Activité 09 - Fréquence dans des textes

Equipe Pédagogique LU1IN0*1

Consignes : Cette activité se compose d'une première partie guidée, suivie de suggestions. Il est conseillé de traiter en entier la partie guidée avant de choisir une ou plusieurs suggestions à explorer.

L'objectif de cette activité est l'étude de textes à travers la fréquence d'apparition de certains mots ou lettres.

1 Partie Guidée : Occurence des mots

Dans toute l'activité, on étudie le nombre d'occurences (ou la fréquence d'apparition) des mots dans des fichiers textes. Dans un premier temps, on veut ouvrir un fichier texte, récupérer les mots en découpant le fichier selon des symboles de ponctuation, puis identifier minuscules et majuscules.

On rappelle que les fichiers textes peuvent être ouverts par Python en utilisant la fonction suivante :

```
def ouvre_fichier(nom : str) -> List[str] :
    """ renvoie la liste des lignes du fichier texte ./nom.txt """
    with open("./"+nom+".txt", "r", encoding = "utf-8") as f:
        return f.readlines()
```

Par exemple, l'ouverture du fichier `beaute.txt` (disponible sur Moodle), donne :

```
>>> ouvre_fichier("beaute")
['Je suis belle, o mortels ! comme un reve de pierre,\n',
 "Et mon sein, ou chacun s'est meurtri tour a tour,\n",
 'Est fait pour inspirer au poete un amour\n',
 'Eternel et muet ainsi que la matiere.\n',
 '\n',
 "Je trone dans l'azur comme un sphinx incompris ;\n",
 "J'unis un coeur de neige a la blancheur des cygnes ;\n",
 'Je hais le mouvement qui deplace les lignes,\n',
 'Et jamais je ne pleure et jamais je ne ris.\n',
 '\n',
 'Les poetes, devant mes grandes attitudes,\n',
 "Que j'ai l'air d'emprunter aux plus fiers monuments,\n",
 "Consumeront leurs jours en d'austeres etudes ;\n",
 '\n',
 "Car j'ai, pour fasciner ces dociles amants,\n",
 'De purs miroirs qui font toutes choses plus belles :\n',
 'Mes yeux, mes larges yeux aux clartes eternelles !']
```

On constate que les lignes du fichiers sont lues en chaînes de caractères, qui apparaissent, dans l'ordre, comme éléments d'une même liste. Chacune de ces chaînes finit par le caractère *retour chariot* `"\n"`.

Question 1. Ecrire une fonction `decompose_ligne`, qui prend en entrée une ligne `li` de fichier .txt (donc une chaine de caractères) et un **ensemble de** caractères `sep` qui renvoie une liste de chaînes de caractères, correspondant au découpage de `li` selon le caractère `sep`. C'est-à-dire qu'on récupère dans la liste résultat les différentes sous-chaînes **non vides** de la ligne qui se trouvent entre des éléments de `sep`. En outre, on fera en sorte de supprimer le dernier caractère de `li` (le retour chariot `"\n"`) dans la liste résultat.

Par exemple avec l'ensemble de caractères de séparation suivant :

```
ponctuation : Set[str] = {" ", ",", ";", "'", "(", ")", ".", "!", "?", ":"}
```

On aura les résultats suivants :

```
>>> decompose_ligne(exemple1[0], ponctuation)
```

```
['Je', 'suis', 'belle', 'o', 'mortels', 'comme', 'un', 'reve', 'de', 'pierre']
>>> decompose_ligne(exemple1[4], ponctuation)
[]
>>> decompose_ligne(exemple1[8], ponctuation)
['Et', 'jamais', 'je', 'ne', 'pleure', 'et', 'jamais', 'je', 'ne', 'ris']
```

Question 2. Ecrire une fonction `minusculise` qui prend en entrée un mot (une chaine de caractères) `m` et qui renvoie une chaîne correspondant à `m` dans lequel toutes les lettres romaines majuscules sont converties en minuscule.

```
>>> minusculise("bonjour")
"bonjour"
>>> minusculise("BONJOUR")
"bonjour"
>>> minusculise("Bonjour")
"bonjour"
```

Question 3. Donner une définition **avec compréhension** d'une fonction `mots` qui prend en entrée une liste de chaînes `lis` correspondant à la lecture d'un fichier texte et un ensemble `sep` de caractères de séparation, et qui renvoie la liste des mots de `lis`, c'est à dire la liste des sous-chaînes séparées par des éléments de `set` des éléments de `lis`, converties en minuscules.

