Université Paris Nanterre

Analyse de données d'écriture en temps réel

Proposition d'un étiqueteur POS adapté aux données

Baptiste GILLET
Amandine JOUVENEL
Ioana-Madalina SILAI

1. Introduction

- ## Les objectifs du projet

L'objectif principal du projet consiste à étudier les schémas et les comportements des individus lorsqu'ils tapent du texte sur un clavier. Il vise à répondre à des interrogations telles que : qu'est-ce qui entraîne une pause chez une personne pendant qu'elle tape ? Est-ce une partie spécifique du discours (verbe, nom, pronom), une position particulière dans une phrase (début de syntagme, fin de syntagme), ou des mots particuliers qui posent constamment problème ?

Dans cette perspective, une expérience a été menée sur plusieurs participants, qu'ils soient experts ou non dans un domaine. On leur a demandé de saisir un extrait de texte, et leurs frappes de clavier ont été enregistrées à l'aide d'un logiciel spécialisé conservant des informations comme la position du curseur, les retours en arrière, les corrections, le temps de pause entre deux saisies, etc.

Le résultat obtenu était un document XML contenant ces informations pour chaque touche pressée, qu'il s'agisse d'une lettre, d'une suppression, d'un espace, etc. Ces données ont ensuite été prétraitées afin d’obtenir des fichiers csv contenant ces informations, où chaque ligne correspond à la production entre deux intervalles de pauses. Notre rôle dans ce projet est de proposer un étiqueteur en parties du discours adapté à ces données de nature particulière.

Nous commencerons dans un premier temps par présenter les données puisque ces dernières sont complexes et inhabituelles en TAL. Puis, nous expliquerons nos stratégies pour traiter ces données et la gestion des différents types d’erreurs. Enfin, nous donnerons les premiers résultats de notre étiquetage et nous conclurons sur les limites de notre approche et sur les améliorations possibles.

- **Les données**

Les fichiers dont nous disposons sont trois fichiers csv "formulation", "planification" et "révision". Chacun des fichiers csv présente les colonnes suivantes :

ID	charge	outil	n_burst	debut_burst	duree_burst	duree_pause	duree_cycle	pct_burst	pct_pause	longueur_burst	burst
P+S1	+	TW	1	22,397	10,464		10,464	1	0	45	Dans le cadre de la médecine traditionnelle,
P+S1	+	TW	2	38,557	7,296	5,696	12,992	0,56157635468	0,43842364532	31	nous voici face à un dillemme.
P+S1	+	TW	3	54,837	21,672	8,984	30,656	0,706941544885	0,293058455115	94	Beaucoup critiqué cette dernière s'avère parfois en effet fort efficace sur certaine
P+S1	+	TW	4	85,301	21,12	8,792	29,912	0,706071142017	0,293928857983	120	En effet si l'ont y réfléchie plus longuement beaucoup de nos méthodes actuelles
P+S1	+	TW	5	108,453	3,584	2,032	5,616	0,638176638177	0,361823361823	17	ecine alternative
P+S1	+	TW	6	116,061	10,767	4,024	14,791	0,727942667839	0,272057332161	39	, certains remède sont encore utilisés
P+S1	+	TW	7	129,972	2,512	3,144	5,656	0,444130127298	0,555869872702	11	à nos jours
P+S1	+	TW	8	134,38	3,464	1,896	5,36	0,646268656716	0,353731343284	2	.
P+S1	+	TW	9	145,38	0,096	7,536	7,632	0,0125786163522	0,987421383648	1	s
P+S1	+	TW	10	160,22	0,24	14,744	14,984	0,0160170848905	0,983982915109	1	
P+S1	+	TW	11	167,164	0,12	6,704	6,824	0,0175849941383	0,982415005862	1	s
P+S1	+	TW	12	184,604	18,136	17,32	35,456	0,511507220217	0,488492779783	67	De plus nous pouvons souligner le fait que la médecine alternative
P+S1	+	TW	13	204,732	7,736	1,992	9,728	0,795230263158	0,204769736842	31	n'utilise pas que les croyances
P+S1	+	TW	14	214,996	0,4	2,528	2,928	0,136612021858	0,863387978142	2	,

startPos	endPos	docLength	categ	charBurst
0	45	45	P	Dans␣le␣cadre␣de␣la␣médecine␣traditionnelle,␣
45	76	76	P	nous␣voici␣face␣à␣un␣dillemme.␣
76	170	170	P	Beaucoup␣critiqué␣cette␣dernière␣s'avère␣parfois␣ene⌫␣effet␣fort␣efficace␣sur␣certaines␣points.␣⌫⌫.
170	290	290	P	En␣effet␣si␣l'ont␣y␣réfléchie␣plus␣longuement␣beaucoup␣de␣nos␣méthodes␣actuelles␣se␣sont␣basé␣sur␣les␣fond␣de␣cette␣médi
290	306	306	RB	⌫lecine␣alternative
306	345	345	P	,␣certaine⌫s␣remède␣sont␣encà⌫lore␣utilisés␣
345	356	356	P	à␣nos␣jours
356	358	358	P	.␣C⌫
323	347	359	R	s
358	359	360	RB	␣
276	361	361	R	s
363	428	428	P	L'u⌫⌫⌫De␣plus␣nous␣pouvons␣com⌫⌫⌫souligner␣le␣fait␣que␣la␣médecine␣alternative␣
428	459	459	RB	n'utilisepas⌫⌫⌫␣pas␣que␣les␣croyances
459	461	461	P	.␣
461	483	497	P	les␣remèdes␣naturelles␣sont␣en␣effet
482	483	495	RB	⌫⌫
495	724	724	P	␣beaucoup⌫⌫⌫⌫⌫⌫⌫⌫⌫⌫⌫⌫t␣énom⌫rmément␣mis␣en␣avant␣contrairement␣à␣la␣médecine␣actuelle␣qui␣mêms⌫le␣si␣elle␣s'avère␣efficace␣ut
724	782	782	P	Cependant␣si␣nous␣devons␣faire␣une␣comparaison␣entre␣ces␣
782	882	882	P	deux␣méthodes␣médicales.␣il␣s'avère␣être␣vrai␣que␣la␣médecine␣alternative␣pose␣beaucoup␣de␣problème␣
855	864	887	RB	␣peut
865	887	887	RB	⌫é
886	888	889	P	s␣

