Ecole **N**ationale des **S**ciences **A**ppliquées - **A**l Hoceima

Travaux pratiques N°1 : L'algorithme Minimum Edit Distance et Autocorrection

Module : Traitement Automatique de la Langue Naturelle (NLP)

Deuxième Année Ingénierie des données

- Ce TP est à réaliser en binômes et en utilisant le langage Python.
- Le compte rendu est à rendre sous forme d'un jupyter notebook avant le 20/Mars/2024

Exercice 1

Pour cet exercice, tester toutes les expressions régulières avec des programmes Python sur des textes de votre choix.

1. Ecrire une expression régulière pour vérifier qu'une chaîne de caractères contient uniquement des caractères alphanumériques.
2. Ecrire une expression régulière pour trouver toutes les occurrences de l'article « *le* » dans un texte.
3. Ecrire une expression régulière qui remplace toutes les occurrences des noms des mois ou des jours de la semaine par <*DATE*>.
4. Ecrire et tester les expressions régulières pour extraire:
   a. L'ensemble de tous les mots minuscules se terminant par un b;
   b. L'ensemble de toutes les chaînes avec deux mots répétés consécutifs (e.g., "Cours COURS" et "Le le")
5. Ecrire une expression régulière pour extraire tous les mots qui commencent au début de la ligne par un 'b' ou qui se terminent à la fin de la ligne par un 'b'.
6. Ecrire une expression régulière pour extraire l'ensemble de tous les nombres en chiffres (e.g. 125 ; 12.22 ; 1.22 ; 1,23 ; ٥٧, …) et les normaliser :
   125,25 → 125.25
   ٥,٥ → 5.5
   15.5 →15.50
7. Ecrire une expression régulière pour extraire toutes les dates à partir d'un texte puis les normaliser sous format *dd/mm/yyyy*. On se limite aux dates de forme *dd/mm/yyyy*, *dd/mm/yy*, *dd-mm-yyyy*, ou *dd-mm-yy*.

Exercice 2

1- Implémenter l'algorithme de distance d'édition minimale en complétant le code donné dans le fichier *emd.py*

2- Enrichissez votre fonction de distance d'édition pour également enregistrer la rétrogradation (*backtrace*) comme décrit dans le cours. Utilisez cette rétrogradation pour imprimer un alignement entre les chaînes de caractères.

3- Coûts d'adjacence. Les coûts dans ce calcul sont finalement des décisions arbitraires que nous avons le droit de prendre. Essayez de modifier les fonctions de coût pour calculer une version plus intelligente de la distance d'édition qui pourrait être applicable à la correction orthographique, ou du moins qui pourrait mieux gérer le texte désordonné du web. Il existe plusieurs façons de le faire. On pourrait dire, par exemple, que le coût de substitution pour frapper une touche adjacente est plus bas que le coût de frapper une touche non adjacente. En

supposant un clavier QWERTY vous pouvez utiliser les données indiquées dans le fichier *qwerty_graph.txt*

4- Implémenter votre première version d'un correcteur d'orthographe en se basant sur l'algorithme simple suivant :
   Soit w un mot contenant une erreur d'orthographe, calculer la distance entre w et tous les mot d'un dictionnaire contenant les mots valides puis proposer les k corrections possibles (les k plus proches mot selon la distance d'édition minimale).
5- Optimiser l'algorithme précédent en prenant en compte les remarques suivantes :
   - Les gens ont généralement la première lettre du mot correcte, donc nous pouvons limiter notre recherche aux mots commençant par la même lettre.
   - Nous pouvons restreindre notre recherche aux mots de même longueur ou de longueur similaire.
   - Nous pouvons restreindre notre recherche aux mots qui ont le même son, en utilisant un code phonétique pour regrouper les mots (comme Soundex).
   - En utilisant d'autres techniques de votre choix

