = []
    for i in l:
        string = ''
        for j in i[0]:
            for k in j:
                if k in stop_V:
                    string += 'stopV'
```

FIGURE 11 – Fonction *freq_ label* pour le calcul de fréquences.

4 Résultats

Les tableaux 1, 2 et 3 montrent les résultats des 15 plus fréquentes syllabes du français, selon différents paramètres. Premièrement, les formes VC. Deuxièmement, les caractéristiques phonétiques. Troisièmement, les consonnes et voyelles sous leur forme phonétique.

TABLE 1 – Fréquence des structures syllabiques VC.

1	CV
2	CVC
3	CCV
4	V
5	CCVC
6	VC
7	CVCC
8	CVV
9	CCVCC
10	VVV
11	CCCV
12	CCVV
13	CCCVCV
14	CCCVC
15	VCC

Tout d'abord, on peut remarquer que le français se compose plus de syllabes ouvertes (i.e. se terminant par une voyelle) que de syllabes fermées (i.e. se terminant par une consonne). De plus, les premières syllabes les plus fréquentes sont courtes ; elles contiennent seulement 1, 2 ou 3 éléments.

TABLE 2 – Fréquence des structures syllabiques labellisées.

1	stopU vowelsO
2	liquids vowelsO
3	nasals vowelsO
4	stopV vowelsO
5	fricU vowelsO
6	vowelsO
7	fricV vowelsO
8	stopU liquids vowelsO
9	stopU vowelsO liquids
10	vowelsN
11	stopU vowelsN
12	stopV liquids vowelsO
13	nasals vowelsN
14	fricU vowelsO liquids
15	liquids vowelsN

Ensuite, d'après le tableau 2, on peut dire que les structures syllabiques les plus communes se forment principalement de voyelles orales (*/a e i o u y/*) ou encore de

consonnes occlusives voisées et sourdes (*/p t k b d g/*). Les liquides (*/l, R/*) et nasales (*/m n N G/*) apparaissent également souvent.

TABLE 3 – Fréquence des structures syllabiques sous leur forme phonétique.

1	/de/
2	/a/
3	/te/
4	/e/
5	/Ra/
6	/ti/
7	/m@/
8	/@/
9	/Re/
10	/li/
11	/k1/
12	/R*/
13	/5/
14	/se/
15	/si/

Enfin, après avoir observé que les mots et syllabes du français se composent fréquemment de voyelles orales, cette analyse nous permet de voir que */e, i/* et */a/* sont trois des voyelles orales les plus courantes. Aussi, */d/* et */t/* font parties des consonnes occlusives les plus sollicitées par la langue française. */R/* est la nasale que l'on retrouve souvent.

5 Conclusion

Ce projet portait sur la représentation des structures syllabiques de mots d'un corpus en langue française se composant au départ de leurs formes orthographique et phonétique.

Après avoir développé un algorithme permettant cette représentation, en python, nous avons pu extraire les 15 structures syllabiques les plus fréquentes :

1. sous forme VC,
2. sous forme labellisée (i.e. voyelles O, stop U, ...),
3. sous forme phonétique.

Nous avons donc pu constater, entre autres, que les structures syllabiques les plus communes se composent principalement de voyelles orales, ou encore que les consonnes occlusives (*/d/* et */t/*) sont les plus utilisées en français.

Ce projet nous a aidé à mieux comprendre la construction des mots du français. En effet, la syllabification est régie par des règles spécifiques.

Notre projet peut être utilisé pour l'apprentissage du français chez les jeunes enfants, mais aussi, pour des personnes souhaitant apprendre cette langue.