Aperçu du fichier csv planification

Nous avons essentiellement travaillé sur les colonnes charburst, burst, et categ.

➢ La colonne charburst correspond au texte écrit entre deux intervalles de pause, avec les indications sur les modifications effectuées
(ex : de␣bien␣connaô⌫ître␣la␣maldie␣⌫⌫⌫⌫adie␣afin␣de␣la␣soignée)

➢ La colonne burst correspond au texte correspondant à charburst après modifications
(ex : de bien connaître la maladie afin de la soignée)

➢ La colonne categ correspond au type de production. Elle peut prendre trois valeurs :
- P (production) : production simple, avec ou sans correction(s) interne(s) à la production
- RB (révision de bord) : modification de la production précédente après une pause suffisamment longue
- R (révision) : modification d'une production plus haut dans le texte

burst	categ	charburst
Malheureusement en médecine,	P	Mla⌫⌫alheureusemet⌫nt␣en␣médecine,␣
les erreurs médicale de diago	P	les␣erreurs␣médicale␣de␣diago
nostic ce font beaucoup plus	RB	⌫nostic␣ce␣font␣é⌫beaucoup␣plus␣
s	R	s

Exemple de colonnes burst, categ, et charburst

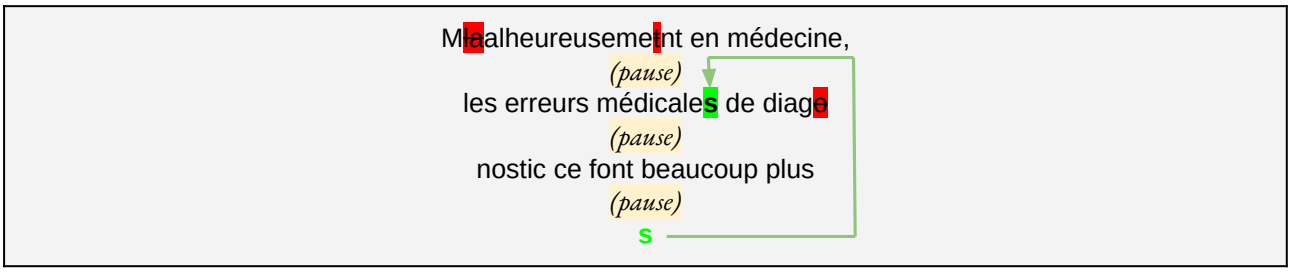
Dans cet exemple simplifié inspiré des données, la première et la deuxième lignes correspondent à des productions simples (avec ou sans correction interne). La troisième ligne correspond à une révision de bord puisque la personne effectue une correction de la ligne précédente avant de poursuivre (suppression du dernier "o" de la ligne 2). Enfin, la quatrième ligne correspond à une révision, puisque la personne effectue une correction sur la deuxième ligne (ajout d'un "s" après "médicale").

Le texte final à partir de ces données ressemblerait donc à ceci :

Malheureusement en médecine, les erreurs médicales de diagnostic ce font beaucoup plus

Résultat des productions

Avec le processus suivant :



Autre représentation du processus de production

On voit que les données représentent donc un véritable enjeu de traitement, notre but étant d'arriver à un résultat de ce type :

Token.text	Token.text	Token.pos	Erreur	Type_erreur	Correction	Correction_pos
1	Mla⌫⌫alheureusemet⌫nt	Unknown	TRUE	Lettres inversées	Malheureusemet	Adverbe
1	Mla⌫⌫alheureusemet⌫nt	Unknown	TRUE	Lettre manquante à l'intérieur du mot	Malheureusement	Adverbe
...	...	...	...	...	...	...
5	s	Unknown	TRUE	Lettre individuelle	Remèdes	Nom commun, pluriel
...	...	...	...	...	...	...
7	N'utilisepas	Unknown	TRUE	Mots collés	N'utilise pas	Verbe, present, troisième personne

2. Stratégies

• Stratégie 1

Étant donné la nature très particulière des données, le choix d'un étiqueteur est déterminant. En effet, puisqu'il s'agit de données d'écriture en temps réel, et non de textes finaux standards, certains mots sont incomplets ou incorrects. Notre première piste a donc été d'essayer de trouver un étiqueteur qui en tienne compte, et qui possèderait une étiquette "Unknown" pour ces mots incomplets/incorrects. Nous avons donc commencé par tester plusieurs étiqueteurs sur un mot incorrect isolé puis dans une phrase, afin d'évaluer leurs avantages et inconvénients.