Par exemple (dans ce test, on regarde uniquement les 15 premiers mots du résultat) :

```
>>> mots(exemple1, ponctuation)[:15]
['je', 'suis', 'belle', 'o', 'mortels', 'comme', 'un',
 'reve', 'de', 'pierre', 'et', 'mon', 'sein', 'ou', 'chacun']
```

Question 4. Ecrire une fonction `dictionnaire_occ_mots` qui prend en entrée une liste de mots `ms`, et qui construit le dictionnaire des occurences des mots de `ms`, c'est-à-dire le dictionnaire où les clefs sont les mots de `ms` et la valeur associée à un mot et son nombre d'occurences dans `ms`.

```
dico1 : Dict[str, int] = dictionnaire_occ_mots(mots(exemple1, ponctuation))
>>> dico1["je"]
5
>>> dico1["belle"]
1
>>> dico1["jamais"]
2
```

Question 5. Un *hapax* est un mot qui apparaît exactement une fois dans un texte.

Donner une définition **par compréhension** d'une fonction `hapax` qui prend en entrée un dictionnaire d'occurences de mots `occ` et renvoie l'ensemble des hapax de `occ`.

```
>>> len(hapax(dico1))
67
>>> ("sphinx" in hapax(dico1))
True
>>> ("jamais" in hapax(dico1))
False
```

Il y a 67 mots qui apparaissent exactement une fois dans *La Beauté* dont `sphinx` mais pas `jamais`.

Question 6. Ecrire une fonction `plus_frequent` qui prend en entrée un dictionnaire d'occurences de mots `occ` et renvoie un des mots (le premier dans l'ordre du dictionnaire) avec le nombre d'occurences le plus élevé.

```
>>> plus_frequent(dico1)
"je"
```

2 Suggestion : Comparaison de fréquences

On peut se servir des fréquences d'apparition des mots pour comparer et identifier des textes.

La *fréquence* du mot `m` dans la liste de mots `ms` est donné par $\frac{o}{l}$ où o est le nombre d'occurences de `m` dans `ms` et l la longueur de `ms`.

Question 1. Ecrire une fonction `dictionnaire_freq_mots` qui prend en entrée une liste de mots `ms` et construit le dictionnaire des fréquence des mots de `ms`

```
dico2 : Dict[str, float] = dictionnaire_freq_mots(mots(exemple1, ponctuation))
>>> dico2["je"]
0.040650406504065
>>> dico2["belle"]
0.0081300813008130
```

Question 2. Ecrire une fonction `distance_freq` qui prend en entrée deux dictionnaires de fréquence `d1` et `d2` et calcule la *distance de* `d2` *à* `d1` selon le modèle suivant :

- On commence avec un score de 100.
- On parcourt les mots de `d2`.
- Pour chaque mot `m` de `d2`, s'il apparait dans `d1`, on calcule $100 * |\texttt{f1} - \texttt{f2}|$ où `f1` est la fréquence de `m` dans `d1` et `f2` sa fréquence dans `d2` et on retranche ce résultat au score.
- S'il n'apparait pas dans `d1`, on calcule $100 * \texttt{f1}$ où `f1` est la fréquence de `m` dans `d1` et on retranche ce résultat au score.
- le score restant après ce parcours est la distance de `d2` à `d1`.

3 Suggestion : Auteur d'un texte

Question 1. Ecrire une fonction `jointure_dict_freq` qui prend en entrée deux dictionnaires de fréquences `d1` et `d2` et deux entiers `l1` et `l2` et qui renvoie un dictionnaire de fréquence `d` qui joint `d1` et `d2` en un unique dictionnaire, en considerant que `d1` s'applique à un texte de `l1` mots et `d2` à un texte de `l2` mots.

Si `m` apparait avec une fréquence `f1` dans `d1` et `f2` dans `d2`, il doit apparaitre avec une fréquence $\frac{\texttt{f1.l1+f2.l2}}{\texttt{l1+l2}}$ dans `d`.

Si `m` apparait avec une fréquence `f1` dans `d1` et n'apparait pas dans `d2`, il doit apparaitre avec une fréquence $\frac{\texttt{f1.l1}}{\texttt{l1+l2}}$ dans `d`. Et réciproquement.

Question 2. Ecrire une fonction `dict_auteur` qui prend entrée une liste de titre de fichier texte `li` et qui construit le dictionnaire de fréquence des mots apparaissant dans les fichiers dont le nom apparait dans `li`.