Exercice 3

On considère le problème de la détermination de la correction la plus probable d'un mot introuvable en dictionnaire. Ainsi, le problème est de trouver la correction c, parmi toutes les corrections candidates possibles, qui maximise la probabilité que *c* soit la correction voulue, étant donné le mot original *w* : $argmax_{c \in candidates} P(c|w)$. Par le théorème de Bayes, cela équivaut à :

$argmax_{c \in candidates} P(c)\ P(w|c) / P(w)$

Étant donné que P(w) est la même pour chaque candidat possible *c*, nous pouvons l'éliminer, ce qui donne : $argmax_{c \in candidates} P(c)\ P(w|c)$

Les éléments de cette équations sont :
Modèle de langage : *P(c)* La probabilité que *c* apparaisse en tant que mot dans un texte en anglais.
Modèle d'erreur : *P(w|c)* La probabilité que *w* soit tapé dans un texte lorsque l'auteur voulait écrire *c*. Par exemple, P(teh|the) est relativement élevée, mais P(theeexyz|the) serait très faible.

1- Implémenter la fonction *process_data(corpus_file)* qui permet de lire le corpus donné sous forme d'un fichier texte, convertir le texte en minuscule et segmenter le texte et retourne la liste des mots.
2- Implémenter la fonction *get_vocabulary(corpus_file)* qui retourne le vocabulaire construit à partir d'un corpus passé en argument de la fonction.
3- Nous pouvons estimer la probabilité d'un mot, en comptant le nombre de fois où ce mot apparaît dans un corpus de texte de grande taille et en divisant sur la taille de corpus. Ecrire une fonction qui construit le modèle de langue en calculant la probabilité de chaque mot en se basant sur le fichier *big.txt* fournit avec ce TP et elle stocke le résultat dans une structure de données adéquate. N'oubliez pas d'effectuer le preprocessing avec la fonction *process_data.*
4- Ecrire les fonctions suivantes :
   - *edits1(s) :* qui retourne l'ensemble de toutes les chaînes (qu'il s'agisse de mots ou non) qui peuvent être obtenues avec une seule modification (insertion, substitution ou suppression) à effectuer sur la chaîne s.

- *edits2(s) :* qui retourne l'ensemble de toutes les chaînes (qu'il s'agisse de mots ou non) qui peuvent être obtenues avec 2 modifications (insertion, substitution ou suppression) à effectuer sur la chaîne s.
- *knownWord(words)* qui filtre les mots de la liste *words* qui ne sont pas dans le dictionnaire (elle garde donc que les mots valides). On peut utiliser la fonction *get_vocabulary(corpus_file)*

5- On suppose que nous n'avons pas des données pour construire le modèle d'erreur, ainsi nous allons adopter les hypothèses suivantes: tous les mots connus avec une distance d'édition de 1 sont infiniment plus probables que les mots connus avec une distance d'édition de 2, et infiniment moins probables qu'un mot connu avec une distance d'édition de 0. Ainsi, pour sélectionner les candidats les plus probables nous considérons leurs probabilités selon le modèle de langue préalablement construit et leurs priorités en fonction de leur distance d'édition du mot original. Avec cette simplification, nous n'avons pas besoin de multiplier par un facteur P(w|c), car chaque candidat à la priorité choisie aura la même probabilité.
Ecrire la fonctions *candidates(word)* qui retourne la première liste non vide de candidats par ordre de priorité :
   Le mot original, s'il est connu ; sinon
   La liste des mots connus à une distance d'édition de un, s'il y en a ; sinon
   La liste des mots connus à une distance d'édition de deux, s'il y en a ; sinon
   Le mot original, même s'il n'est pas connu.

6- En utilisant les fonctions précédente, écrire la fonctions *correction(word, k)* qui retourne les *k* plus probables corrections du mot *word.*

Références :

Speech and Language Processing : https://web.stanford.edu/~jurafsky/slp3/2.pdf
Peter norvig : https://norvig.com/spell-correct.html