inalco
Institut national
des langues
et civilisations orientales

# MASTER TRAITEMENT AUTOMATIQUE DES LANGUES

## Ingénierie des connaissances

Rapport

**Mise à disposition d'un corpus annoté pour étudier la variation sociolinguistique diastratique**

Étudiants :

**Kenza Ahmia**
**Florian Jacquot**
**Tifanny Nguyen**
**Shami Thirion Sen**

Année universitaire 2023/2024

# Table des matières

# 1 Introduction

*Contribution : Florian*

Le présent rapport a pour objet de présenter le travail effectué dans le cadre d'un projet de recherche visant à étudier la variation sociolinguistique diastratique dans deux types de données : des données issues de l'oral et de la communication médiée par les réseaux. Notre équipe est chargée de contribuer à l'avancement des connaissances dans ce domaine. Tous les scripts, corpus brut et annotés se trouvent sur le git du projet [1].

Les variables étudiées dans le cadre de ce projet sont les temps verbaux, la négation complète et incomplète, ainsi que l'absence ou la présence de /y/ dans « tu » et du schwa dans les clitiques. Pour mener à bien cette étude, nous avons suivi un processus en six étapes :

1. faire l'inventaire des sources de connaissances disponibles (corpus)
2. acquisition des connaissances (corpus)
3. modélisation formelle des connaissances acquises
4. modélisation formelle des connaissances à ajouter (modèles d'annotation)
5. ajout de nouvelles connaissances (annotation automatique)
6. formalisation des connaissances obtenues (création d'une nouvelle version du corpus)

Dans ce rapport, nous détaillerons chacune de ces étapes et présenterons les résultats obtenus. Nous expliquerons également les choix méthodologiques que nous avons faits et les éventuelles difficultés rencontrées. Enfin, nous présenterons les nouvelles versions des corpus ESLO et 88milSMS, que nous avons généré dans le cadre de ce projet.

## Répartition des tâches

Note : les contributions indiquées sous chaque titre de partie du rapport ne concernent que les contributions à la rédaction et n'ont rien à voir avec le travail fait en amont. Pour ce travail en particulier, la répartition fut plutôt la suivante :

— Extraction des connaissances du corpus ESLO : **Tifanny**
— Extraction des connaissances du corpus 88milSMS : **Shami**
— Annotation des corpus : **Kenza et Florian**
— Sortie XML : **Kenza et Tifanny**
— Exportation TXM : **Kenza et Florian**

Mais pour chacune de ces étapes, tous les membres du groupe ont participé à la réflexion.

1. `https://github.com/sham-glam/Ingenierie_Connaissance.git`

# 2 Présentation des corpus

*Contribution : Florian*

Le travail a commencé par une recherche de deux corpus de natures différentes pour avoir des données contrastées. Les corpus retenus furent les corpus ESLO et 88milSMS. Les corpus étant plutôt compliqué à récupérer, notre professeur Loïc Liegeois nous les a fournis.

## 2.1 Corpus ESLO

Le corpus ESLO – Enquête SocioLinguistique à Orléans – est un corpus de données linguistiques orales retranscrites, recueillies dans le cadre d'une enquête sociolinguistique menée à Orléans, en France. Il a été créé par une équipe de chercheurs en linguistique et en sociolinguistique et est destiné à être utilisé pour l'étude de la variation linguistique dans la ville d'Orléans. Le corpus a été constitué en deux périodes. Un premier travail a été fait entre 1969 et 1974, corpus ESLO1 et un second en 2014, ESLO2. Étant donné que le projet se concentre sur la période contemporaine, seul le corpus ESLO2 est utilisé ici.

Le corpus comprend quatre-vingt quatre fichiers XML, correspondants à quatre-vingt quatre enregistrements audio, ainsi qu'un fichier `tsv` avec des métadonnées supplémentaires. Dans les fichier XML, chaque unité de parole possède un attribut indiquant l'identifiant du locuteur. Par exemple, l'unité donnée ci-dessous nous indique que le locuteur avec l'identifiant BV1 a énoncé « d'accord » entre 600.637s et 600.862s de l'enregistrement.

```
<u who="BV1" start="600.637" end="600.862">d'accord</u>
```

De son côté, le fichier `tsv` contient des métadonnées sur les locuteurs tels que leur niveaux d'étude, ou leur situation familiale. Pour les besoins du projet, seuls les énoncés, les identifiants et les niveaux d'étude furent conservés (voir section **??**).

## 2.2 Corpus 89milSMS

Le corpus 88milSMS est un corpus de données linguistiques écrites, constitué de plus de 88 000 SMS en français, collectés en 2011 pour dans le cadre du projet sud4science LR [2]. Lors de la collecte, des données sociolinguistiques ont été recueillies sur les auteurs des SMS. Le corpus se présente sous la forme d'un unique fichier XML au format de balisage TEI, contenant toutes les informations recueillies. Le format d'encodage TEI permet de faire des relations entre les différents éléments contenus dans le fichier. Ainsi, ce dernier présente plusieurs sections. Une première, présente des métadonnées avec toutes les informations sur les auteurs des SMS, dont l'identifiant les représentant, le niveau d'éducation, l'âge ou encore

2. `http://88milsms.huma-num.fr/`

| Corpus | Nombre tokens |
|---|---|
| ESLO | 1 131 986 |
| 88milSMS | 932 471 |

Table 1 – Nombre de tokens des corpus ESLO2 et 88milSMS utilisés.

le type de téléphone utilisé pour écrire les SMS. Une seconde section présente les SMS qui sont associés aux identifiants des auteurs. Comme pour le corpus ESLO2, nous avons extrait de ce document les énoncés des SMS, les identifiants et les niveaux d'études.

Contrairement au corpus ESLO2, qui est constitué de données orales retranscrites avec des règles définies, le corpus 88milSMS est constitué de données écrites, avec toutes les particularités orthographiques et syntaxiques que cela implique. Par exemple, les SMS peuvent contenir des abréviations qui ne seront pas présentes dans les retranscriptions. Ces particularités peuvent être intéressantes à étudier en elles-mêmes, mais elles peuvent aussi compliquer la tâche d'annotation. Le corpus ESLO2, quant à lui, peut également amener son lot de difficultés pour l'annotation, avec les erreurs de syntaxes qui apparaissent à l'oral du fait des répétitions, des hésitations, des coupures de phrases etc.