Par exemple on pourra définir :

```
baudelaire : Dict[str, float] = dict_auteur(["beaute", "spleen", "albatros"])
```

(les fichiers `beaute.txt`, `spleen.txt` et `albatros.txt` sont disponibles dans Moodle.)

Question 3. Ecrire une fonction `auteur` qui prend en entrée une liste `candidats` de couples `(nom, dico)` où `nom` est un nom d'auteur et `dico` un dictionnaire de fréquences d'apparation de mots dans les oeuvres de `nom`, et une chaîne `titre` correspondant au nom d'un fichier texte contenant un texte d'auteur inconnu, et qui renvoie le nom de l'auteur de `candidats` le plus probable pour le fichier `titre.txt`, c'est-à-dire le nom de l'auteur dont le score de distance de la fréquence des mots du fichier au dictionnaire de fréquence de l'auteur est maximal.

4 Suggestion : Attaque de César

Vu a l'activité 4, le chiffre de César consiste à décaler toutes les lettres d'un texte de n places dans l'alphabet. L'attaque de ce chiffre proposée consistait à afficher 26 décalages différents d'un extrait du

texte chiffré et de laisser un cerveau humain décider quel décalage produit un texte intelligible.

Le but est ici d'automatiser cette attaque (de se passer de l'intervention du cerveau humain) en devinant quel décalage produit le texte intelligible.

La clef de cette opération est l'utilisation de dictionnaires de fréquence propres à chaque langue[1]. Pour chaque décalage possible (entre 0 et 25), on compare le dictionnaire de fréquence d'apparition des lettres du texte obtenu au dictionnaire de fréquence d'apparition des lettres en français et on sélectionne le décalage avec le meilleur score de distance.

Un dictionnaire de fréquence pour le français est disponible dans le modèle sur Moodle :

```
lettres_francais : Dict[str, float] = {"e" : 0.1210,
    "a" : 0.0711,
    "i" : 0.0659,
    "s" : 0.0651,
    "n" : 0.0639,
    "r" : 0.0607,
    "t" : 0.0592,
    "o" : 0.0502,
    "l" : 0.0496,
    "u" : 0.0449,
    "d" : 0.0367,
    "c" : 0.0318,
    "m" : 0.0262,
    "p" : 0.0249,
    "g" : 0.0123,
    "b" : 0.0114,
    "v" : 0.0111,
    "h" : 0.0111,
    "f" : 0.0111,
    "q" : 0.0065,
    "y" : 0.0046,
    "x" : 0.0038,
    "j" : 0.0034,
    "k" : 0.0029,
    "w" : 0.0017,
    "z" : 0.0015}
```

Ecrire une fonction `decode_cesar_auto` qui prend en entrée le nom d'un fichier encodé avec le chiffre de César et écrit dans un fichier le résultat du décodage automatique.

5 Suggestion : Attaque de Vigenère

Le chiffre de Vigenère (cf. Activité 4) est plus difficile à attaquer car il est polyalphabétique (une même lettre peut être chiffrée en des lettres différentes, selon sa place dans le texte).

Si on connait la longueur de la clef, on peut se ramener à l'attaque du chiffre de César : si la clef est de longueur `c`, on découpe le texte initial en `c` textes différents, le premier texte contient les lettres d'indice `0`, `c`, `2*c`, ..., le deuxième texte les lettres d'indice `1`, `c + 1`, `2* c + 1`, ... et ainsi de suite.

Chaque texte découpé a été décalé avec la même lettre, on peut donc appliquer l'attaque de César, indépendamment, à chaque texte découpé, puis rassembler ensuite les résultats.

Découvrir la taille d'une clef inconnue est par contre plus difficile, et nécessite une analyse complexe des *périodes* avec lesquelles certains motifs se répètent. On pourra trouver plus d'information ici :

`https://fr.wikipedia.org/wiki/Cryptanalyse_du_chiffre_de_Vigen%C3%A8re`

6 Suggestion : Autres idées

On pourra, bien sûr, imaginer d'autres analyses de textes basées sur la fréquence d'apparition des lettres ou des mots dans un texte (par exemple, deviner la langue d'un texte en utilisant *l'indice de coïncidence*[2]).

[1] `https://fr.wikipedia.org/wiki/Fr%C3%A9quence_d%27apparition_des_lettres_en_fran%C3%A7ais`
[2] `https://fr.wikipedia.org/wiki/Indice_de_co%C3%AFncidence`