

# Analyse de données d'écriture en temps réel

---

Baptiste GILLET  
Amandine JOUVENEL  
Ioana-Madalina SILAI

# Contenu de la présentation

<b>Introduction</b>	Les objectifs de notre projet et les données traitées
<b>Strategies</b>	Ce que nous avons essayé et les choix que nous avons faits
<b>Gestion des erreurs</b>	La gestion des erreurs de frappe et types d'erreurs identifiés
<b>Résultats de l'étiquetage</b>	Les résultats de notre étiquetage
<b>Conclusion</b>	Les limites de notre approche et travail futur

# 01

# Introduction

---

Les objectifs de notre projet et les données traitées

# Contexte du projet

Le but plus large du projet est **d'analyser les patterns** et les comportements des personnes quand elles tapent du texte.

Il cherche à répondre à des questions telles que : qu'est-ce qui provoque une pause chez quelqu'un pendant qu'il tape ? Est-ce une partie spécifique du discours, une position particulière dans une phrase, des mots particulières qui posent toujours des problèmes?

Dans cette optique, plusieurs participants, à la fois experts et non-experts, ont été invités à taper un morceau de texte et leurs frappes de clavier ont été enregistrées à l'aide d'un logiciel spécialisé.

Le résultat est un document XML qui contient ces informations pour chaque touche pressée – que ce soit une lettre, un retour en arrière, espace, etc.

```
<event type="keyboard" id="277">
  <part type="wordlog">
    <position>119</position>
    <documentLength>120</documentLength>
    <replay>True</replay>
  </part>
  <part type="winlog">
    <startTime>14903463</startTime>
    <endTime>14903528</endTime>
    <key>VK_T</key>
    <value>t</value>
    <keyboardstate />
  </part>
</event>
```

# Les données

A partir de ces fichiers XML, on a pu obtenir un csv qui colle toutes ces informations dans un format plus lisible:

startPos	endPos	docLength	categ	charBurst
1453	1499	1500	P	Mla⊗⊗alheureusemet⊗nt,en,médecine,il est important,
1499	1546	1547	P	de,bien,connaô⊗tre,Ja,maldie,⊗⊗⊗⊗adie,afin,de,la,soignée
1546	1637	1638	P	..et,ce,n'est,pas,toujours,le,cas,de,la,médecine,alternative,même,si,cette,⊗d,⊗,dernière,dispose,
1637	1703	1704	P	aussi,de,remède,pouvant,être,plus,afi⊗f⊗⊗efficace,qu'en,médecine,moderne.

# Exemples

P	Mla ↗ alheureusement ↗ int_en _médecine, _il_est_important_
P	de_bien_connâo ↗ ître_la_maldi e_ ↗ ↗ ↗ adie_afin_de_la_s oignée
P	_et_ce_n'est_pas_toujours_le_ cas_de_la_médecine_alternativ e_même_si_cette_d ↗ d ↗ derni ère_dispose_
P	aussi_de_remède_pouvant_être _plus_afi ↗ f ↗ ↗ efficace_qu 'en_médecine_moderne.

**P**

Production

start Pos	end Pos	docLength	categ	charBurst
873	919	1903	R	d'autre_problème_ ↗ s_m éi ↗ dixau ↗ ↗ ↗ caux_ch ez_le_plus_patient
1078	1090	1935	R	_malgré_tout
1215	1221	1960	R	s
1264	1355	1960	R	↗ s

**R**

Révision  
(on remonte dans le texte)

start Pos	end Pos	docLength	categ	charBurst
782	882	882	P	deux_méthodes_médicale s,_il_s'avère_être_vrai_qu e_la_médecine_alternative _pose_beaucoup_de_pro blème_
855	864	887	RB	_peut
865	887	887	RB	↗ é

**RB**

Révision de bord  
(on modifie ce qui a  
été produit juste  
avant une pause)

# Objectif

L'objectif final de notre projet c'est de proposer un étiqueteur POS adapté aux données, qui donc tient compte des problèmes liés aux erreurs de frappe.

Notre but c'était d'arriver à un résultat de ce type:

Token.text	Token.text	Token.pos	Erreur	Type_erreur	Correction	Correction_pos
1	Mla ↵ ↵ alheureusement ↵ nt	Unknown	TRUE	Lettres inversées	Malheureusement	Adverbe
1	Mla ↵ ↵ alheureusement ↵ 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, présent, troisième personne

# 02

# Strategies

---

Ce que nous avons essayé et les choix que nous avons faits



# Notre plan initial

## Analyse les données

Un étiqueteur comme Spacy va mettre "Unknown" si le mot est incomplet

## Analyse des erreurs

On fait une liste des "unknown" et on pourra en tirer des conclusions sur le contexte