Nous présentons dans le tableau 1 le nombre de tokens totaux de chacun de ces deux corpus.

# 3 Réflexion et parcours suivi.

*Contribution : Tifanny, Florian*

Le projet, comme indiqué précédemment, nous demandait d'identifier dans les deux corpus les temps verbaux, la négation complète et incomplète, l'absence ou la présence de /y/ dans « tu » et du schwa.

Avant de nous mettre à faire des scripts, nous avons pris un temps de réflexion afin de pouvoir coordonner le travail sur les deux corpus et également trouver comment faire pour identifier tout ce qui nous a été demandé.

## 3.1 Temps verbaux

Pour les temps verbaux, il nous fallait un outil qui permettait d'identifier les verbes mais aussi de nous donner son temps. Le travail étant assez clair, il n'y avait rien de spécial à identifier si ce n'est que l'outil que nous allions utiliser. Au début, nous nous sommes penchés sur TreeTagger qui semblait pouvoir faire le travail. Cependant, ayant rencontrés de nombreux problèmes à son installation, nous avons opté pour Spacy. Dans le cadre de ce travail, nous avons choisi de travailler avec le modèle français le plus léger de Spacy mais il est tout à fait possible de changer ce paramètre dans le script d'annotation A.1 pour obtenir un meilleurs résultat.

Ainsi, pour obtenir une annotation sur les temps verbaux, nous avons utilisé l'annotation en partie du discours de Spacy qui nous permet d'avoir ces informations.

## 3.2 Négation

Il y a deux types de négation : la complète et l'incomplète. La différence entre les deux est le « ne » ou « n' » qui n'est pas présent dans la négation incomplète. Notre professeur Loïc Liegeois nous avait déjà identifié plusieurs tournures de négations :

— Il (ne) ... pas.
— Il (ne) ... point.
— Il (ne) ... rien.
— Il (ne) ... jamais.
— etc...

Il nous fallait juste un moyen de les détecter sans les lister un par un. Pour la négation complète, nous avons remarqué que dans la plupart des cas, lors d'une négation complète, le verbe est entouré par deux adverbes : « ne » et « pas » par exemple. Nous avions donc décidé d'utiliser l'annotation en partie du discours déjà implémentée pour la recherche des temps verbaux afin de pouvoir identifier la négation complète plus facilement.

Cette annotation, qui aurait pu se faire avec TreeTagger, a finalement été faite avec Spacy.

## 3.3 Le /y/ et schwa

L'absence ou la présence du /y/ dont nous parlons ici est celle qui accompagne le « tu » : « Tu as pris » qui se transforme en « T'as pris ». Quant au schwa, c'est l'absence du « e » dans les clitiques « ce, de, je, le, me, ne, que, se, te » dont la liste nous a été donnée par notre professeur.

Aucun outil spécifique ne permet de faire ce type d'annotation, nous avons donc défini notre propre méthode pour la réaliser. Il est à noter que le corpus ESLO ne présente pas d'absence de schwa du fait que les retranscriptions ont ignoré cette information linguistique par simplicité. L'annotation sur les schwa n'est donc présente que sur le corpus 88milSMS.

La méthode retenue ici pour effectuer cette annotation est l'utilisation de motifs – expressions régulières. En effet, ces deux variables possèdent dans une grande majorité des cas, des similarités de formes qui ne se croisent pas entre les deux cas, ce qui permet de distinguer les deux variables.

Dans le cas de l'absence de /y/ dans le « tu », nous avons remarqué que, dans la grande majorité des cas, la modification se produit – dans ce corpus – lorsque le mot suivant « tu » commence par une voyelle comme « tu as » ou « tu es » qui deviennent « t'as » (ou « tas ») et « t'es » (ou « tes ») respectivement. Le motif est donc « t' + mot commençant par une voyelle ». Cependant, cette condition est trop large car la forme « t'es » existe aussi dans

la langue – par exemple, dans la phrase « Tu t'es perdue ? ». Nous avons donc ajouté une condition en mettant à profit notre utilisation de Spacy qui nous permet de récupérer la relation de dépendance des tokens par rapport à la tête. Ici, puisque nous sommes dans le cas d'une relation sujet, le motif utilisé est plutôt « t' de dépendance sujet + mot commençant par une voyelle ».

### 3.4 Les formats de données d'entrée et de sortie

Maintenant que les outils d'identification des connaissances sont définis 3, il nous faut définir les formats de données que nous utilisons et expliquer les raisons derrière les choix réalisés.

Pour commencer, comme présenté dans la section sur les corpus 2, les données d'entrée étaient en format XML – et csv pour le corpus ESLO. Ce format étant universel, simple et à la base de beaucoup d'applications gérant des données, nous avons choisi de viser un format de sortie XML. De plus, devant mettre à disposition le corpus annoté sur un outil facilement utilisable par les futurs utilisateurs, ce format est idéal, notamment pour l'outil de textométrie TXM – qui sera l'outil de formalisation des connaissances obtenues de ce projet.

## 4 Méthodes utilisées

*Contribution : Kenza*

Comme indiqué précédemment, après des réflexions approfondies sur les outils à utiliser, les formats d'entrée et de sortie, et la manière de modéliser nos connaissances pour traiter le corpus, nous avons commencé par récupérer les deux corpus depuis leurs sources.

Nous avons soigneusement sélectionné les fichiers XML de la manière suivante : pour le corpus SMS, il n'y avait qu'un seul fichier contenant à la fois les métadonnées et les énoncés (ID des locuteurs et niveaux d'étude). Pour le corpus ESLO, en plus des nombreux fichiers d'énoncés qui contiennent uniquement les ID des locuteurs, nous avons récupéré un fichier de métadonnées pour compléter l'information sur le niveau d'étude des locuteurs.

Tous ces fichiers ont été stockés dans un dépôt Git du projet, accessible à tous les membres de l'équipe.

1. Préparation des données d'entrée

   La préparation des données a été réalisée grâce à des scripts Python, dont le code est disponible en annexe.

