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 pour cela le principe de l'apprentissage supervisé.

On procédera en trois étapes :

a) 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) ;
b) on étudie les bigrammes présents dans ces phrases, ce qui permet d'associer à chaque phrase un vecteur de l'espace vectoriel $\mathcal{M}_{26(\mathbb{R})}$;
c) 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 texte `miserables.txt` et `scarlet.txt`.

On considère la fonction de lecture suivante, qui nécessite d'importer le module `re` et que l'on téléchargera directement sur le Cahier de prépa de la classe :

```
def lire_phrases(nom_fichier: str) -> list:
    """Prend en entrée un nom de fichier et renvoie une liste
    de chaînes de caractères correspondant aux phrases lues
    dans le 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)
        # Enlève 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 début 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, ï, ü, etc.

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 apparaitre plusieurs fois : il y a deux 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 renvoyant sa matrice de bigrammes. Cette matrice M prendra la forme d'un dictionnaire indicé sur des couples de lettres tel 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).

Indication. 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 `dist(A, B)` calculant la distance euclidienne entre deux matrices de bigrammes `A` et `B`. Cette distance est définie par

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

3 Apprentissage supervisé avec 1NN

Question 5. Écrire une fonction `plusprochevoisin(phrase, p1, p2)` renvoyant un triplet `(d, ppv, langue)` où `ppv` est une phrase de `p1` ou de `p2` qui soit la plus proche possible de `phrase`, où `d` est la distance correspondante et où `langue` est la langue de `ppv`.

Question 6. En déduire un programme demandant à l'utilisateur de saisir une phrase, puis qui affiche la phrase du corpus la plus proche de la phrase saisie. 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 sert de `q1` et `q2` pour obtenir une matrice de confusion.

Question 7. Écrire une fonction `extraire(l, n)` renvoyant un couple de listes `(q, r)` où `q` contient `n` éléments extraits de la liste `l` et où la liste `r` est la liste des éléments restants, qui est donc de taille `len(l) - n`. *On pourra utiliser la fonction* `sample` *du module* `random`.

Question 8.

a) Utiliser `extraire` sur `p1` et `p2` pour extraire dans chaque texte environ 100 phrases.
b) 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.
c) Calculer le taux d'erreur obtenu.

4 Algorithme des k plus proches voisins

Question 9. Écrire une fonction `kppv(phrase, p1, p2, k)` renvoyant une liste de k triplets `(d, p, l)` correspondant aux k phrases de `p1` ou `p2` les plus proches possible 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 10. En déduire une fonction `détecte(phrase, p1, p2, k)` détectant, par la méthode des k plus proches voisins, la langue d'une phrase saisie.

Question 11. Construire et afficher pour certaines valeurs de k la matrice de confusion obtenue par la méthode des k plus proches voisins.