## Entrainement d'un modèle pour l'étiquetage

En utilisant une partie de données qu'on annote manuellement

# Il nous fallait d'abord un étiqueteur qui ait l'étiquette "unknown" pour les mots incomplets.

Spacy	Stanza	TreeTagger
<ul style="list-style-type: none"><li>• Devine le POS</li><li>• Garde le lemme</li></ul>	<ul style="list-style-type: none"><li>• Devine le POS</li><li>• Devine le lemme</li><li>• Donne "noun" si impossible à deviner</li></ul>	<ul style="list-style-type: none"><li>• Donne "UNK" mais pas pour chaque mot</li><li>• Devine la plupart des cas</li></ul>
<i>Resultats pour "ch" de la phrase "L'oiseau ch" provenant de "L'oiseau chante"</i>		
POS: NOUN Lemme: ch	POS: NOUN Lemme: cheval	POS: UNK Lemme: ch



## C'était donc impossible d'étiqueter les mots erronés

---

Par conséquent, nous avons changé de stratégie

# Notre *nouveau* plan ~~initial~~

## Analyse les données

**Nous allons créer un lexique des mots en utilisant les textes finaux. Si un mot n'existe pas dans le lexique, on mettra l'étiquette "Unknown"**

## Analyse des erreurs

On fait une liste des "unknown" et on pourra en tirer des conclusions sur le contexte

## Entrainement d'un modèle pour l'étiquetage

**Comme la quantité des données est insuffisante nous ne pouvons pas utiliser des outils entraînés. Il faudra donc utiliser une approche plus "rigide" pour l'étiquetage.**

```
@dataclass
class Difference:
    """Classe représentant une erreur."""
    mot_errone: str
    ligne: int
    pos_reel: str
    pos_suppose: str
```

```
@dataclass
class Ligne:
    """Classe représentant une ligne du fichier de données."""
    texte_complete: str
    texte_simple: str # texte sans les caractères de suppression, et qui a les caractères supprimés
    catégorie: str
    start_position: int
    end_position: int
    doc_length: int
    id: str
    n_burst: int

@dataclass
class Token:
    """Classe représentant un token du fichier de données."""
    texte: str
    pos_suppose: str
    lemme: str
    erreur: bool
    details: str
    pos_reel: str
    correction: str
    ligne: Ligne
```

## D'abord on s'est mis d'accord sur une structure des classes

Avec ces méthodes dans la classe Token

```
def get_pos_suppose(self):
    """Retourne la pos supposée."""
    doc = nlp(self.texte)
    for token in doc:
        self.pos_suppose = token.pos_
    return self.pos_suppose

def get_lemme(self):
    """Retourne le lemme."""
    doc = nlp(self.texte)
    for token in doc:
        self.lemme = token.lemma_
    return self.lemme
```

# Ensuite nous avons créé notre lexique

```
def obtener_lexique(file_list):
    lexique = []
    for file in file_list:
        with open(file, 'r') as lecture_fichier:
            # on lit le fichier et on le tokenise
            lecture = lecture_fichier.readlines()
        for line in lecture:
            line = line.strip()
            mots = nlp(line)
            # on ajoute les mots du fichier dans le lexique
            for mot in mots:
                mot = mot.text.lower()
                if mot not in lexique :
                    lexique.append(mot)
                else :
                    pass
    return lexique
```

Et nous avons comparé les données dans le format vu avant, qui maintenant étaient nettoyées et tokenisées, avec les tokens du lexique

```
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_complet)  
        for mot in mots_originaux:  
            if mot.text!=" " and mot.text.lower() not in lexique:  
                erreur = Difference(str(mot), n, "Unknown", mot.pos_)  
                liste_erreurs.append(erreur)  
            n += 1  
    return liste_erreurs
```

**Le résultat démontre parfaitement pourquoi on ne peut pas juste utiliser un pos tagger "classique".**

**Il nous permet aussi d'identifier les types d'erreurs présents.**

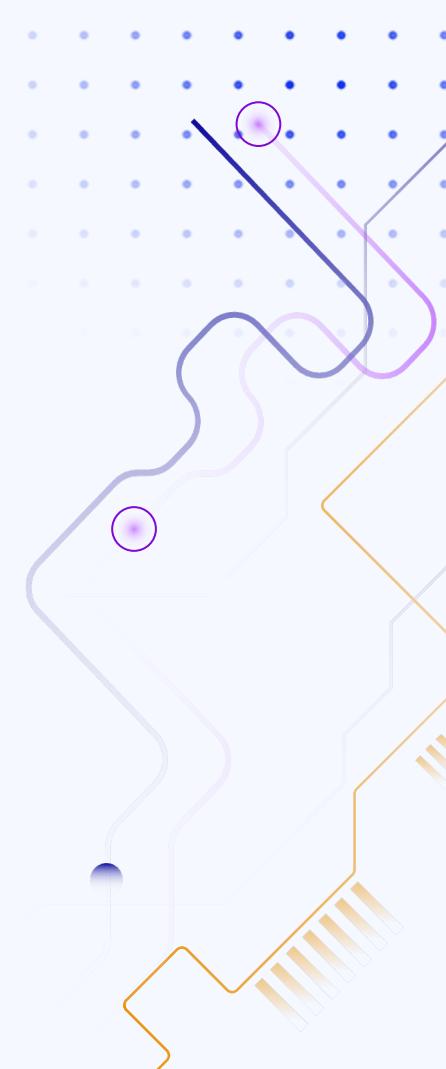
Ligne	Mot erroné	Pos réel	Pos supposé
0	ene	Unknown	PROPN
1	✉	Unknown	PROPN
2	✉	Unknown	PROPN
3	✉	Unknown	PROPN
4	médi	Unknown	NOUN
5	✉	Unknown	NOUN
6	ecine	Unknown	ADJ
7	✉	Unknown	PROPN
8	s	Unknown	PRON
9	encà	Unknown	VERB
10	✉	Unknown	NOUN
11	ore	Unknown	ADJ
12	✉	Unknown	NOUN
13	s	Unknown	PRON
14	s	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