   (a) Extraction des fichiers textes : Pour chaque corpus, nous avons extrait trois fichiers : un fichier texte contenant tous les énoncés avec un énoncé par ligne. Un deuxième fichier avec les identifiants des locuteurs dans le même ordre. Enfin, nous

avons récupéré les fichiers des niveaux d'études des locuteurs et leurs id correspondants.

(b) Correspondance des données : Ces fichiers représentent la deuxième version du corpus que nous prenons en entrée de notre script d'annotation et qui nous permettent de faire correspondre les énoncées à annoter avec les locuteurs ainsi que leurs niveaux d'étude.

2. Annotation des données

Les données ont été annotés à l'aide d'un script python utilisant l'outil spaCy [3], une bibliothèque gratuite et open source publiée sous la licence MIT pour le traitement naturel du langage. Les annotations incluent les parties du discours (POS) incluant les temps verbaux, les négations complètes, les absences `y` et la présence de `schwa`. Le processus s'est déroulée en plusieurs étapes détaillées ci-dessous :

(a) Chargement des données

Les fichiers textes contenant les énoncées du corpus ont été importés dans le script Python à l'aide de la bibliothéque de manipulation de fichiers `sys`. Une fonction de lecture a été utilisé pour extraire les énoncés, les identifiants des locuteurs ainsi que les niveaux d'études, et ont été stockés dans des variables pour une utilisation ultérieur.

(b) Traitement des données

Chaque énoncé a été analysé avec spaCy pour extraire les tokens et leurs parties du discours. La bibliothèque a été principalement utilisé pour annoter les temps verbaux grâce à sa fonction `token.morph`. Pour détecter les négations complètes, ainsi que les absences ou présences de 'y' et les 'schwa', nous avons opté pour l'utilisation des expressions régulières (`regex`) en complément car elles permettent de spécifier de manière précise les éléments que nous souhaitons identifier.

(c) Identification et annotation des données :
- **Détection des temps verbaux :** comme spécifié dans la partie Réflexion et parcours suivi, la détection des temps verbaux a été effectué avec la fonction `token.morph` de spaCy.
- **Détection des négations complètes :** pour celle-ci, nous avons d'abord élaboré deux listes : une comprenant toutes les formes de négation couramment utilisées, telles que "ne", "n'", et "n", et une autre pour tous les adverbes de négation. Cela nous a permis de couvrir un large éventail de contextes où la négation peut apparaître. Ensuite, en utilisant des expressions régulières, nous avons parcouru chaque énoncé pour identifier les occurrences de ces formes de négation. Les regex nous ont permis de rechercher des motifs spécifiques dans le texte, comme la présence de "ne" suivi immédiatement d'un adverbe de

3. `https://spacy.io/usage`

négation tel que "pas", "rien", ou "jamais". Une fois qu'une négation complète a été identifiée, elle a été annotée dans notre système d'annotation sous forme de booléen pour indiquer sa présence.

— **Détection de l'absence de `y` :** dans les contractions de "tu" en "t'", dans les données, nous utilisons une fois de plus la bibliothèque `regex` pour effectuer des correspondances d'expressions régulières sur le texte. Le code parcourt chaque token (mot ou caractère) dans le document. Si le token est "tu" et qu'il est suivi d'une voyelle, le code considère que le "y" est présent et ajoute la valeur `False` à la liste `Yabsent` dans le dictionnaire d'annotations. Si le token correspond à "t'" et est suivi d'une voyelle, le code considère que le "y" est absent et ajoute la valeur `True` à la liste.

— **Détection de l'absence du `schwa` :** Pour détecter l'absence du `schwa`, le script utilise la même bibliothèque que pour les détections précédentes et cela en parcourant chaque token (mot ou caractère) dans le corpus. Si le token correspond à un clitique se terminant par une apostrophe (par exemple "c'", "d'", "j'", "l'", "m'", "n'", "q'", "s'", "t'") et qu'il est suivi d'un caractère qui n'est pas une voyelle ou "h", le code considère que le `schwa` est absent et ajoute la valeur `True` à la liste d'annotations. De même, si le token est "j" et qu'il est suivi d'un caractère qui n'est pas "v", "f", "t", "p", "s", "d", "c", le code considère que le `schwa` est absent et ajoute également la valeur `True`.

3. Génération des données en sortie

Pour la génération de données en sortie nous avons opté pour un format dictionnaire de dictionnaire pour les résultats d'annotations que nous avons ensuite transformé en fichier XML.

(a) Structuration des données en dictionnaire

Le choix du stockage des annotations dans un dictionnaire en Python repose sur sa structure de données, qui stocke des paires clé-valeur. Chaque clé est unique et les valeurs peuvent être de n'importe quel type.

Dans ce cas, chaque clé du dictionnaire `annotation` est une chaîne de caractères représentant un identifiant unique pour chaque énoncé du corpus, par exemple "eslo 1", "eslo 2", etc.

La valeur associée à chaque clé est un autre dictionnaire contenant des informations sur l'énoncé correspondant. Ce dictionnaire interne a plusieurs clés telles que `source`, `nv_etudes`, `tokens`, `pos`, `neg_comp`, `y_absent`, et `schwa_absent`. Chaque clé a une liste comme valeur. Les listes `y_absent` et `schwa_absent` contiennent des booléens (`True`, `False`) ou `None`, indiquant respectivement si le phonème /y/ ou le schwa sont absents dans l'énoncé, ou si l'information n'est pas disponible.

En plus de cela, nous avons d'autres clés : `source` qui représente l'identifiant du locuteur, `nv_etudes` pour son niveau d'études, ainsi que les `tokens` et les `pos` de chaque token de l'énoncé.

Voici un exemple du format du dictionnaire :

```
annotation =
{
"eslo 1": {
    "source": "ch_OB1",
    "nv_etudes": "bac_5_et_plus",
    "tokens": ["alors"],
    "pos": ["ADV"],
    "neg_comp": [],
    "y_absent": [None],
    "schwa_absent": [None]
    },
"eslo 2": {
    "source": "BV1",
    "nv_etudes": "bac_1",
    "tokens": ["marche", "bien", "au", "moins", "votre", "truc", "euh"],
    "pos": ["VERB:Pres", "ADV", "ADP", "ADV", "DET", "NOUN", "ADV"],
    "neg_comp": [],
    "y_absent": [None, None, None, None, None, None, None],
    "schwa_absent": [None, None, None, None, None, None, None]
    }
}
```

Dans cet exemple, pour l'énoncé "eslo 1", le locuteur est "ch_OB1", son niveau d'études est "bac_5_et_plus", l'énoncé contient un seul token "alors" qui est un adverbe (`ADV`), il n'y a pas de négation complète, et l'information sur l'absence du phonème /y/ et du schwa n'est pas disponible (`None`).