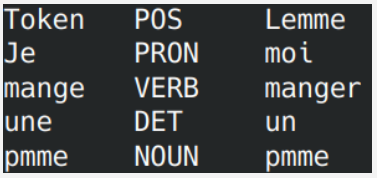
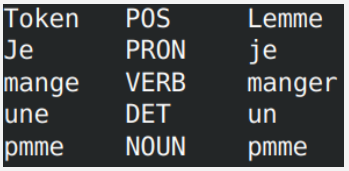
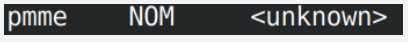
Etiqueteur	Mot incorrect isolé	Mot incorrect en contexte
Stanza		
Spacy		
Treetagger		

Tableau présentant les annotations du mot incorrect "pmme" isolé et en contexte

Étiqueteurs	Problèmes
Stanza	• Lorsque Stanza ne reconnaît pas un mot, il essaye de deviner sa POS, qu'il soit isolé ou en contexte. • Ne possède pas d'étiquette "UNKNOWN".
Spacy	• Lorsque Spacy ne reconnaît pas un mot, il essaye de deviner sa POS, qu'il soit isolé ou en contexte (comme Stanza). • Ne possède pas d'étiquette "UNKNOWN" (comme Stanza).
Treetagger	• Possède une étiquette "UNKNOWN" ! • Lorsque Treetagger ne reconnaît pas un mot isolé, il n'essaye pas de deviner sa POS • Mais lorsque ce mot est en contexte, il essaye de deviner sa POS

Tableau récapitulant les problèmes rencontrés avec chaque étiqueteur

Treetagger paraissait être l'outil le mieux adapté pour annoter des mots incorrects, puisqu'il possède une étiquette "UNKNOWN" et est capable de l'appliquer lorsqu'il ne reconnaît pas un mot. Néanmoins, dès lors que le mot est dans un contexte plus grand, Treetagger essaye de deviner sa POS. Aucun de ces étiqueteurs ne semblant adéquat pour étiqueter les mots incorrects, nous avons dû changer de stratégie.

• Stratégie 2

Nous avons alors créé nous-même un lexique comprenant tous les mots présents dans le texte final. Nous avons pour cela utilisé les fichiers docx résultant des productions écrites. Le raisonnement est le suivant : si un mot n'appartient pas au lexique, alors on l'annote avec l'étiquette "UNKNOWN", sinon, on l'annote avec Spacy. Cette méthode possède un double avantage. Premièrement, elle permet d'éliminer le problème des étiqueteurs qui devinent la POS. Deuxièmement, elle est généralisable à n'importe quelles autres données de ce type du moment que l'on possède le texte final produit (ce qui est normalement toujours le cas).

a) Constitution du lexique

Nous avons commencé par convertir les fichiers docx finaux en txt en utilisant la librairie pypandoc.
Puis, nous avons itéré sur chaque ligne de chaque fichier, et récupéré les mots dans une liste "lexique".

```python
import pypandoc
from pathlib import Path
import re

# Parcours arborescence :
def get_filenames(chemin_dossier: str) -> list[Path]:
    dossier = Path(chemin_dossier)
    return list(dossier.glob('*.docx'))

folder_list = sorted(get_filenames("TextesFinaux/
Planification"), key=lambda x: x.name)

liste = []
for file in folder_list:
    output = pypandoc.convert_file(file, 'plain',
    outputfile=f"{file.name}.txt")
    assert output == ""
    with open(file, 'r', encoding='UFT-8') as lecture_fichier:
        lecture_fichier.read()
        liste.append(lecture_fichier)
```

docx_to_txt.py

```python
from pathlib import Path
import spacy

nlp = spacy.load("fr_core_news_sm")

def get_filenames(chemin_dossier: str) -> list[Path]:
    dossier = Path(chemin_dossier)
    return list(dossier.glob('*.txt'))

file_liste = get_filenames("TextesFinaux_txt")
    lexique = []
    for file in file_liste:
        with open(file, 'r') as lecture_fichier:
            lecture = lecture_fichier.readlines()
            for line in lecture:
                line = line.strip()
                mots = nlp(line)
                for mot in mots:
                    mot = mot.text.lower()
                    if mot not in lexique :
                        lexique.append(mot)
                    else :
                        pass
    return lexique
```

creation_lexique.py

Dans *docx_to_txt.py*, on commence par récupérer la liste des chemins de chaque fichier docx. Puis, pour chaque fichier, on le convertit en txt en conservant son nom (mais en changeant l'extension). Enfin, on lit chaque fichier et on récupère son contenu dans une liste.

Dans *creation_lexique.py*, on commence par récupérer la liste des chemins de chaque fichier txt. Puis, pour chaque fichier, on tokenise chaque mot de chaque ligne (avec Spacy). Enfin, pour chaque mot (mis en minuscule), si le mot n'est pas déjà dans le lexique, alors on l'ajoute.

b) Détection des mots incorrects

Une fois notre lexique constitué, nous avons comparé les mots de la colonne charburst avec ce lexique afin de détecter les mots incorrects. Cette méthode s'appuie sur le présupposé que les textes finaux produits ne contiennent que des mots corrects (pas incomplets ou avec des erreurs de frappe), ce qui paraît être une hypothèse assez raisonnable étant donné qu'il s'agit des productions finales et corrigées par les participants.

```python
def clean_lines(liste_lignes: List[Ligne]) -> List[Ligne]:
    """Nettoie les lignes de la liste."""
    for ligne in liste_lignes:
        ligne.texte_complete = ligne.texte_complete.replace("␣"," ")
    return liste_lignes
```