# 03

# Gestion des erreurs

---

Types d'erreurs et comment les identifier automatiquement



# Exemples des erreurs identifiés

**Suppression des lettres à l'intérieur d'un mot**

*monrra ☒☒erait*

**Mots collés ou avec espaces erronés**

*ene ☒\_effet  
n'utilisepas ☒☒☒\_pas*

**Lettres ou mots individuels ajoutés sur une autre ligne que l'actuelle**

*apportera ☒☒☒☒☒☒☒☒☒☒☒☒*

**Mots entiers supprimés avec une seule lettre erronnée**

**Bursts qui ne sont que des caractères de supprime**

*☒☒☒☒☒☒☒☒☒☒☒*

# Les erreurs identifiées

**Erreurs dans la production**

**Erreurs dans la révision**

## Erreurs internes

Des modifications dans la même ligne (burst)

## Erreurs non adjacentes

Des modifications sur une ligne antérieure

## Le defi principal

Savoir à quel point une chaîne est un mot "réel"

## Le defi principal

Trouver une manière d'identifier exactement où la révision a été faite

Comment aborder une combinaison entre les deux types d'erreurs?

Dans le cadre de la médecine traditionnelle,

nous voici face à un dilemme.

Beaucoup critiqué cette dernière s'avère parfois en effet fort efficace sur certaines.

En effet si l'ont y réfléchie plus longuement beaucoup de nos méthodes actuelles se

meilleure alternative

, certains remèdes sont encore utilisés

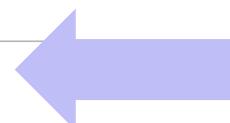
à nos jours

Ces

s

l'

s



**Les erreurs dans  
la révision**

Au début on se posait le problème de l'efficacité d'un script, où chaque fois qu'il y a une révision il faut vérifier 4000 bursts pour voir où la révision a été faite.

En fait on n'a pas besoin de chercher la correcte ligne dans l'ensemble de données, mais seulement dans l'ensemble de bursts d'une seule personne.

J'ai donc écrit une fonction qui réorganise les bursts dans un dictionnaire où la clé est l'id d'une personne et la valeur une liste des bursts de cette personne.

Cela permet d'optimiser et faciliter la recherche d'un burst lors d'une révision.

```
def get_persons(liste_lignes: List[Ligne]) -> Dict[str, List[Ligne]]:  
    """Retourne une liste de dictionnaires, dont la clé est l'id et la valeur une liste de burts."""  
    personnes = {}  
    for ligne in liste_lignes:  
        if ligne.id not in personnes.keys():  
            personnes[ligne.id] = []  
            personnes[ligne.id].append(ligne)  
    return personnes
```

Ensuite, pour pouvoir trouver la position d'une lettre ajoutée par exemple, on a besoin de chercher dans version "réelle" du texte.

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_dilemme..
76	170	170	P	Beaucoup_critiqué_cette_dernière_s'avère_ parfois_une_à_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	éicine_alternative
306	345	345	P	,_certaines_à_is_remède_sont_ençà_ore_utilisés..
345	356	356	P	à_nos_jours
356	358	358	P	..Cà
323				Si on comptait de 306 jusqu'à 323
358				on introduirait le s entre le d et le e
276				voit qu'il a été ajouté à la position 323, donc on sait

Si on comptait de 306 jusqu'à 323  
on introduirait le s entre le d et le e  
voit qu'il a été ajouté à la position 323, donc on sait

Donc on a besoin de chercher dans une version qui prend en considération le fait que le "e" en "certaines" a été supprimé - la version qui est équivalente à ce que la personne avait devant ses yeux quand elle a fait la modification

Donc pour “recréer” les actions de la personne qui tape le texte, nous avons écrit cette fonction.

Elle nous permet de réécrire le texte, en supprimant et ajoutant des caractères dans chaque burst.

Le résultat c'est la version réelle du texte après chaque burst.