Pour l'énoncé "eslo 2", le locuteur est "BV1", son niveau d'études est "bac_1", l'énoncé contient sept tokens avec leurs parties du discours correspondantes, il n'y a pas de négation complète, et l'information sur l'absence du phonème /y/ et du schwa pour chaque token n'est pas disponible (`None`).

(b) Création et formatage du fichier XML

Le format XML est un standard largement utilisé pour structurer, stocker et transporter des données. Il permet de représenter les données de manière hiérarchique et lisible aussi bien par les humains que par les machines.

Le fichier XML généré dans le cadre de notre projet suit une structure bien définie. Pour commencer, il est stocké dans une variable `<DATA>` dans laquelle chaque énoncé du corpus est représenté par un élément unique : `<SMS>` ou `<ESLO>`, selon le corpus, qui contient des attributs pour l'identifiant du locuteur (`id`) et son niveau d'études (`niveau`).

À l'intérieur de chaque élément, les tokens (mots ou caractères) de l'énoncé sont encapsulés dans des sous-éléments `<w>` avec leurs attributs respectifs : les parties

du discours (`pos`), les informations sur les négations complètes (`neg_complete`), l'absence du phonème /y/ (`y_absent`) et l'absence du schwa (`schwa`).

Voici un exemple du fichier XML avec les annotations :

```
<DATA>
    <SMS id="cmr-88milsms-p236" niveau="inconnu">
        <w pos="PROPN">Dors</w>
        <w pos="ADV">bien</w>
        <w pos="CCONJ">et</w>
        <w pos="ADV">n'</w>
        <w pos="VERB:Pres">oublie</w>
        <w pos="ADV" neg_complete="True">pas</w>
        <w pos="ADP" schwa="True">d'</w>
        <w pos="VERB:Inf">manger</w>
        <w pos="NUM">5</w>
        <w pos="NOUN">fruits</w>
        <w pos="CCONJ">et</w>
        <w pos="NOUN">legumes</w>
        <w pos="ADP">par</w>
    </SMS>
    <SMS id="cmr-88milsms-p136" niveau="inconnu">
        <w pos="PRON">Je</w>
        <w pos="ADV">n'</w>
        <w pos="VERB:Pres">arrive</w>
        <w pos="ADV" neg_complete="True">pas</w>
        <w pos="ADP">à</w>
    </SMS>
</DATA>
```

# 5 Résultat et retour d'expérience

*Contribution : Florian, Shami*

## 5.1 Observation des résultats

Les annotations que nous avons effectuées semblent relativement satisfaisantes. Nous aurions souhaiter évaluer de façon quantitative le travail effectué mais le manque de temps nous en a empêché.

D'après nos observations, les annotations faites semblent assez justes mais incomplètes. En effet, les règles que nous avons définies pour capturer les caractéristiques souhaitées sont

suffisamment sélectives pour éviter les grosses erreurs d'annotation mais certainement trop sélectives pour être exhaustives dans nos captures. De plus, l'annotation Spacy pourrait être meilleure si nous utilisions un plus gros modèle mais l'exécution de nos script étant longues sur nos machines, nous sommes restés sur le plus petit modèle.

L'extraction et l'uniformisation des données ont nécessité une réflexion en amont. Étant un groupe de quatre, nous avons fait des modifications à plusieurs reprises pour que les données extraits soient exploitables par le script d'annotation.

Néanmoins, malgré plusieurs essais et corrections des expressions régulières et du script d'annotation, nous n'avons pas réussi écarter certaines erreurs, surtout en ce qui concerne les annotation des schwa. Par exemple, le token « enchainement » est annoté alors qu'il ne correspond pas du tout aux conditions de captures d'absence de schwa définies dans la section 4.

### 5.2 Sorties

Les démarches réalisées pour extraire et modéliser les information nous ont permis dans un premier temps d'organiser et de présenter des informations que nous avons estimées pertinentes, sous des divers formats. Ainsi, nous avons en sortie cinq corpus :

— eslo_annotation.txt
— eslo_annotation.xml
— SMS_annotation.xml
— ESLO_annotation.txm
— SMS_annotation.txm

Ces données peuvent être exploitées afin d'effectuer des études statistiques des phénomènes sociolinguistiques plus avancées. Pour en citer un exemple, les corpus xml peuvent servir à faire des calculs de corrélation entre le niveau d'étude et les types de termes utilisés : des personnes avec un certain niveau d'étude sont plus ou moins susceptibles d'utiliser le schwa.

## 6 Conclusion

*Contribution : Tout le monde*

En conclusion, ce projet de recherche nous a permis d'étudier la variation sociolinguistique diastratique dans deux types de données : des données issues de l'oral et de la communication médiée par les réseaux. Les deux corpus ont été annotés en utilisant des outils tels que Spacy et des expressions régulières et les résultats obtenus ont été formalisés dans de nouvelles extensions : XML et TXM.

Ce projet nous a permis d'en apprendre plus sur ce qu'est de travailler sur la chaîne

complète de travail sur l'ingénierie des connaissances : récupération des connaissances, enrichissement / ajout de nouvelles connaissances et leur transmission. Nous avons également pu constater que ce processus demande de la réflexion et de la flexibilité. Par exemple, trouver une manière d'identifier les schwas n'a pas été de tout repos et le résultat n'est toujours pas parfait.

Surtout , cela nous a permis d'entrer de plein-pied dans le domaine du travail d'ingénieur et de réaliser notre capacité à naviguer entre la linguistique et l'informatique, en réalisant ce projet en un temps limité, tout en nous adaptant à diverses situations.

Il est vrai qu'avec plus de temps et de ressources nous aurions pu enrichir nos observations sur les résultats extraits, en faisant une analyse liée au niveau d'études, par exemple. Néanmoins, nous réussi à aboutir à des résultats plutôt satisfaisants.

