Détection de langue

Thèmes : algorithmique du texte, dictionnaires, apprentissage supervisé, algorithme des k plus proches voisins

On souhaite réaliser un programme permettant de déterminer la langue utilisée dans une phrase tapée par l'utilisateur. On utilisera pou cela le principe de l'apprentissage supervisé.

On procèdera en 3 étapes :

- On utilise un texte écrit en langue française (*Les Misérables*, Victor Hugo) et un texte écrit en langue anglaise (*Une Étude en Rouge*, Conan Doyle) pour extraire un ensemble de phrases étiquetées par leur langue (français ou anglais).
- On étudie les bigrammes présents dans ces phrases, ce qui permet d'associer à chaque phrase un vecteur de l'espace vectoriel $\mathcal{M}_{26}(\mathbb{R})$.
- On utilise l'algorithme des k plus proches voisins pour déterminer l'étiquette d'une phrase tapée par l'utilisateur.

1 Extraction du corpus

Le corpus utilisé est donné sous forme de deux fichiers textes `miserables.txt` et scarlet.txt.

On considère la fonction de lecture suivante :

```
"""Prend en entrée un nom de fichier et retourne une liste
de chaînes de caractères correspondant aux phrases lues
dans le fichier."""
def lire_phrases(nom_fichier):
    phrases = None
    with open(nom_fichier, 'r') as fichier:
        contenu = fichier.read()
        # Passe tout en minuscules
        contenu = contenu.lower()
        # Efface certains caractères du texte
        contenu = re.sub('[\n\"\'\\-\,:;\\(\\)\\_«»]', ' ', contenu)
        # Efface les espaces superflus
        contenu = re.sub(' +', ' ', contenu)
        # Enleve les accents de la langue française
        contenu = re.sub('[éèê]', 'e', contenu)
        # Coupe le texte en phrases selon le caractère . ! ou ?
```

```
        phrases = re.split('[\.\\?!]', contenu)
        # Efface les espaces en trop au debut et a la fin
        phrases = [p.strip() for p in phrases if len(p) > 5]
        fichier.close()
    return phrases
```

Question 1

Compléter la fonction `lire_phrases` pour qu'elle élimine tous les accents de la langue française, c'est-à-dire : à, â, ô, ù, î, et ç.

Question 2

Utiliser votre programme pour déterminer le nombre de phrases dans chacun des fichiers donnés.

2 Bigrammes d'une phrase

On définira les variables globales suivantes :

```
p1 = lire_phrases('miserables.txt')
p2 = lire_phrases('scarlet.txt')
alpha = "abcdefghijklmnopqrstuvwxyz"
```

Un bigramme dans un texte est une suite de 2 lettres consécutives. Par exemple : `to be or not to be` contient les bigrammes `to be or no ot`. Un bigramme peut apparaître plusieurs fois : il y a 2 occurrences du bigramme `to`.

Si p est une phrase, on appelle *matrice de bigrammes* de p une matrice $M \in \mathcal{M}_{26}(\mathbb{R})$, où les lignes et colonnes sont indexées par les lettres de l'alphabet et où $M_{u,v}$ contient le nombre d'occurrences du bigramme uv dans p. Cette matrice récapitule donc le nombre d'occurrences dans la phrase donnée de chacun des diagrammes possibles.

Question 3

Écrire une fonction `bigramme(phrase)` prenant en entrée une phrase et retournant sa matrice de bigrammes M. Cette matrice prendra la forme d'un dictionnaire indicé sur des couples de lettres telle que $M[u,v]$ est le nombre d'occurrences du bigramme uv. Le dictionnaire devra avoir une valeur, éventuellement nulle, pour tout couple de lettres (u,v).

INDICATIONS

Parcourir les suites de deux caractères consécutifs de `phrase` et lorsque ces caractères sont tous les deux des lettres incrémenter la valeur correspondante dans la matrice de bigrammes.

Question 4

Écrire une fonction `taille_bigramme(b)` retournant le nombre d'occurrences de bigramme d'une phrase à partir de sa matrice de bigrammes `b`.

Question 5

Écrire une fonction `dist(A, B)` calculant la distance euclidienne entre deux matrices de bigrammes `A` et `B`. Cette distance est donnée par

$$d(A,B) = \sqrt{\sum_{u \in \text{alpha}} \sum_{v \in \text{alpha}} \left(A_{u,v} - B_{u,v}\right)^2}$$

3 Apprentissage supervisé

Question 6

Écrire une fonction `plusprochevoisin(phrase, p1, p2)` retournant un triplet `(d, ppv, langue)` où `ppv` est la phrase de `p1` ou `p2` la plus proche de `phrase`, `d` est la distance correspondante et `langue` est la langue de `ppv`.

Question 7

En déduire un programme demandant à l'utilisateur de taper une phrase, puis qui affiche la phrase du corpus la plus proche de la phrase tapée. On affichera également la langue détectée.

On souhaite évaluer notre méthode de détection. Pour cela on extrait un petit nombre de phrases `q1` depuis `p1` et `q2` depuis `p2`. Puis, on se serivra de `q1` et `q2` pour obtenir une matrice de confusion.

Question 8

Écrire une fonction `extraire(l,n)` retournant un couple de listes `q, r` où `q` contient `n` éléments extraits de la liste `l`. La liste `r` est la liste des éléments restants, elle est donc de taille `len(l) - n`.

Question 9

1. Utiliser extraire sur `p1` et `p2` pour extraire dans chaque texte environ 100 phrases.
2. Détecter la langue des éléments de `q1` et `q2` en appliquant la méthode précédente sur l'ensemble d'apprentissage `r1` et `r2`. On conservera les résultats dans une **matrice de confusion**.
3. Calculer le taux d'erreur obtenu.

4 Algorithme des k plus proches voisins

Question 10

Écrire une fonction `kppv(phrase, p1, p2, k)` retournant une liste de k triplets `(d, p, l)` correspondant aux k phrases de `p1` ou `p2` les plus proches de `phrase`. Dans le triplet `p` est la phrase, `d` est la distance entre `p` et `phrase` et `l` est la langue de `p`.

Question 11

En déduire une fonction `detecte(phrase, k)` détectant la langue d'une phrase tapée par la méthode des k plus proches voisins.

Question 12

Construire et afficher pour certaines valeurs de k, la matrice de confusion obtenue par la méthode des K plus proches voisins.