TP2 – IFT2015 – H18 – Major – Texte mystère

Certains d'entre vous connaissent peut-être le roman *Democracy: An American Novel*. C'est un roman publié en 1880 par un auteur initialement anonyme qui fut très populaire aux États-Unis. À l'époque, beaucoup de gens ont essayé d'identifier l'auteur par toutes sortes d'analyses comparatives du texte et d'autres textes d'auteurs connus. Finalement, l'identité de l'auteur véritable a été révélée après sa mort; il s'agissait de Henry Adams.

Pour ce TP, vous allez développer une méthode d'analyse de proximité entre deux textes et l'utiliser pour identifier l'auteur d'un texte mystère.

Données fournies

Vous aurez accès à plusieurs romans d'auteurs français pour lesquels il n'y a plus de droits d'auteur. Les auteurs sont : Jules Verne, Victor Hugo, Honoré de Balzac, Voltaire, la Comtesse de Ségur et Émile Zola. Ce sont ces textes que vous devrez traiter pour en extraire une signature qui vous servira à les comparer entre eux et à comparer le texte mystère à la signature de chaque auteur. Tous les romans d'un même auteur seront regroupés dans un seul fichier texte.

Signature

La signature que vous allez calculer pour ce TP sera basée sur la fréquence des doublets de mots. Un doublet de mots est simplement la suite de deux mots. Par exemple, dans la phrase suivante :

```
Le chien mange le chat et le chat mange la souris.
```

On retrouve les doublets :

Doublet	Fréquence
Le chien	1
chien mange	1
mange le	1
le chat	2
chat et	1
et le	1
chat mange	1
mange la	1
la souris.	1

Dans cet exemple, on conserve la ponctuation qui fait partie des mots. Aussi, vous remarquerez qu'on obtient des doublets comme « et le » qui se retrouveront probablement

dans la majorité des textes, peu importe leur auteur. Pour augmenter la spécificité de notre signature, nous allons donc filtrer les textes comme suit.

Filtration du texte
Pour augmenter la spécificité de votre signature, vous allez pré-traiter le texte de façon à :

- retirer la ponctuation
- retirer les mots de deux lettres ou moins
- transformer toutes les lettres en minuscules

Comme cet exercice n'est pas très excitant, je l'ai déjà fait pour vous. Vous trouverez une méthode `treatLine` qui fait tout ça et le début d'une méthode `treatText` qui appelle `treatLine` dans le fichier **tp2_code.py**. Je vous suggère fortement d'utiliser la méthode `treatLine` telle quelle. Vous vous assurerez ainsi que votre filtration du texte soit standard. Si on reprend la phrase donnée en exemple ci-dessus, elle contiendrait maintenant les doublets :

Doublet	Fréquence
chien mange	1
mange chat	1
chat chat	1
chat mange	1
mange souris	1

Distance entre deux signatures
Pour calculer la distance entre deux signatures, il faut d'abord trouver l'ensemble des doublets qui sont présents dans les deux signatures. Puis on normalise les fréquences de ces doublets communs pour chacune des signatures. Cela revient à diviser le nombre de doublets observés dans chaque signature par le nombre total d'observations qui correspondent à un doublet commun, pour chaque signature. Puis, on calcule ensuite le « root-mean-square deviation » ou RMSD entre ces signatures normalisées, donné par la formule :

$$RMSD = \sqrt{\frac{\sum_{doublets}(s_i[doublet] - s_j[doublet])^2}{n}}$$

où $s_i[doublet]$ est la fréquence normalisée du doublet dans la signature s_i et n est le nombre total de doublets. Pour ceux qui préféreraient voir du code, une méthode `dist_between_ds` qui calcule le RMSD entre deux dictionnaires est donnée en appendice.

Vous pouvez vous en inspirer autant que vous voulez, mais lisez bien la section qui suit; vous devez implanter **votre propre dictionnaire**.

Table de hachage

Vous devrez faire vous-même l'implantation de votre dictionnaire en utilisant une table de hachage. Cela veut donc dire que vous ne pourrez **pas utiliser dict()** qui est fourni par Python, ni **aucun objet/fonction de type abstrait dictionnaire** provenant de Python ou d'un module externe. Vous devrez aussi implanter votre propre fonction de hachage et ne devrez **pas utiliser la fonction hash de Python**. Vous implanterez donc une classe HashTable à l'intérieur de laquelle vous définirez les méthodes suivantes :

- __getitem__(*self,* key) : retourne la valeur associée à l'index correspondant à key
- __setitem__(*self,* key, value) : insère la valeur value à l'index correspondant à key
- __delitem__(*self,* key) : supprime la valeur à l'index correspondant à key
- autres méthodes internes pour votre fonction de hachage (voir plus bas)

Le constructeur de votre classe devra prendre la taille du tableau initial en entrée et vous devrez implanter une méthode de réajustement de la taille du tableau.