# A Scripts Python utilisés

Les scripts pour l'annotation des corpus (A.1, A.2) sont identiques au niveau de l'annotation. La différence entre les deux est que dans le script d'annotation des SMS, il y a une partie pour écrire la sortie au format XML alors que pour ESLO, il y a un script à part.

## A.1 Script pour l'annotation du corpus SMS et faire la sortie XML

Nom : `annotation-complete.py`

```python
import sys
import spacy
import regex
import xml.etree.ElementTree as ET

# Vérifie que le nom du corpus à traiter est bien fourni en argument
if len(sys.argv) < 2:
    print(
        "Veuillez fournir le corpus que vous souhaitez traiter: eslo ou sms \
        \nExemple d'exécution : python3 annotation.py eslo\n"
    )
    sys.exit(1)

# Récupère le nom du corpus à traiter
corpus = sys.argv[1]

# Ouvre les fichiers contenant les textes et les identifiants et les niveaux d'étude des locuteurs
if corpus == "eslo":
    texts = open("../data/transformes/xml-ESLO_contenu/concat_contenu.txt", "r")
    ids = open("../data/transformes/xml-ESLO_id/concat_id.txt", "r")
    niv = open("../data/transformes/tsv/eslo_id_etudes.tsv", "r")
elif corpus == "sms":
    texts = open("../data/transformes/xml-SMS_contenu/SMS_contenu.txt", "r")
    ids = open("../data/transformes/xml-SMS_id/SMS_ids.txt", "r")
    niv = open("../data/transformes/tsv/SMS_id_education.tsv", "r")
else:
    print("Corpus non reconnu")
    sys.exit(1)

# Charge le modèle spacy pour le français
nlp = spacy.load("fr_core_news_md")
print("Modèle spacy chargé...")

```

```
# Initialise le dictionnaire qui va contenir les niveaux d'étude
source_niveau = {}

for line in niv:
    source, niveau = line.split("\t")
    source_niveau[source] = niveau.rstrip()

# reinitialise the file pointer
niv.seek(0)

# Initialise le dictionnaire qui va contenir les annotations
annotation = {}

# Liste des formes de négation et des adverbes de négation
list_neg = ["ne", "n'", "n"]
list_adv_neg = [
    "pas",
    "rien",
    "jamais",
    "point",
    "aucunement",
    "aucun",
    "aucunement",
    "nul",
    "nullement",
    "nulle",
    "plus",
]
list_neg_tok = list_neg + list_adv_neg

# Boucle sur les textes et les identifiants des locuteurs
n = 1
for t, i in zip(texts, ids):
    # Initialise la variable qui va contenir l'information sur la négation complète
    complete_neg = None

    # Affiche un message tous les 1000 textes traités
    if n % 5000 == 0:
        #break
        print(f"working on eslo {n}...")
    #elif n % 1000 == 0:
        #print(f"working on eslo {n}...")

    # Ajoute les informations sur le texte
    annotation[f"eslo {n}"] = {}
    annotation[f"eslo {n}"][f"source"] = i.strip()
```

```
80
81      # Récupère le niveau correspondant à la source
82      niveau = source_niveau.get(i.strip())
83      if niveau:
84          annotation[f"eslo {n}"][f"niveau"] = niveau
85      else:
86          annotation[f"eslo {n}"][f"niveau"] = "inconnu"
87
88      # Initialisation des annotations
89      annotation[f"eslo {n}"]["tokens"] = []
90      annotation[f"eslo {n}"]["pos"] = []
91      annotation[f"eslo {n}"]["neg_comp"] = []
92      annotation[f"eslo {n}"]["y_absent"] = []
93      annotation[f"eslo {n}"]["schwa_absent"] = []
94
95      # Vérifie si le texte contient une négation complète
96      complete_negation_in_sent = None
97      if
        ↪  regex.findall(r"\b(ne|n')\b.*\b({0})\b".format("|".join(list_adv_neg)),
        ↪  t):
98          complete_negation_in_sent = True
99
100     # Traite le texte avec spacy et boucle sur les tokens
101     doc = nlp(t)
102     for j, token in enumerate(doc):
103         # Ignore les tokens vides
104         if token.pos_ != "SPACE":
105             # Ajoute le token et sa catégorie grammaticale au dictionnaire
                ↪  d'annotations
106             annotation[f"eslo {n}"]["tokens"].append(token.text)
107             complete_pos = token.pos_
108             if token.pos_ == "VERB":
109                 if len(token.morph.get("Tense")) > 0:
110                     complete_pos = token.pos_ + ":" +
                        ↪  token.morph.get("Tense")[0]
111                 else:
112                     complete_pos = token.pos_ + ':' + 'Inf'
113             annotation[f"eslo {n}"]["pos"].append(complete_pos)
114
115             # Ajoute l'information sur la négation complète au dictionnaire
                ↪  d'annotations
116             if complete_negation_in_sent:
117                 if token.text in list_neg:
118                     position = len(annotation[f"eslo {n}"]["tokens"]) - 1
119                     complete_neg = True
120
121                 if (
```

```
                    complete_neg
                    and token.text in list_adv_neg
                    and regex.match(
                        r"(^VERB.*|AUX)", annotation[f"eslo
                        ↪ {n}"]["pos"][position + 1]
                    )
                ):
                    annotation[f"eslo {n}"]["neg_comp"].append(True)
                else:
                    annotation[f"eslo {n}"]["neg_comp"].append(None)

            assert token.text == doc[j].text
            # Ajoute l'information sur l'absence de /y/ dans "tu" au
            ↪ dictionnaire d'annotations
            if (
                j < len(doc) - 1
                and token.dep_ == "nsubj"
                and regex.match(r"t\'", token.text)
                and regex.match(r"[aeiouy]", doc[j + 1].text)
            ):
                if token.text == "tu":
                    annotation[f"eslo {n}"]["y_absent"].append(False)
                else:
                    annotation[f"eslo {n}"]["y_absent"].append(True)

            # Ajoute l'information sur l'absence de schwa dans les clitiques
            ↪ au dictionnaire d'annotations
            elif (
                j < len(doc) - 1
                and regex.match(r"[cdjlmnqst]\'", token.text)
                and regex.match(r"^[^aeiouyàâéèêôhAEIOUYÉÀÂÊÔÈH]", doc[j +
                ↪ 1].text)
            ):
                annotation[f"eslo {n}"]["schwa_absent"].append(True)

            elif (
                j < len(doc) - 1
                and regex.match(r"^j[^aeiouyàâéèêôhAEIOUYÉÀÂÊÔÈH\']",
                ↪ token.text)
            ):
                annotation[f"eslo {n}"]["schwa_absent"].append(True)
            else:
                annotation[f"eslo {n}"]["y_absent"].append(None)
                annotation[f"eslo {n}"]["schwa_absent"].append(None)
    n += 1

# print(annotation)
```

```

texts.close()
ids.close()

# Generate XML
root = ET.Element("DATA")

for key, value in annotation.items():
    sms = ET.Element(
        corpus.upper(), id=value["source"], niveau=value["niveau"]
    )  # Utilisation de la variable corpus pour le nom de l'élément
    has_content = False
    for token, pos, neg, y, schwa in zip(
        value["tokens"],
        value["pos"],
        value["neg_comp"],
        value["y_absent"],
        value["schwa_absent"],
    ):
        attributes = {"pos": pos}
        if neg is not None:
            attributes["neg"] = str(neg)
        if y is not None:
            attributes["y"] = str(y)
        if schwa is not None:
            attributes["schwa"] = str(schwa)
        ET.SubElement(sms, "w", **attributes).text = token
        has_content = True
    if has_content:
        root.append(sms)


# Prettify the XML
def indent(elem, level=0):
    i = "\n" + level * "    "
    if len(elem):
        if not elem.text or not elem.text.strip():
            elem.text = i + "    "
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
        for elem in elem:
            indent(elem, level + 1)
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
    else:
        if level and (not elem.tail or not elem.tail.strip()):
            elem.tail = i
```

```

indent(root)

# Write the pretty XML to a file
output_file = f"../data/annote/{corpus.upper()}_annotation.xml"  # Utilisation de la variable corpus pour le nom du fichier de sortie
tree = ET.ElementTree(root)
with open(output_file, "wb") as fh:
    tree.write(fh, encoding="utf-8", xml_declaration=True)

print(
    f"Fichier XML joliment formaté pour le corpus {corpus.upper()} généré avec succès."
)