Fonction de nettoyage des lignes de charburst

Afin d'éviter des erreurs d'encodage, nous avons commencé par nettoyer les lignes de charburst en remplaçant le caractère "␣" par un espace.

```python
def compare_data_lexique(liste_lignes: List[Ligne], lexique: List[str]):
    """Compare les données avec le lexique et retourne une liste d'erreurs."""
    n = 0
    liste_erreurs = []
    for ligne in liste_lignes:
        mots_originaux = nlp(ligne.texte_complete)
        for mot in mots_originaux:
            if mot.text!=" " and mot.text.lower() not in lexique:
                erreur = Erreur(mot, n, "Unknown", mot.pos_)
                liste_erreurs.append(erreur)
                # print(f"{n}. Erreur : {mot}")
                n += 1
    return liste_erreurs
```

Fonction d'identification des erreurs

Une fois les lignes nettoyées, nous avons itéré sur chaque ligne, et pour chaque mot (segmentation avec Spacy), si le mot n'appartenait pas au lexique, alors on l'ajoutait à la liste des erreurs.

Nous avons ainsi obtenu une liste des erreurs avec la partie du discours devinée par Spacy, et l'étiquette "Unknown" qui correspondrait à la POS réelle.

Ligne	Mot erroné	Pos réel	Pos supposé
0	ene	Unknown	PROPN
1	⌫	Unknown	PROPN
2	⌫	Unknown	PROPN
3	⌫	Unknown	PROPN
4	médi	Unknown	NOUN
5	⌫	Unknown	NOUN
6	ecine	Unknown	ADJ
7	⌫	Unknown	PROPN
8	s	Unknown	PRON
9	encà	Unknown	VERB
10	⌫	Unknown	NOUN
11	ore	Unknown	ADJ
12	⌫	Unknown	NOUN
13	s	Unknown	PRON
14	s	Unknown	PRON
15	u	Unknown	AUX
16	⌫	Unknown	PROPN
17	⌫	Unknown	PROPN
18	⌫	Unknown	PROPN
19	com	Unknown	ADJ
20	⌫	Unknown	PROPN
21	⌫	Unknown	PROPN
22	⌫	Unknown	PROPN

Extrait des erreurs obtenues

3. Gestion des erreurs

● Types d'erreurs identifiés

A partir de cette liste d'erreurs, nous avons essayé d'identifier les différents types que l'on observait, afin de les catégoriser. Nous avons ainsi identifié les types d'erreurs suivants :

- **Erreurs internes (production) :**
  - Suppression de caractères à l'intérieur d'un mot
    - ex : Mla⌫⌫alheureusemet
  - Mots collés
    - ex : utilisepas
  - Mots collés avec la première lettre du mot suivant
    - ex : ene
  - Mots qui semblent corrects mais qui n'existent pas dans le document final
    - ex : soulevaient

- **Erreurs de révision adjacente :**
  - Mot divisé sur plusieurs lignes
    - ex : ut */n* ilise
  - Mot corrigé sur plusieurs lignes (ligne 56-57)
    - ex : maldie */n* ⌫⌫⌫⌫adie

- **Erreurs de révision non adjacente :**
  - Lettre unique ajoutée à un mot précédent
    - ex : s ajouté à remède
  - Espace ajouté à un mot précédent
    - ex : ajouté à pomme
  - Un ou plusieurs backspaces (⌫) supprimant une chaîne
    - ex : ⌫⌫⌫⌫⌫⌫⌫⌫
  - Un ou plusieurs deletes (⌦) supprimant une chaîne
    - ex : ⌦⌦⌦⌦⌦
  - Mot inséré entre deux mots
    - ex : des inséré entre et et alentours
  - Insertion d'une chaîne entre deux mots
    - ex : au sein même de l'université inséré entre fumoirs et irait

- ## Types d'erreurs détectés automatiquement

Notre objectif suivant était d'essayer d'annoter le corpus en attribuant automatiquement le type d'erreur. Nous avons dû sélectionner certains types d'erreurs qui nous semblaient détectables automatiquement, que nous nous sommes répartis. Le but était d'écrire un script qui prend en entrée le fichier csv constituant le corpus (planification.csv), et renvoie un nouveau fichier csv contenant les erreurs détectées. Autrement dit, pour chaque token erroné détecté, on renvoie l'ID et le n_burst de la ligne correspondante, le token erroné, sa partie du discours prédite par Spacy, sa partie du discours réelle, son lemme, la valeur booléenne de l'erreur, le détail de l'erreur, et la correction.

Voici les erreurs que nous avons gérées pour le moment :

1. Suppression de caractères à l'intérieur d'un mot
2. Lettre unique ajoutée à un mot précédent
3. Espace ajouté à un mot précédent
4. Un ou plusieurs backspaces (⌫) supprimant une chaîne
5. Un ou plusieurs deletes (⌦) supprimant une chaîne
6. Mot inséré entre deux mots
7. Insertion d'une chaîne entre deux mots