Comme ça, je peux introduire le texte de la révision au bon endroit.

```
def add_burst_to_text(existing_text: str, burst: str, position: int) -> str:  
    """Adds a burst (including backspaces) to existing text at position"""\n    cursor = position\n    result = existing_text\n    for char in burst:\n        if char == "\u0008":\n            if cursor > 0:\n                result = result[:cursor-1] + result[cursor:]\n                cursor -= 1\n        elif char == "\u001f":\n            if cursor < len(result) - 1:\n                result = result[:cursor] + result[cursor+1:]  
        elif char == "\u0009":\n            result = result[:cursor] + " " + result[cursor:]\n            cursor += 1\n        else:\n            result = result[:cursor] + char + result[cursor:]  
            cursor += 1\n    return result
```

# Les différents types de révision



Si le burst  
est une  
revision

le burst est  
un caractère

une lettre

un espace

un caractère de supprime

le burst est  
plusieurs  
caractères

plusieurs caractères de supprime

un mot

plusieurs mots

D'abord on construit le texte

```
# Add this line to the document text so far
running_text_after = add_burst_to_text(running_text, list_lines[i].texte_complete, list_lines[i].start_position)

if list_lines[i].start_position != list_lines[i-1].end_position and list_lines[i].start_position != len(running_text):
    correction_position = list_lines[i].start_position
    # if it's one character
    if len(list_lines[i].texte_simple.strip()) == 1:
        if list_lines[i].texte_simple.isalpha():
            # if it's just one letter
            try:
                word_before, start_of_word, _ = get_word(correction_position, running_text)
                word_after, _, _ = get_word(start_of_word, running_text_after)
                token = Token(
                    texte=list_lines[i].texte_simple,
                    pos_suppose="",
                    lemme=" ",
                    erreur=True,
                    details=f'Lettre unique appartenant à "{word_before}"',
                    pos_reel="Unknown",
                    correction=word_after,
                    ligne=list_lines[i]
                )
                token.pos_suppose = token.get_pos_suppose()
                token.lemme = token.get_lemme()
                tokens.append(token)
            except IndexError:
                print(list_lines[i].id, len(running_text), list_lines[i-1].doc_length, correction_position, list_lines[i].texte_simple)
```

On obtient la position de la correction

Ensuite on vérifie que c'est une révision

Si le burst est un caractère alphanumérique

On va chercher et et

Comment?

On crée un token avec les détails de l'erreur

Et s'il y a des erreurs, on va investiguer les données manuellement

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":";  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

... à Nanterre. ...



index = 323  
word\_start = 323  
word\_end = 323

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":";  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

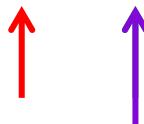
... à\_Nanterre. ...



index = 323  
word\_start = 322  
word\_end = 323

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":";  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

... à\_Nanterre. ...



index = 323  
word\_start = 321  
word\_end = 323

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":";  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

... à\_Nanterre. ...



index = 323  
word\_start = 321  
word\_end = 323

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":", ";"]  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

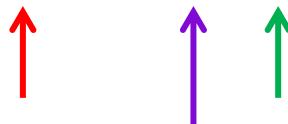
... à\_Nanterre. ...



index = 323  
word\_start = 321  
word\_end = 324

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":", ";"]  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

... à\_Nanterre. ...



index = 323  
word\_start = 321  
word\_end = 325

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":", ";"]  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

... à\_Nanterre. ...



The diagram shows the string "... à\_Nanterre. ..." followed by three arrows pointing upwards from the underscores in the string to the underscores in the variable assignments below.

index = 323  
word\_start = 321  
word\_end = 326

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":", ";"]  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

... à\_Nanterre. ...



index = 323  
word\_start = 321  
word\_end = 327

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":", ";"]  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

... à\_Nanterre. ...



index = 323  
word\_start = 321  
word\_end = 328

```
def get_word(index: int, chaine: str) -> Tuple[str, int, int]:  
    """Gets a word based on an index into a string."""  
    word_delimiters = [" ", ".", ",", ":", ";"]  
    word_start = index  
    word_end = index  
    if len(chaine) == 0:  
        return "", 0, 0  
    while word_start > 0:  
        if chaine[word_start-1] in word_delimiters:  
            break  
        word_start -= 1  
    while word_end < len(chaine) - 1:  
        if chaine[word_end] in word_delimiters:  
            break  
        word_end += 1  
    return chaine[word_start:word_end], word_start, word_end
```

... à\_Nanterre. ...