```

## A.2 Script pour l'annotation du corpus ESLO

Nom : `annotation.py`

```
import sys
import spacy
import regex

# Vérifie que le nom du corpus à traiter est bien fourni en argument
if len(sys.argv) < 2:
    print(
        "Veuillez fournir le corpus que vous souhaitez traiter: eslo ou sms \
        \nExemple d'exécution : python3 annotation.py eslo\n"
    )
    sys.exit(1)

# Récupère le nom du corpus à traiter
corpus = sys.argv[1]

# Ouvre les fichiers contenant les textes et les identifiants et les niveaux d'étude des locuteurs
if corpus == "eslo":
    texts = open("../data/transformes/xml-ESLO_contenu/concat_contenu.txt", "r")
    ids = open("../data/transformes/xml-ESLO_id/concat_id.txt", "r")
    nv_etudes = open("../data/transformes/xml-eslo_nvetudes.txt", "r")
elif corpus == "sms":
    texts = open("../data/transformes/xml-SMS_contenu/SMS_contenu.txt", "r")
```

```
    ids = open("../data/transformes/xml-SMS_id/SMS_ids.txt", "r")
else:
    print("Corpus non reconnu")
    sys.exit(1)

# Charge le modèle spacy pour le français
nlp = spacy.load("fr_core_news_md")
# print("Modèle spacy chargé...")

# Initialise le dictionnaire qui va contenir les annotations
annotation = {}

# Liste des formes de négation et des adverbes de négation
list_neg = ["ne", "n'", "n"]
list_adv_neg = [
    "pas",
    "rien",
    "jamais",
    "point",
    "aucunement",
    "aucun",
    "aucunement",
    "nul",
    "nullement",
    "nulle",
    "plus",
]
list_neg_tok = list_neg + list_adv_neg

# Boucle sur les textes et les identifiants des locuteurs
n = 1
for t, i, ne in zip(texts, ids, nv_etudes):
    # Initialise la variable qui va contenir l'information sur la négation
    ↪ complète
    complete_neg = None

    # if n % 1000 == 0:
    #     break
    # elif n % 100 == 0:
    #     print(f"working on eslo {n}...")

    # Ajoute les informations sur le texte et initialisation des annotations
    annotation[f"eslo {n}"] = {}
    annotation[f"eslo {n}"][f"source"] = i.strip()
    annotation[f"eslo {n}"]["nv_etudes"] = ne.strip()
    annotation[f"eslo {n}"]["tokens"] = []
    annotation[f"eslo {n}"]["pos"] = []
```

```
    annotation[f"eslo {n}"]["neg_comp"] = []
    annotation[f"eslo {n}"]["y_absent"] = []
    annotation[f"eslo {n}"]["schwa_absent"] = []
    # print("\t<eslo>")

    # Vérifie si le texte contient une négation complète
    complete_negation_in_sent = None
    if
    ↪  regex.findall(r"\b(ne|n')\b.*\b({0})\b".format("|".join(list_adv_neg)),
    ↪  t):
        complete_negation_in_sent = True

    # Traite le texte avec spacy et boucle sur les tokens
    doc = nlp(t)
    for j, token in enumerate(doc):
        # Ignore les tokens vides
        if token.pos_ != "SPACE":
            # Ajoute le token et sa catégorie grammaticale au dictionnaire
            ↪  d'annotations
            annotation[f"eslo {n}"]["tokens"].append(token.text)
            # print(f'<w pos="{token.pos_}', end='')
            complete_pos = token.pos_
            if token.pos_ == "VERB":
                if len(token.morph.get("Tense")) > 0:
                    complete_pos = token.pos_ + ":" +
                    ↪  token.morph.get("Tense")[0]
                else:
                    complete_pos = token.pos_ + ":" + "Inf"
            annotation[f"eslo {n}"]["pos"].append(complete_pos)

            # Ajoute l'information sur la négation complète au dictionnaire
            ↪  d'annotations
            if complete_negation_in_sent:
                if token.text in list_neg:
                    position = len(annotation[f"eslo {n}"]["tokens"]) - 1
                    complete_neg = True

                if (
                    complete_neg
                    and token.text in list_adv_neg
                    and regex.match(
                        r"(^VERB.*|AUX)", annotation[f"eslo
                        ↪  {n}"]["pos"][position + 1]
                    )
                ):
                    annotation[f"eslo {n}"]["neg_comp"].append(True)
                else:
```

```
                    annotation[f"eslo {n}"]["neg_comp"].append(None)

            assert token.text == doc[j].text
            # Ajoute l'information sur l'absence de /y/ dans "tu" au dictionnaire d'annotations
            if (
                j < len(doc) - 1
                and token.dep_ == "nsubj"
                and regex.match(r"t\'", token.text)
                and regex.match(r"[aeiouy]", doc[j + 1].text)
            ):
                if token.text == "tu":
                    annotation[f"eslo {n}"]["y_absent"].append(False)
                    print(token, doc[j + 1].text, " : ypresent")
                else:
                    annotation[f"eslo {n}"]["y_absent"].append(True)
                    print(token, doc[j + 1].text, " : yabsent")

            # Ajoute l'information sur l'absence de schwa dans les clitiques au dictionnaire d'annotations
            elif (
                j < len(doc) - 1
                and regex.match(r"[cdjlmnqst]\'", token.text)
                and regex.match(r"^[^aeiouyàâéèêôhAEIOUYÉÀÂÊÔÈH]", doc[j + 1].text)
            ):
                annotation[f"eslo {n}"]["schwa_absent"].append(True)
            elif (
                j < len(doc) - 1
                and regex.match(r"^j[^aeiouyàâéèêôhAEIOUYÉÀÂÊÔÈH\']", token.text)
            ):
                annotation[f"eslo {n}"]["schwa_absent"].append(True)
            else:
                annotation[f"eslo {n}"]["y_absent"].append(None)
                annotation[f"eslo {n}"]["schwa_absent"].append(None)
    n += 1

print(annotation)

texts.close()
ids.close()

