

# Projet SEO

---

## Rapport

Antoine DEBRENNE (debren\_a)

Zinedine REBIAI (rebiai\_z)

Carine ZHANG (zhang\_d)

13 Juillet 2018

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Présentation générale du projet</b>	<b>3</b>
2.1	Présentation . . . . .	3
2.2	Exécution . . . . .	3
<b>3</b>	<b>Le projet</b>	<b>4</b>
3.1	Extraction des n-grammes . . . . .	4
3.1.1	N-grammes de caractères . . . . .	4
3.1.2	N-grammes de mots . . . . .	4
3.2	Détection de la langue . . . . .	4
3.3	Aide à la rédaction . . . . .	5
3.3.1	Point sur la TF-IDF . . . . .	5
3.3.2	Calcul de la tf-idf sur des documents en entrée . . . . .	6
3.3.3	Auto-complétion d'une phrase . . . . .	6
3.3.4	Auto-complétion d'un mot . . . . .	7
<b>4</b>	<b>Conclusion</b>	<b>8</b>

# 1 Introduction

Ce document a pour but de présenter notre projet pour la matière SEO.

Notre équipe est composée de 3 étudiants de la majeure SCIA promotion 2019 d'EPITA :

- Antoine DEBRENNE (debren\_a)
- Zinedine REBIAI (rebiai\_z)
- Carine ZHANG (zhang\_d)

Nous avons choisi de faire le projet numéro 2 : Aide à la rédaction.

## 2 Présentation générale du projet

### 2.1 Présentation

Le but de ce projet est d'implémenter un système d'aide à la rédaction. Le projet est divisé en 3 étapes :

- Extraction des n-grams
- Détection de la langue
- Aide à la rédaction

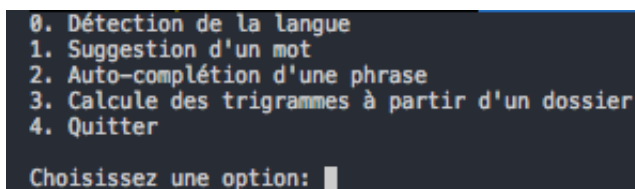
### 2.2 Exécution

Pour exécuter le code, à l'aide de Python 3, il suffit de saisir la commande suivante :  
`python menu.py`

Lorsque l'on lance le programme, on arrive sur un menu qui nous propose différentes options :

- Détection de la langue
- Suggestion d'un mot
- Auto-complétion d'une phrase
- Calcul des trigrammes à partir d'un dossier

FIGURE 1 – Menu



```
0. Détection de la langue
1. Suggestion d'un mot
2. Auto-complétion d'une phrase
3. Calcule des trigrammes à partir d'un dossier
4. Quitter

Choisissez une option: █
```

## 3 Le projet

### 3.1 Extraction des n-grammes

Un n-gramme est une séquence de  $n$  mots ou caractères. Dans ce projet, nous travaillons avec des n-grammes de mots pour proposer une auto-complétion de phrase, et avec des n-grammes de lettres pour proposer l'auto-complétion d'un mot. Dans l'idée, on prend chaque séquence de  $n$  mots de chaque document, que nous aurons au préalable normalisé, ainsi que chaque séquence de  $n$  lettres.

#### 3.1.1 N-grammes de caractères

Pour ce type de n-grammes, nous avons décidé de faire varier  $n$  de 2 à 4, car cela semblait adapté pour les applications que nous ferons de ces n-grammes.

Les n-grammes de caractères ont de nombreuses utilisations et il est important de les extraire tous des fichiers de texte. Nous avons donc décidé de créer une structure adaptée nous permettant de stocker les n-grammes extraits. Nous avons opté pour un dictionnaire qui va contenir, pour chaque n-gramme de lettre, le ou les mots d'où il a été extrait.

Cette structure nous servira pour l'auto-complétion d'un mot. Nous avons décidé d'utiliser la bibliothèque python *Pickle* qui permet de sérialiser ou désérialiser une structure de donnée. Ainsi, le traitement des n-grammes de lettres a été fait une seule fois en amont, et n'aura pas d'impact sur le temps d'exécution du projet.