index = 323  
word\_start = 321  
word\_end = 328

```
# Add this line to the document text so far
running_text_after = add_burst_to_text(running_text, list_lines[i].texte_complete, list_lines[i].start_position)

if list_lines[i].start_position != list_lines[i-1].end_position and list_lines[i].start_position != len(running_text):
    correction_position = list_lines[i].start_position
    # if it's one character
    if len(list_lines[i].texte_simple.strip()) == 1:
        if list_lines[i].texte_simple.isalpha():
            # if it's just one letter
            try:
                word_before, start_of_word, _ = get_word(correction_position, running_text)
                word_after, _, _ = get_word(start_of_word, running_text_after)
                token = Token(
                    texte=list_lines[i].texte_simple,
                    pos_suppose="",
                    lemme=" ",
                    erreur=True,
                    details=f'Lettre unique appartenant à "{word_before}"',
                    pos_reel="Unknown",
                    correction=word_after,
                    ligne=list_lines[i]
                )
                token.pos_suppose = token.get_pos_suppose()
                token.lemme = token.get_lemme()
                tokens.append(token)
            except IndexError:
                print(list_lines[i].id, len(running_text), list_lines[i-1].doc_length, correction_position, list_lines[i].texte_simple)
```

Dans notre script on appelle cette fonction 2 fois:  
pour obtenir le mot avant qu'il soit corrigé, et après

La décision est

```
elif list_lines[i].texte_complete == " ":
    # if it's a space
    word_before, start_of_word, _ = get_word(correction_position, running_text)
    word_after, _, _ = get_word(start_of_word, running_text_after)
    token = Token(
        texte=list_lines[i].texte_simple,
        pos_suppose="",
        lemme=" ",
        erreur=True,
        details=f'Espace appartenant à "{word_before}"',
        pos_reel="SPACE",
        correction=word_after,
        ligne=list_lines[i]
    )
    token.pos_suppose = token.get_pos_suppose()
    token.lemme = token.get_lemme()
    tokens.append(token)
```

On obtient le mot avant  
et après la correction

On crée un token avec les  
détails de l'erreur

```
else:  
    # if it's multiple characters  
    string_length = len(list_lines[i].texte_complete)  
    if list_lines[i].texte_complete == "\u00a0" * string_length:  
        # if it's just backspaces on the entire line  
        word_after, _, _ = get_word(correction_position - string_length, running_text_after)  
        deleted_string = running_text[correction_position - string_length:correction_position]  
        token = Token(  
            texte=list_lines[i].texte_complete,  
            pos_suppose="",  
            lemme=" ",  
            erreur=True,  
            details=f"{string_length} backspaces supprimant '{deleted_string}'",  
            pos_reel="BACKSPACE",  
            correction=word_after,  
            ligne=list_lines[i]  
        )  
        token.pos_suppose = token.get_pos_suppose()  
        token.lemme = token.get_lemme()  
        tokens.append(token)
```

On obtient la chaîne après avoir supprimé des caractères

ne

On obtient la chaîne qui a été supprimée

On fait la même chose s'il s'agit d'un caractère delete (qui supprime les caractères qui suivent)

```
elif list_lines[i].texte_complete == "¤" * string_length:  
    # if it's just delete on the entire line  
    word_after, _, _ = get_word(correction_position, running_text_after)  
    deleted_string = running_text[correction_position : correction_position + string_length]  
    token = Token(  
        texte=list_lines[i].texte_complete,  
        pos_suppose="",  
        lemme=" ",  
        erreur=True,  
        details=f"{string_length} delete (¤) supprimant '{deleted_string}'",  
        pos_reel="DELETE",  
        correction=word_after,  
        ligne=list_lines[i]  
    )  
    token.pos_suppose = token.get_pos_suppose()  
    token.lemme = token.get_lemme()  
    tokens.append(token)
```

# Ajouter des mots entiers

D'abord il fallait définir qu'est-ce que c'est un mot dans le contexte de nos données

Il fallait donc utiliser le "texte simple" qui ne contenait pas les caractères de suppression ou les erreurs.

En ce qui concerne les mots qui ne sont pas des mots réels, nous n'avons pas trouvé une solution robuste, car on ne pouvait traiter que les mots faisant partie du lexique.

```
elif len(list_lines[i].texte_simple) > 0:  
    # if it's a word  
    begins_with_space = list_lines[i].texte_simple[0] == " "  
    ends_with_space = list_lines[i].texte_simple[-1] == " "  
    inserted_after_space = running_text[correction_position - 1] == " "  
    inserted_before_space = running_text[correction_position] == " "  
  
    if len(list_lines[i].texte_simple.split()) == 1:  
        # if it's one word, not just multiple letters  
        if (begins_with_space or inserted_after_space) and (ends_with_space or inserted_before_space):  
            previous_word, _, _ = get_word(correction_position - 1, running_text)  
            next_word, _, _ = get_word(correction_position+1, running_text)  
            token = Token(  
                texte=list_lines[i].texte_simple,  
                pos_suppose="",  
                lemme=" ",  
                erreur=True,  
                details=f"Mot inseré entre '{previous_word}' et '{next_word}'",  
                pos_reel=" ",  
                correction=f"{previous_word}{list_lines[i].texte_simple}{next_word}",  
                ligne=list_lines[i]  
            )  
            token.pos_suppose = token.get_pos_suppose()  
            token.pos_reel = token.pos_suppose  
            token.lemme = token.get_lemme()  
            tokens.append(token)
```