Pour mieux comprendre la gestion de ces erreurs, voici la structure générale du script :

```
# Erreurs internes :
Si la ligne contient un backspace qui n'est pas le premier caractère et aussi au moins un autre caractère :
	Erreur : Suppression de caractères à l'intérieur d'un mot

# Erreurs de révision non adjacente :
Si la position de départ du curseur est différente de la position de fin du curseur de la ligne précédente et de la longueur du texte :
	Si la ligne est composée d'un unique caractère :
		Si le caractère est une lettre :
			Erreur : Lettre unique ajoutée à un mot précédent
		Sinon si le caractère est un espace :
			Erreur : Espace ajouté à un mot précédent
	Sinon (si la ligne est composée de plusieurs caractères) :
		Si la ligne est uniquement composée de backspaces :
			Erreur : Backspaces supprimant une chaîne
		Sinon si la ligne est uniquement composée de deletes :
			Erreur : Deletes supprimant une chaîne
		Sinon si la ligne est composée d'au moins un mot :
			Si la ligne est composée d'un seul mot :
				Erreur : Mot inséré entre deux mots
			Sinon (si la ligne est composée de plusieurs mots) :
				Erreur : Insertion d'une chaîne entre deux mots
```

1. Suppression de caractères à l'intérieur d'un mot

Définition : suppression d'un ou plusieurs caractères à l'intérieur d'un mot sans affecter les autres mots, renvoie en correction le mot corrigé

Exemple : encà⌫lore → *Suppression de caractères à l'intérieur d'un mot* → encore

Fonctionnement : Pour chaque ligne, si elle contient un backspace ("⌫") qui n'est pas le premier caractère et aussi au moins un autre caractère, on reconstitue le mot corrigé, puis on ajoute au token l'ID et le n_burst de la ligne correspondante, le token, sa partie du discours prédite par Spacy, sa partie du discours réelle ("Unknown"), son lemme prédit par Spacy, la valeur True à l'erreur, la description "Suppression de caractères à l'intérieur d'un mot", et le mot corrigé. On ajoute ensuite le token à la liste des tokens erronés.

Pour reconstituer le mot corrigé, on itère sur chaque caractère du mot, et s'il s'agit d'un backspace, alors on supprime le caractère précédent, sinon, on ajoute le caractère au mot corrigé.

2. Lettre unique ajoutée à un mot précédent

Définition : lettre isolée ajoutée à un mot pas forcément adjacent, renvoie en correction le mot avec l'ajout de la lettre

Exemple : s → *Lettre unique appartenant à "remède"* → remèdes

Fonctionnement : Si la position de départ du curseur est différente de la position de fin du curseur de la ligne précédente et de la longueur du texte, si la ligne est composée d'un unique caractère, si le caractère est une lettre, on ajoute au token l'ID et le n_burst de la ligne correspondante, le token, sa partie du discours prédite par Spacy, sa partie du discours réelle ("Unknown"), son lemme prédit par Spacy, la valeur True à l'erreur, la description "Lettre unique appartenant à {word_before}", et le mot corrigé. On ajoute ensuite le token à la liste des tokens erronés.

Pour trouver le mot auquel la lettre est ajoutée, on utilise la position de départ du curseur et on récupère le mot contenant le caractère correspondant à cette position. Pour obtenir le mot corrigé, on ajoute simplement la lettre à cette même position.

3. Espace ajouté à un mot précédent

Définition : espace ajouté à un mot pas forcément adjacent, renvoie en correction le mot avec l'ajout de l'espace

Exemple : pas d'exemple dans les données

Fonctionnement : Si la position de départ du curseur est différente de la position de fin du curseur de la ligne précédente et de la longueur du texte, si la ligne est composée d'un unique caractère, si le caractère est un espace, on ajoute au token l'ID et le n_burst de la ligne correspondante, le token, sa partie du discours prédite par Spacy, sa partie du discours réelle ("Unknown"), son lemme prédit par Spacy, la valeur True à l'erreur, la description "Espace appartenant à {word_before}", et le mot corrigé. On ajoute ensuite le token à la liste des tokens erronés.

Pour trouver le mot auquel l'espace est ajouté, on utilise la position de départ du curseur et on récupère le mot contenant le caractère correspondant à cette position. Pour obtenir le mot corrigé, on ajoute simplement l'espace à cette même position.

4. Un ou plusieurs backspaces (⌫) supprimant une chaîne

Définition : suppression avec un backspace d'un ou plusieurs mots pas forcément adjacents, renvoie en correction le mot obtenu après suppression

Exemple : s et autres → *11 backspaces supprimant 's et autres'* → plante

Fonctionnement : Si la position de départ du curseur est différente de la position de fin du curseur de la ligne précédente et de la longueur du texte, si la ligne est composée de plusieurs caractères, si la ligne est uniquement composée de backspaces ("⌫"), on ajoute au token l'ID et le n_burst de la ligne correspondante, le token, sa partie du discours prédite par Spacy, sa partie du discours réelle ("BACKSPACE"), son lemme prédit par Spacy, la valeur True à l'erreur, la description "{string_length} backspaces supprimant '{deleted_string}'", et le mot corrigé. On ajoute ensuite le token à la liste des tokens erronés.

Pour récupérer la chaîne supprimée, on utilise la position du curseur et on récupère la ligne correspondant à cette position. Pour obtenir le mot corrigé, on se place à cette même position et on supprime autant de caractères précédant la position du curseur qu'il y a de backspaces.

5. Un ou plusieurs delete (⌦) supprimant une chaîne

Définition : suppression avec un delete d'un ou plusieurs mots pas forcément adjacents, renvoie en correction le mot après suppression

Exemple : ⌦ → *1 delete (⌦) supprimant ' '* → non

Fonctionnement : Si la position de départ du curseur est différente de la position de fin du curseur de la ligne précédente et de la longueur du texte, si la ligne est composée de plusieurs caractères, si la ligne est uniquement composée de deletes ("⌦"), on ajoute au token l'ID et le n_burst de la ligne correspondante, le token, sa partie du discours prédite par Spacy, sa partie du discours réelle ("DELETE"), son lemme prédit par Spacy, la valeur True à l'erreur, la description "{string_length} delete (⌦) supprimant '{deleted_string}'", et le mot corrigé. On ajoute ensuite le token à la liste des tokens erronés.