Stratégie de hachage

Vous pouvez vous baser sur les diapositives ***Tables de hachage (Hashing)*** vues en classe pour la sélection d'une stratégie de hachage et de gestion des collisions. La plus simple est la table de hachage avec listes chaînées, mais vous pouvez utiliser les autres aussi. Pour le code de hachage, vous pouvez essayer différentes stratégies. La plus simple consiste à simplement considérer la chaîne de caractères à traiter comme un entier, mais elle engendrera beaucoup de collisions. La somme polynomiale (**p. 8** des diapos, « polynomial accumulation » en anglais) est très efficace pour les chaînes de caractères.

Entrée

Votre programme doit comprendre un fichier `tp2_matricule1_matricule2.py` qui sera **exécuté dans un dossier**. Ce dossier contiendra un texte par auteur ainsi qu'un texte mystère. Voici les noms de fichiers qui seront utilisés :

```
verne.txt       hugo.txt
segur.txt       voltaire.txt
zola.txt        balzac.txt
mystere.txt
```

Vous devrez créer une signature pour chacun de ces textes, puis calculer la distance entre la signature mystère et celle de chacun des auteurs (donc six distances en tout à calculer). Vous afficherez les distances et l'auteur le plus probable du texte mystère (celui de distance minimale). Cette fois, votre programme ne prendra **aucun argument de ligne de commande.**

Sortie

Vous produirez 7 lignes en sortie. Les 6 premières lignes contiendront le nom de chaque auteur et la distance entre le texte de cet auteur et le texte mystère. La septième ligne donnera le nom de l'auteur le plus probable du texte mystère. Voici un exemple de sortie :

```
Verne 0.0012
Zola 0.0005
Balzac 0.0001
Hugo 0.0005
Segur 0.0021
Voltaire 0.0023
Auteur du texte mystère : Balzac
```

Structure du code

Avertissement : vous devez développer VOTRE PROPRE CODE et tout plagiat détecté entraînera automatiquement un échec.

Votre code doit être clair et bien documenté. Vous êtes encouragés à consulter PEP 8 qui est un guide de style pour Python à la page suivante : https://www.python.org/dev/peps/pep-0008/

Version de Python

Il est fortement recommandé d'utiliser Python 3.6.4 ou toute autre version 3.6.*X*. N'utilisez pas Python 2 car des problèmes de compatibilité pourraient survenir.

Équipes

Vous pouvez faire votre travail seul ou en équipes de **deux** personnes **maximum**. Vous pouvez discuter avec les autres équipes évidemment, mais tout code dupliqué entre deux équipes sera considéré comme un plagiat des deux côtés.

Remise

Vous avez jusqu'au **15 avril à 23h55** pour remettre votre travail sur Studium. Remettez votre travail soit sous forme d'un seul fichier `tp2_matricule1_matricule2.py` ou sous la forme d'une archive `tp2_matricule1_matricule2.tar.gz` ou `tp2_matricule1_matricule2.zip` (**aucune** autre forme de compression ne sera tolérée) si vous avez plus d'un fichier. Dans ce cas, l'archive sera décompressée dans le dossier contenant les fichiers d'entrée et devra contenir entre autres le fichier `tp2_matricule1_matricule2.py` qui sera exécuté.

Correction négative

- 50% des points peuvent être perdus si votre code ne s'exécute pas ou ne produit pas de sortie
- 50% des points peuvent être perdus si vous n'avez **pas implanté** votre **propre table de hachage** ou votre **propre fonction de hachage**
- Une pénalité de **20% par jour de retard** sera appliquée dès la première seconde de chaque période de 24h, en commençant à 23h56 le 15 avril

Correction positive

- 30% : clarté du code
- 10% : respect du format de sortie
- 20% : implantation de la table de hachage
- 40% : tests d'identification de textes mystères

Points boni

Comme pour le TP1, 10% supplémentaires seront accordés en fonction de la vitesse d'exécution (par rapport à la moyenne) de votre code :
Code rapide : +4%
Code très rapide : +7%
Code extrêmement rapide : +10%

Questions

Envoyez vos questions sur le travail à omailhot92@gmail.com.

Bon travail !!!

Appendice

```
def dist_between_ds(d1, d2):
    """
    Returns the RMSD of normalized frequencies between two dicts storing the
    frequency as the value.
    """
    commond = dict()
    thisn = 0
    othern = 0
    dist = 0
    for k in d1.keys():
        x = d1[k]
        if k in d2:
            y = d2[k]
            commond[k] = (x, y)
            thisn += x
            othern += y
    for k in d2.keys():
        if k in commond:
            continue
        y = d2[k]
        if k in d1:
            x = d1[k]
            commond[k] = (x, y)
            thisn += x
            othern += y
    for k in commond.keys():
        x = commond[k]
        dist += (x[0] / thisn - x[1] / othern) ** 2
    dist = dist / len(commond.keys())
    dist = math.sqrt(dist)
    return dist
```