On initialise des variables qui vont nous aider à identifier s'il s'agit d'un mot et pas de plusieurs caractères

Le fonctionnement de la fonction est similaire aux cas précédents

# Dans le cas de plusieurs mots, on “tokenise” chaque mot de la chaîne

```
else:  
    # if it's multiple words  
    words = list_lines[i].texte_simple.split()  
    previous_word, _, _ = get_word(correction_position - 1, running_text)  
    next_word, _, _ = get_word(correction_position+1, running_text)  
    for word in words:  
        token = Token(  
            texte=word,  
            pos_suppose="",  
            lemme=" ",  
            erreur=True,  
            details=f"Partie de la chaîne '{list_lines[i].texte_simple}' insérée entre '{previous_word}' et '{next_word}'",  
            pos_reel= "",  
            correction=f"{previous_word}{list_lines[i].texte_simple}{next_word}",  
            ligne=list_lines[i]  
        )  
        token.pos_suppose = token.get_pos_suppose()  
        token.pos_reel = token.pos_suppose  
        token.lemme = token.get_lemme()  
        tokens.append(token)  
    pass
```

La différence c'est que dans ce cas on ne met pas l'étiquette "Unknown" car les mots sont des mots complets, « étiquetables » par des pos-taggers traditionnels



# Les erreurs dans la production

startPos	endPos	docLength	categ	charBurst
495	724	724	P	„beaucoup de médecine actuelle qui, même si elle s'avère efficace, utilise la moins chimique que pour la plupart du temps des médicaments plus ou moins nocifs que la maladie en elle-même.“
1332	1453	1454	P	recherches afin de découvrir de nouvelles maladies, mais aussi afin de trouver un antidote contre certaines dernières.
1453	1499	1500	P	Mais, au contraire, il est important de bien connaître la maladie afin de la soigner.
1499	1546	1547	P	de la médecine alternative, même si cette dernière dispose
1546	1637	1638	P	„et ce n'est pas toujours le cas de la médecine alternative, même si cette dernière dispose d'un remède pouvant être plus efficace qu'en médecine moderne.“
1637	1703	1704	P	aussi de remède pouvant être plus efficace qu'en médecine moderne.“

En fait il s'agit que de suppressions d'une ou plusieurs lettres à l'intérieur de la chaîne de caractères.

```
def deletion_within_word(line: Ligne) -> List[Token]:  
    """Returns a list of tokens where characters have been deleted within a word."""  
    list_tokens = []  
    line_words = line.texte_complete.split(" ")  
    for line_word in line_words:  
        if "¤" in line_word[1:-1] and len(line_word) > 1:  
            formed_word = ""  
            for char in line_word:  
                if char == "¤":  
                    formed_word = formed_word[:-1]  
                elif char == "¤":  
                    pass  
                else:  
                    formed_word += char  
            token = Token(  
                texte = line_word,  
                pos_suppose="",  
                lemme="",  
                erreur=True,  
                details="Suppression de caractères à l'intérieur d'un mot",  
                pos_reel="Unknown",  
                correction=formed_word,  
                ligne=line  
            )  
            token.pos_suppose = token.get_pos_suppose()  
            token.lemme = token.get_lemme()  
  
            list_tokens.append(token)  
  
    return list_tokens
```

**Ensuite, pour combiner les révisions et les productions, cette fonction est appelée dans la boucle où on itère sur les bursts de chaque personne.**

```
for list_lines in personnes.values():
# for list_lines in list(personnes.values())[1:]:
    running_text = list_lines[0].texte_simple
    tokens += deletion_within_word(list_lines[0])

for i in range(1,len(list_lines)):
    # Fix issue where some start positions are after the end of the text
    if list_lines[i].start_position > list_lines[i-1].doc_length:
        list_lines[i].start_position = list_lines[i-1].doc_length

    tokens += deletion_within_word(list_lines[i])

    # Add this line to the document text so far
    running_text_after = add_burst_to_text(running_text, list_lines[i].texte_complete, list_lines[i].start_position)

    if list_lines[i].start_position != list_lines[i-1].end_position and list_lines[i].start_position != len(running_text):
        correction_position = list_lines[i].start_position
        # if it's one character
        if len(list_lines[i].texte_simple.strip()) == 1:
            if list_lines[i].texte_simple.isalpha():
                # if it's just one letter
```

# 04

# Résultat de notre étiquetage

Ce qu'on obtient à la fin de notre script

Notre output est un fichier csv qui contient toutes les erreurs qui ont été traitées et les informations que nous avons extraites.

# L'évaluation de notre étiqueteur

## Comment pourrait-on évaluer nos résultats?

Du point de vue qualitatif, nous obtenons plus d'informations sur les données représentant le processus de taper un texte en temps réel.

Du point de vue quantitatif, on ne peut pas utiliser les mesures classiques (rappel, précision, f-mesure) parce que nous n'avons pas une version annotée à la main.