Pour récupérer la chaîne supprimée, on utilise la position du curseur et on récupère la ligne correspondant à cette position. Pour obtenir le mot corrigé, on se place à cette même position et on supprime autant de caractères suivant la position du curseur qu'il y a de deletes.

6. Mot inséré entre deux mots

Définition : insertion d'un mot entre deux mots, renvoie en correction la chaîne correspondant au triplet (mot d'avant, mot ajouté, mot d'après)

Exemple : qui → *Mot inseré entre 'l'université' et 'comportera'* → l'universitéqui comportera

Fonctionnement : Si la position de départ du curseur est différente de la position de fin du curseur de la ligne précédente et de la longueur du texte, si la ligne est composée de plusieurs caractères, si la ligne est composée d'au moins un mot, si la ligne est composée d'un seul mot, alors on ajoute au token l'ID et le n_burst de la ligne correspondante, le token, sa partie du discours prédite par Spacy, sa partie du discours réelle (qui est la même que celle prédite par Spacy), son lemme prédit par Spacy, la valeur True à l'erreur, la description "Mot inseré entre '{previous_word}' et '{next_word}'", et la chaîne obtenue après insertion. On ajoute ensuite le token à la liste des tokens erronés.

Pour obtenir le mot d'avant, on récupère le mot correspondant à la position précédant celle du mot ajouté. Pour obtenir le mot d'après, on récupère le mot correspondant à la position suivant celle du mot ajouté. Pour obtenir la chaîne obtenue après insertion, on concatène simplement le mot d'avant, le mot ajouté, et le mot d'après.

7. Insertion d'une chaîne entre deux mots

Définition : insertion d'un ou plusieurs mots entre deux mots, renvoie en correction la chaîne correspondant au triplet (mot d'avant, chaîne ajoutée, mot d'après)

Exemple : dans → *Partie de la chaine ' dans la faculté' inserée entre 'fumoirs' et ''* → fumoirs dans la faculté

Fonctionnement : Si la position de départ du curseur est différente de la position de fin du curseur de la ligne précédente et de la longueur du texte, si la ligne est composée de plusieurs caractères, si la ligne est composée d'au moins un mot, si la ligne est composée de plusieurs mots, alors on ajoute à chaque token l'ID et le n_burst de la ligne correspondante, le token, sa partie du discours prédite par Spacy, la partie du discours réelle (qui est la même que celle prédite par Spacy), son lemme prédit par Spacy, la valeur True à l'erreur, la description "Partie de la chaine '{list_lines[i].texte_simple}' inserée entre '{previous_word}' et '{next_word}'", et la chaîne obtenue après insertion. On ajoute ensuite chaque token à la liste des tokens erronés.

Pour obtenir le mot d'avant, on récupère le mot correspondant à la position précédant celle de la chaîne ajoutée. Pour obtenir le mot d'après, on récupère le mot correspondant à la position suivant celle de la chaîne ajoutée. Pour obtenir la chaîne obtenue après insertion, on concatène simplement le mot d'avant, la chaîne ajoutée, et le mot d'après.

- **Ecriture du résultat dans un fichier csv**

On obtient donc une liste de tokens erronés (avec leurs informations récupérées). Pour sauvegarder le résultat, on crée un fichier csv dans lequel on écrit les informations de chaque token erroné.

4. Résultats de l'étiquetage

• Résultat final

Voici à quoi ressemble le fichier csv résultant :

Id	n_burst	Token_texte	POS Supposé	Pos réel	Lemme	Erreur	Details	Correction
P+S1	6	encà⌫ore	ADJ	Unknown	or	TRUE	Suppression de caractères à l'intérieur d'un mot	encore
P+S1	9	s	PRON	Unknown	s	TRUE	Lettre unique appartenant à "remède"	remèdes
P+S13	29	⌫⌫⌫⌫⌫ ⌫⌫⌫⌫⌫ ⌫	PROPN	BACKSPACE	⌫	TRUE	11 backspaces supprimant 's et autres'	plante
P+S19	81	⌦	AUX	DELETE	⌦	TRUE	1 delete (⌦) supprimant ' '	non
P-S10	26	qui	PRON	PRON	qui	TRUE	Mot inseré entre 'l'université' et 'comportera'	l'université qui comportera
P-S10	32	dans	ADP	ADP	dans	TRUE	Partie de la chaine ' dans la faculté' inserée entre 'fumoirs' et ''	fumoirs dans la faculté

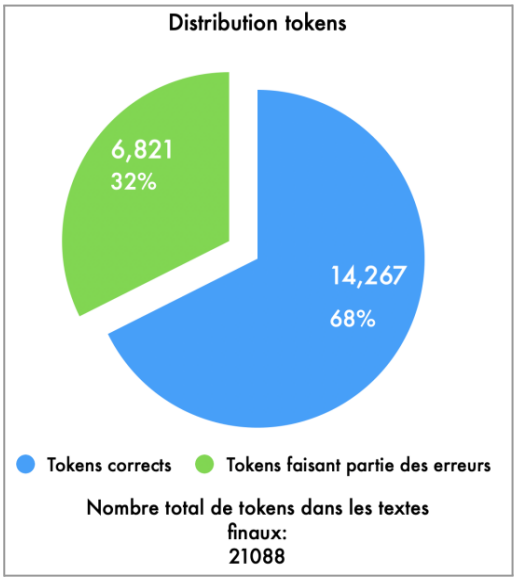
Extrait du fichier csv contenant les erreurs détectées (en ne gardant qu'un seul exemple pour chaque erreur)

Pour chaque erreur détectée, on a donc l'identifiant de la personne qui a tapé la ligne contenant l'erreur avec le numéro du burst, le token erroné, ses parties du discours prédite par spacy et réelle, son lemme, la valeur (TRUE) ainsi que le détail de l'erreur, et la correction.
Les valeurs des colonnes Id et n_burst permettent d'établir une connexion directe avec le fichier d'origine pour savoir quelles productions contiennent des erreurs et de quel type.