#### 3.1.2 N-grammes de mots

Pour les mots, nous traiterons des trigrammes. Ces trigrammes vont nous permettre notamment à proposer de l'auto-complétion de phrases. De manière similaire au traitement des n-grammes de lettres, nous stockons nos trigrammes de mots dans une structure. C'est un dictionnaire qui contient la liste de tous les trigrammes, triés par langue. Nous stockons cette structure encore une fois à l'aide de pickle pour ne pas traiter trop souvent ces données.

### 3.2 Détection de la langue

La technique que nous avons utilisés afin de deviner le langage d'un document est de type probabiliste en utilisant les n-grammes du document en input.

Nous avons fait ce choix de technique puisque dans le projets nous devons déjà extraire les n-grammes d'un document, de plus la méthode est assez simple.

A l'aide de ressources sur internet nous avons pu trouver les Bi/Tri-grammes les plus fréquents selon les différentes langues (allemand, anglais, espagnol, français).

Ainsi, on va extraire les Bi/Tri-grammes de notre document afin de les comparer avec notre liste de Bi/Tri-grammes les plus fréquents.

Grâce à ce procédé assez simple on peut en déduire la langue de notre document.

FIGURE 2 – Bigrammes / Trigrammes les plus fréquents par langue

Français	Anglais	Allemand	Espagnol
on	th	en	de
es	on	er	en
de	an	ch	er
te	he	ei	on
nt	er	un	ci
re	nd	de	es
en	in	nd	re
le	ti	ge	os
it	al	re	io
er	re	in	la
et	io	ie	ra
ti	en	te	na
ou	ri	ng	ec
io	of	he	al
la	or	ne	ad
oi	at	ht	da
ne	it	ic	to
me	to	be	nt
ro	ed	it	ie
ns	nt	sc	ei

Français	Anglais	Allemand	Espagnol
ion	the	der	ion
tio	and	und	cio
ent	ion	ein	rec
oit	tio	ung	ere
ati	ati	cht	der
roi	igh	ich	ien
dro	ght	sch	cho
men	rig	che	ent
tou	ent	ech	ech
con	ver	die	aci
res	one	rec	ona
que	all	ine	nte
les	eve	eit	con
des	ery	gen	ene
eme	his	ver	tod

FIGURE 3 – Exemples de la détection de la langue

Choisissez une option: 0 Saisissez une phrase: Ceci est une phrase pour tester la détection de la langue de la phrase. Français
This is a sentence to test the language detection of the sentence. Anglais
Esta es una oración para probar la detección del idioma de la oración. Espagnol
Dies ist ein Satz, um die Erkennung der Sprache des Satzes zu testen. Allemand

### 3.3 Aide à la rédaction

Pour l'aide à la rédaction, nous avons implémenté 2 types d'aides :

- l'auto-complétion d'un mot
- l'auto-complétion d'une phrase, en donnant le prochain mot possible

#### 3.3.1 Point sur la TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) est une méthode qui permet, à partir d'une liste de documents, de déterminer l'importance d'un mot pour un document. Il est calculé avec les formules suivantes :

FIGURE 4 – Formules pour le calcul de TF-IDF

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

$$idf_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

$$tfidf_{i,j} = tf_{i,j} \cdot idf_i$$

- $tf$  étant la fréquence du  $i$ ème mot dans le document  $j$ , c'est-à-dire, le nombre de fois qu'apparaît le mot dans le document divisé par le nombre total de mot dans le document.
- $idf$  est la fréquence inverse de document, c'est-à-dire, l'importance du mot dans l'ensemble de tout les documents.
- $tfidf$  est la multiplication des deux, c'est donc l'importance d'un mot dans un des documents.

### 3.3.2 Calcul de la tf-idf sur des documents en entrée

Pour cette partie, il s'agit de parser des documents donnés par l'utilisateur, et de simplement en extraire les n-grammes, avec leur tf-idf pour avoir une idée des n-grammes significatifs. L'utilisateur donne donc le répertoire où sont les fichiers à parser, ainsi que le nom du fichier où seront stockés les résultats de la tf-idf.