## Pourquoi ne pas annoter à la main?

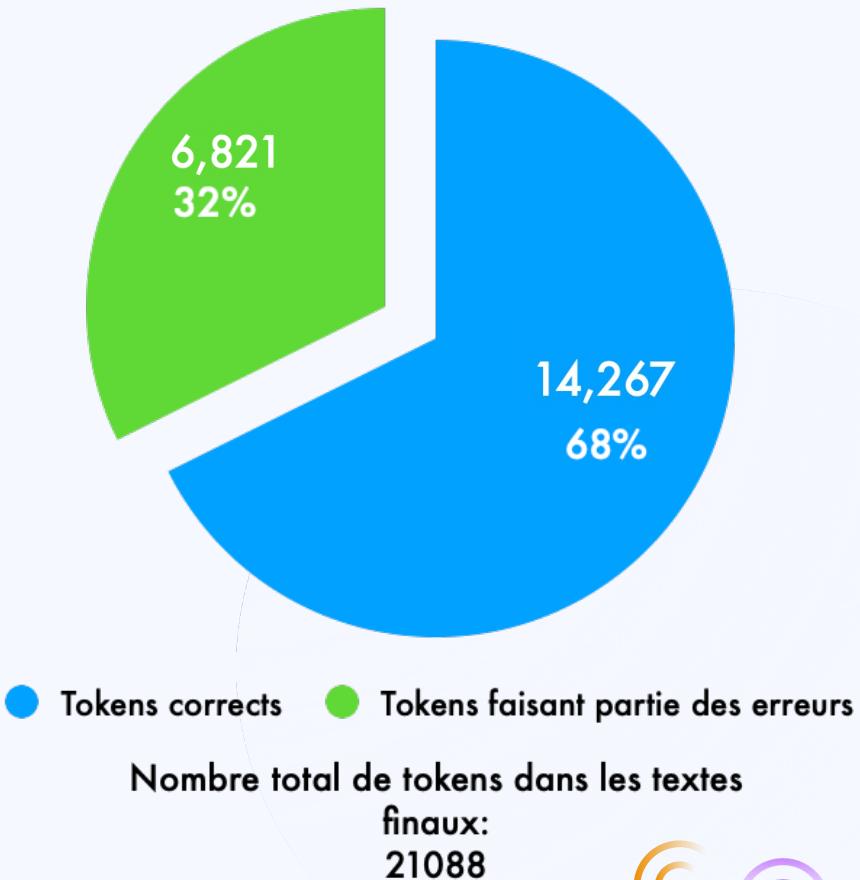
Sans prendre en considération le temps de l'annotation effective, nous n'avons pas de connaissances dans ce domaine de la recherche linguistique.

Nous devrions faire beaucoup de recherches pour savoir comment ce type de données est annoté en général, comment les grouper, s'il y a des logiciels plus adaptés etc.

En plus, notre script ne traite pas toutes les erreurs...

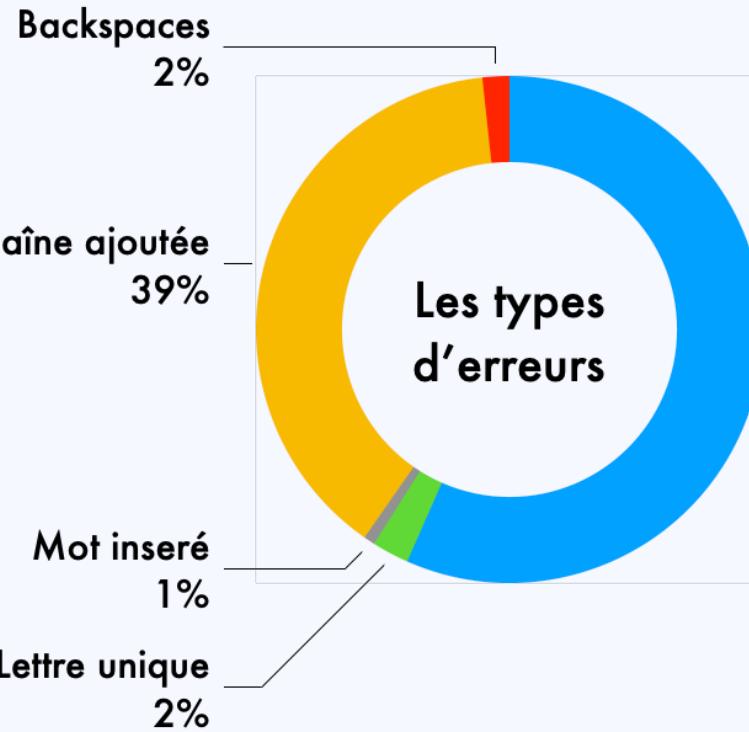
# Combien de tokens erronés?