• Analyse des résultats obtenus pour le moment

Du point de vue qualitatif, nous obtenons plus d'informations sur les données représentant le processus d'écriture d'un texte en temps réel. Du point de vue quantitatif pourtant, nous ne pouvons pas utiliser les mesures classiques (rappel, précision, f-mesure) faute de disposer d'une version annotée manuellement. Cela n'était pas une option viable dans le cadre de ce projet, car même sans prendre en considération le temps de l'annotation effective, nous manquons de connaissances suffisantes dans ce domaine de la recherche linguistique. Il conviendrait donc d'entreprendre des recherches approfondies afin de déterminer comment ce type de données est généralement annoté, comment les regrouper, et s'il existe des logiciels mieux adaptés, entre autres aspects.
De plus, notre solution ne traite pas toutes les erreurs, donc l'évaluer avec une annotation complète serait inutile. Toutefois, nous avons réalisé une première analyse basée sur les résultats des erreurs que nous avons jusqu'à présent traitées.

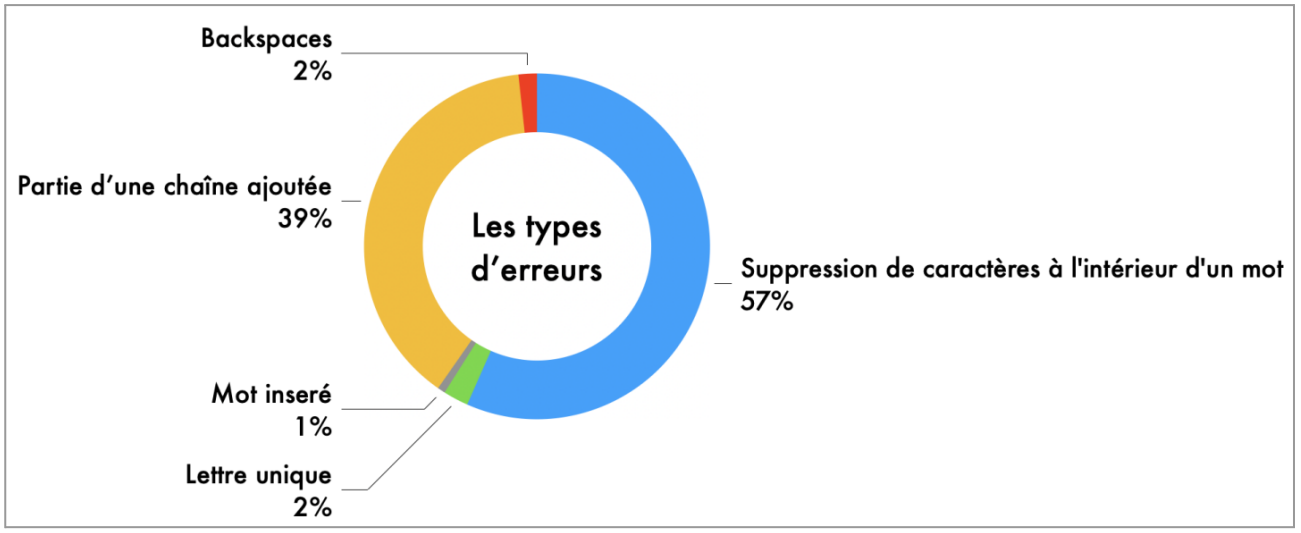
Nous pouvons tout d'abord comparer le nombre de tokens détectés comme étant "incorrects" avec le nombre de tokens "corrects".



On voit que sur les 21088 mots produits, notre script permet de détecter 6821 mots incorrects, contre 14267 mots qui seraient a priori corrects. Cela montre que les erreurs constituent une partie importante des données, puisqu'avec seulement les erreurs que nous avons réussi à détecter, on arrive à un presque un tiers des données totales.