```

## A.3 Script pour l'extraction du corpus sms

Nom : `sms_extraction.py`

```python
from collections import defaultdict
from lxml import etree
from pprint import pprint
import regex
import os

## suppression des espaces multiples
def clean(text):
    text = regex.sub(r'\s+', ' ', text.strip(), flags=regex.MULTILINE)
    return text

## recuperation du root
def get_root():
    root = etree.parse('../data/brut/xml-SMS/cmr-88milsms-tei-v1.xml').getroot()
    return root




def get_person_niveauEtude():
    """
    Recupere le niveau d'education de chaque personne
    :return: dictionnaire {id : niveau_education}
    : exporte dans un fichier tsv
    """
    root = get_root()
    person_edu = {}
    for p, edu in zip(root.xpath('//*[name()="person"]'), root.xpath('//*[name()="education"]')):
        xml_id = p.get('{http://www.w3.org/XML/1998/namespace}id')
        education = edu.text
        person_edu[xml_id] = education

        # export to tsv file xml_id \t education
        # create filepath if not exists
        if not os.path.exists('../data/transformes/tsv/'):
            os.makedirs('../data/transformes/tsv/')

        ''' export to tsv file'''
        ## décommenter si on souhaite exporter à un fichier tsv
        with open('../data/transformes/tsv/SMS_id_education.tsv', 'a') as f:
            f.write(f"{xml_id}\t{education}\n")
```

```
42
43      return person_edu
44
45
46  def append_to_tsv(who, edu):
47      """
48      AJOUTE les locuteurs avec niveau d'études inconnus au fichier tsv
49      :param who: id de la personne
50      :param edu: niveau d'education
51      """
52      with open('../data/transformes/tsv/SMS_id_education.tsv', 'a') as f:
53          f.write(f"{who}\t{edu}\t\n")
54
55  def create_id_dialogues_txt(root, person_edu):
56      """
57      Creation des fichiers txt pour id et contenu en parallele
58      :param root: root du fichier xml
59      :param person_edu: dictionnaire {id : niveau_education}
60
61      :return: dictionnaire { who : [dialogues] }
62      : stockage dans des fichiers tsv
63
64      """
65      dialogues = defaultdict(lambda: defaultdict(list))
66      without_edu_info = []
67      #   recup { who : [dialogues] }
68      for post in root.xpath('//*[name()="post"]'):
69          who = post.get('who')[1:]
70          for p in post.xpath('.//following-sibling::*[name()="p"]'):
71
72              ## création fichier txt pour id et contenu
73              try:
74                  # create a txt file to store the id
75                  if len(clean(p.text)) != 0: # exporter seulement si non vides
76                      if not os.path.exists('../data/transformes/xml-SMS_id/'):
77                          os.makedirs('../data/transformes/xml-SMS_id/')
78                          with
                            ↪  open('../data/transformes/xml-SMS_id/SMS_ids.txt',
                            ↪  'a') as f:
79                              f.write(f"{who}\n")
80                      if not
                        ↪  os.path.exists('../data/transformes/xml-SMS_contenu/'):
81                          os.makedirs('../data/transformes/xml-SMS_contenu/')
82                          with
                            ↪  open('../data/transformes/xml-SMS_contenu/SMS_contenu.txt',
                            ↪  'a') as f:
83                              f.write(f"{clean(p.text)}\n")
```

```
84
85                  # creation du dictionnaire {id : dialogue}
86                  '''
87                  if who not in dialogues.keys():
88                      dialogues[who] = [clean(p.text)]
89                  else:
90                      dialogues[who].append(clean(p.text))
91                  '''
92              except Exception as e:
93                  print(f"An error occurred while updating dialogues for
                    ↪  '{who}': {e}")
94
95              ## creation des fichiers tsv pour id education et dialogues
96              try:
97                  if len(clean(p.text)) != 0: # exporter seulement si non vides
98                      with
                        ↪  open('../data/transformes/tsv/SMS_id_education_dialogues.tsv',
                        ↪  'a') as f:
99                          if who not in person_edu.keys():
100                             person_edu[who] = 'inconnu'
101                             append_to_tsv(who, person_edu[who])
102
                            ↪  f.write(f"{who}\t{person_edu[who]}\t{clean(p.text)}\n")
103
104             except Exception as e:
105                 # print(f"An error occurred while updating dialogues for
                    ↪  '{who}': {e}")
106                 without_edu_info.append(who)
107
108     return dialogues
109
110
111
112
113 def main():
114     root = get_root() # recuperation du root
115     person_edu = get_person_niveauEtude() # dictionnaire {id :
        ↪  niveau_education}
116     dialogues = create_id_dialogues_txt(root, person_edu) # creation des
        ↪  fichiers txt pour id et contenu
117
118
119 if __name__ == '__main__':
120     main()
121
```