Distribution tokens

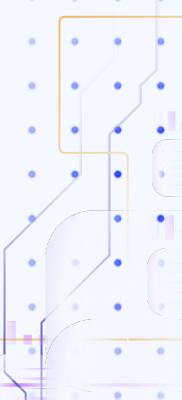




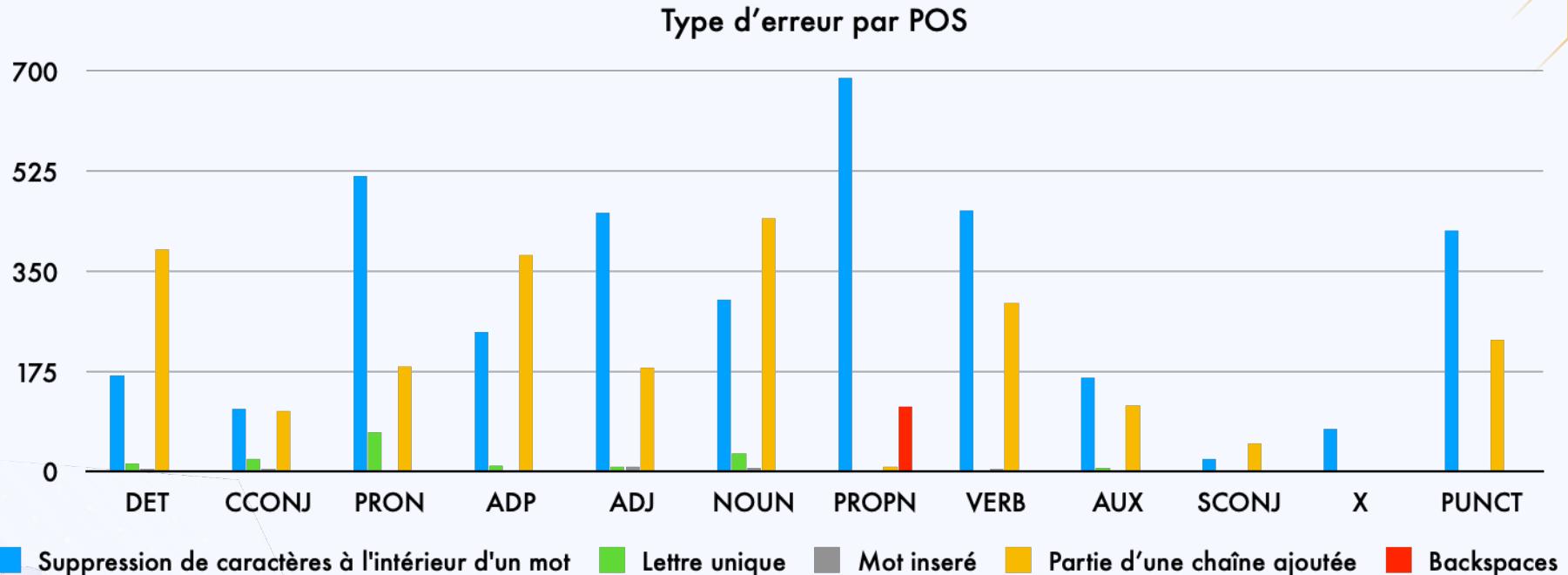
# La distribution des erreurs



Suppression de caractères à l'intérieur d'un mot  
57%



# Distribution des erreurs par POS



# 05

# Conclusion

---

Les limites de notre approche et travail futur

# Ce qu'on a réussi à faire



## Les erreurs de révision

Nous avons réussi à bien étiqueter les caractères et les mots ajouté plus tard dans le texte



## Les erreurs internes

Nous avons réussi à bien étiquetter les modifications dans le même mot



## Plus généralement

Nous avons créé un bon point de départ pour le traitement des autres erreurs

# Les limites de notre approche



## Spécifique

Notre script est très rigide et traite des données très spécifiques.



## Difficile à évaluer

C'est très difficile d'être sûr d'avoir tout traité.



## Les pauses

Nous ne prenons pas en considération les temps / la longueur des pauses

# Les difficultés rencontrées



## Erreurs importantes dans les données

Si on avait eu plus de temps, on aurait testé les données et potentiellement recréé les données à partir des fichiers XML.



## Manque de connaissance dans le domaine

La tâche est très difficile quand on ne connaît pas un algorithme de base. Aussi, il y a très peu de cas généraux, donc c'est très difficile d'écrire un script qui couvre tout.

# Ce qui reste à faire

## Trouver d'autres erreurs

Et optimiser le code qu'on a jusqu'ici

## Apprentissage automatique

Collecter plus de données, refaire ce travail avec machine learning et comparer les résultats



## Ajuster le output en fonction de besoins de la recherche

Ajouter des paramètres supplémentaires – par exemple pour obtenir un lien entre le type d'erreur et le temps de pause

## La recherche

Ce n'est qu'un outil – il faut faire la recherche suivant les résultats

# Merci!

**Vous avez des questions?**

<https://github.com/ims510/pos-tagger>