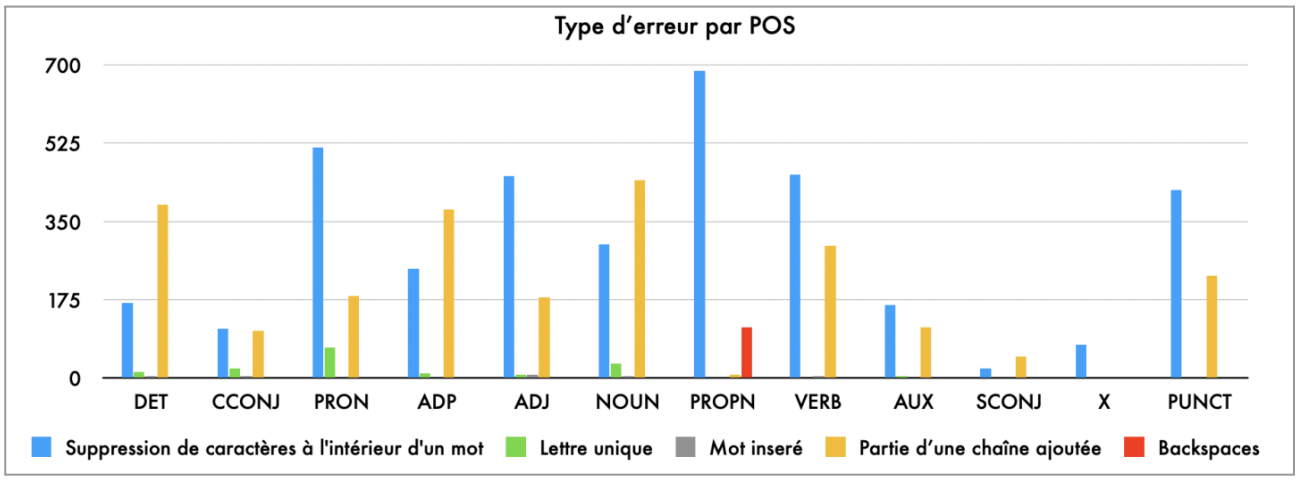
Cependant, on ne peut pas tirer de conclusion sur le pourcentage d'erreurs détectées parmi toutes les erreurs puisqu'il faudrait pour cela avoir identifié toutes les erreurs possibles.

Nous avons également pu obtenir une distribution des différentes erreurs détectées :



Ces résultats nous montrent que les erreurs de suppression de caractères à l'intérieur d'un mot et d'insertion d'une chaîne de plusieurs mots sont les plus fréquentes. Cela nous laisse donc penser qu'il serait pertinent d'approfondir la gestion de ces erreurs pour les diviser en sous-catégories et ainsi obtenir plus d'informations dessus.

Enfin, nous avons regardé s'il y avait une corrélation entre le type d'erreur et la partie du discours du token.



Par exemple, pour la suppression de caractères à l'intérieur d'un mot (en bleu), on remarque que les parties du discours les plus affectées sont de loin les noms propres, puis les pronoms, adjectifs, verbes et ponctuations, et enfin plus minoritairement les autres parties du discours. Concernant l'ajout d'une chaîne de caractères (en jaune), on remarque que les parties du discours les plus affectées sont les noms, les déterminants et les adpositions, puis les verbes et la ponctuation, et plus minoritairement les autres parties du discours. Ce n'est pas surprenant, étant donné le fait que lorsqu'on ajoute plusieurs mots dans un texte déjà écrit, c'est plus probable que ce soit une phrase complète ou une subordonnée.

Il serait donc intéressant, pour les parties du discours les plus touchées, d'approfondir leur analyse afin d'obtenir plus d'informations morphologiques comme le genre, le nombre, le type, le temps, la personne, etc, afin de pouvoir identifier des patterns plus précis.

5. Conclusion

• Limites de notre approche

Pour conclure, nous sommes conscients que notre approche de détection des erreurs possède des limites.

En effet, le fait de créer un lexique à partir des textes finaux pour comparer les mots des productions avec ceux présents dans le lexique pose le problème des mots qui sont détectés comme étant erronés alors qu'ils n'appartiennent juste pas aux documents finaux mais qu'ils sont corrects.
Nous avons longuement réfléchi pour trouver un moyen pour contourner ce problème mais nous n'avons pas réussi à trouver une solution satisfaisante car savoir si une chaîne de caractères est un mot est une tâche simple pour un locuteur d'une langue, mais pas pour une machine. Il faudrait donc utiliser des modèles de langues conséquents avec des connaissances linguistiques plus approfondies.

D'autre part, l'annotation automatique avec Spacy comme annotation de référence pose également deux problèmes. Premièrement, il s'agit d'un parser automatique, donc même si ses performances sont élevées, il n'est pas infaillible. Enfin, Spacy ne fournit pas d'informations sur le temps des verbes, la personne, le genre des noms, etc. Nous pourrions essayer de tester à nouveau notre code mais en utilisant cette fois-ci Treetagger.

• Ce qu'il reste à faire

Il faut tout d'abord poursuivre le travail en essayant de gérer plus de types d'erreurs. On pourrait par exemple, pour les erreurs internes à une production, gérer les mots collés entre eux, les mots collés avec la première lettre du mot suivant, ou encore les mots qui semblent corrects mais qui n'existent pas dans le document final. Pour les erreurs de révision adjacente, on pourrait essayer de gérer les mots divisés sur plusieurs lignes, ou bien les mots corrigés sur plusieurs lignes.

Pour les erreurs que nous avons géré pour le moment, il faudrait également poursuivre cette gestion des erreurs en approfondissant leur analyse pour tenter d'identifier des patterns plus précis. Il serait également intéressant d'ajouter des informations à la sortie obtenue par notre script, comme le temps de pause pour pouvoir peut-être identifier s'il y a ou non un lien avec le type d'erreur produite.

On pourrait également essayer de collecter plus de données pour refaire ce travail avec du machine learning et comparer les résultats avec ceux que nous avons obtenus.

Une fois ce travail terminé, on pourrait alors commencer une tâche de chunking et d'analyse en dépendances pour essayer d'identifier les patterns dans lesquels les fautes apparaissent.

Il ne faut néanmoins pas oublier que notre rôle est simplement de développer un outil qui répond aux besoins d'un domaine particulier de la linguistique, et que les résultats obtenus par cet outil pourront par la suite être étudiés par des spécialistes du domaine.