### 3.3.3 Auto-complétion d'une phrase

Pour l'auto-complétion de phrase, la méthode est la suivante :

Après avoir extrait les trigrammes de mots de notre dataset, nous allons demander une phrase à l'utilisateur. On récupère ensuite le dernier mot de cette phrase pour trier nos trigrammes. En effet, nous n'allons garder que les trigrammes qui contiennent ce mot. Ensuite, nous appliquons une tf-idf sur les trigrammes restants (de manière très similaire à la façon de traiter de simples mots). Nous regardons la valeur maximum de la tf-idf pour chaque trigramme et nous nous servons de cette valeur pour les trier. Nous listons tous les trigrammes qui ont la meilleure tf-idf et nous affichons donc ce résultat.

Du point de vue de l'utilisateur, le programme se déroule ainsi :

- Choisir l'option de complétion d'une phrase
- Entrer une phrase dont la langue sera détectée automatiquement
- Le programme renvoie une liste de trigrammes servant à continuer la phrase

FIGURE 5 – Exemple de résultats de l'algorithme

```
Choisissez une option: 2
Saisissez le début d'une phrase:
Je ne me sens vraiment pas très bien, je suis un petit peu triste en ce moment en allant chez
['j'irai chez moi', 'chez la femme', 'vais chez vous', "j'étais chez moi", 'venez chez moi']
```

Ici, dans les phrases suggérées, le mot "moi" est le plus suggéré et on considère donc que c'est celui-ci qui complète la phrase.

### 3.3.4 Auto-complétion d'un mot

Tout d'abord pour l'aide à la rédaction, un dataset a été utilisé afin d'avoir des exemples de phrases de plusieurs langues et pouvoir deviner la suite d'un mot ou d'une phrase. Comme nous l'avons dit, nous avons utilisé pickle qui nous permet de sérialiser des objets python sous forme de fichier et pouvoir les utiliser directement.

L'auto-complétion d'un mot :

L'objectif de cette partie étant de "deviner" la suite d'un mot tout en étant pertinent. Un screenshot est en fin d'explication.

Afin de rendre l'explication la plus simple possible, nous allons expliquer étape par étape. La première étape étant de récupérer tous les (2,4)-grammes de mon mot. Ensuite, je fais matcher chacun de ces (2,4)-grammes avec notre dictionnaire n-grammes -> mots (mots qui proviennent du dataset).

A ce stade j'ai un tableau qui contient tous les mots possibles qui ont en commun un (2-4)-grammes avec mon mot. L'étape d'après est donc de faire un distance de Levenshtein-Damerau (nous prenons volontairement en compte les rotations) afin de connaître la distance entre mon mot et les mots que j'ai récupérés.

Si j'ai une distance de 0, je retourne le mot (cela veut dire que le mot existe bien dans notre dataset). Si ce n'est pas le cas, je récupère tous les mots qui ont une distance minimale avec le mot d'origine.

A ce stade, j'ai une liste de mots qui ont une distance proche du mot d'origine. Si nous n'avons qu'un mot dans la liste nous le retournons, sinon nous utilisons un dictionnaire sérialisé qui associe à chaque mot son nombre d'occurrences (mots qui proviennent du dataset).

Je vais donc comparer les occurrences dans mon dataset des mots de ma liste. Grâce à cela, j'obtiens le mots le plus fréquent de mon dataset et donc probablement le mot à donner. Bien entendu, les mots choisis sont en fonction de la langue qui est détectée avant la première étape.

```
Choisissez une option: 1
Saisissez le début d'une phrase et d'un mot:
Trouver l'itineraire de Paris a Pontoise en prenant la voiture electriq
électrique
```

Sur le screen, on rentre une phrase, on detecte la langue et le programme va deviner le dernier mot de la phrase. Si le mot existe dans notre dataset on l'affiche sinon on essaie de le compléter via l'explication donnée au dessus. Ici, le programme propose de remplacer "electriq" par "electrique"

## 4 Conclusion

Pour conclure, le projet était plutôt libre et donc très intéressant. Nous avons essayé d'être le plus ambitieux possible tout en respectant ce qui était demandé, c'est-à-dire un programme d'aide à la rédaction en utilisant les méthodes proposées.

De plus, il était intéressant de mettre en application certaines parties du cours et d'avoir des résultats concrets afin de mettre des intuitions sur des notions au départ théorique.

Ainsi, ce projet en plus d'être pédagogique peut servir dans le futur comme base de projet à plus grande échelle. En effet, on pourrait imaginer plusieurs améliorations comme un meilleur dataset, une interface graphique ou encore parallélisation pour aller encore plus vite.