## A.4 Script pour l'extraction du corpus ESLO

Nom : `extract_eslo.py`

```
# Ce script sert à extraire les ids et le contenu du corpus ESLO
import os
from xml.etree import ElementTree as ET

# Chemin du dossier contenant les fichiers XML
input_folder = '../data/xml-ESLO_nettoye/'
# Dossiers de sortie pour les fichiers d'ids et de contenus
output_folder_who = './data/xml-ESLO_id/'
output_folder_content = './data/xml-ESLO_contenu/'

# Vérifie si les dossiers de sortie existent, sinon les crée
os.makedirs(output_folder_who, exist_ok=True)
os.makedirs(output_folder_content, exist_ok=True)

# Parcours des fichiers dans le dossier d'entrée
for filename in os.listdir(input_folder):
    if filename.endswith('.xml'):
        # Chemin complet du fichier XML d'entrée
        input_file_path = os.path.join(input_folder, filename)
        tree = ET.parse(input_file_path)
        root = tree.getroot()

        # Chemins de sortie pour les fichiers who_values.txt et content_values.txt
        who_output_path = os.path.join(output_folder_who, filename.replace('.xml', '_who.txt'))
        content_output_path = os.path.join(output_folder_content, filename.replace('.xml', '_contenu.txt'))

        # Ouvre les fichiers de sortie
        with open(who_output_path, 'w') as who_file, open(content_output_path, 'w') as content_file:
            # Parcours des enfants de la racine
            for child in root:
                # Vérifie si l'élément est une balise <u>
                if child.tag == 'u':
                    # Récupère la valeur de l'attribut "who"
                    who_value = child.attrib.get('who', '')
                    # Récupère le contenu de la balise <u>, vérifie si c'est None avant d'appeler strip()
                    content_value = child.text.strip() if child.text is not None else ''
                    # Écrit les valeurs dans les fichiers de sortie
```

```
                    who_file.write(who_value + '\n')
                    content_file.write(content_value + '\n')

print("Traitement terminé.")

```

## A.5 Script pour la création d'un fichier niveau d'études pour ESLO

Nom : `add_etudes_eslo.py`

```
# Ce script a servi à faire un fichier correspondant aux niveaux d'études des locuteurs ESLO
import csv

# Fichiers d'entrée
id_file = '../data/transformes/xml-ESLO_id/concat_id.txt'
level_file = '../data/transformes/tsv/eslo_id_etudes.csv'
output_file = '../eslo_nvetudes.txt'

# Lecture du fichier des niveaux d'études
levels = {}
with open(level_file, mode='r', encoding='utf-8') as csvfile:
    csvreader = csv.reader(csvfile, delimiter=',')
    for row in csvreader:
        if len(row) >= 2:
            levels[row[0]] = row[1]  # ID est la première colonne, niveau d'études est la deuxième colonne

# Lecture du fichier des IDs et ajout des niveaux d'études
with open(id_file, mode='r', encoding='utf-8') as infile, open(output_file, mode='w', encoding='utf-8') as outfile:
    for line in infile:
        id_ = line.strip()
        level = levels.get(id_, 'Unknown')  # En cas d'ID non trouvé dans le fichier CSV
        outfile.write(f"{level}\n")

print("Traitement terminé. Vérifiez le fichier 'eslo_nvetudes.txt'.")
```

## A.6 Script pour transformer le fichier ESLO annoté en format XML

Nom : `eslo_to_xml.py`

```python
# Ce script sert à transformer le corpus ESLO annoté en format XML, avec de jolies balises.

import ast
import xml.etree.ElementTree as ET
import xml.dom.minidom

# Lire le contenu du fichier TXT
with open('../data/annote/eslo_annotation.txt', 'r', encoding='utf-8') as file:
    data = file.read()

# Convertir le contenu du fichier en dictionnaire Python
data_dict = ast.literal_eval(data)

# Créer l'élément racine
root = ET.Element('data')

# Traiter chaque entrée dans le dictionnaire
for key, value in data_dict.items():
    eslo_id = value.get('source', 'unknown')
    nv_etudes = value.get('nv_etudes', 'unknown')

    # Créer un élément 'eslo' avec les attributs 'id' et 'niveau'
    eslo_element = ET.SubElement(root, 'eslo', id=eslo_id, niveau=nv_etudes)

    # Ajouter les tokens et leurs pos en tant qu'éléments 'w'
    tokens = value.get('tokens', [])
    pos = value.get('pos', [])
    neg_comp = value.get('neg_comp', [])
    y_absent = value.get('y_absent', [])

    for i in range(len(tokens)):
        token = tokens[i]
        pos_tag = pos[i]

        # Créer l'élément 'w' avec l'attribut 'pos'
        w_element = ET.SubElement(eslo_element, 'w', pos=pos_tag)
        w_element.text = token

        # Ajouter les attributs neg_comp et y_absent si présents
        if i < len(neg_comp) and neg_comp[i] is not None:
            w_element.set('neg_comp', str(neg_comp[i]))
        if i < len(y_absent) and y_absent[i] is not None:
            w_element.set('y_absent', str(y_absent[i]))

xml_str = ET.tostring(root, encoding='unicode')
```

```
46
47 # Utiliser minidom pour prettifier le XML
48 dom = xml.dom.minidom.parseString(xml_str)
49 pretty_xml_str = dom.toprettyxml(indent="    ")
50
51 # Écrire le contenu XML dans un fichier avec une mise en page correcte
52 with open('../data/annote/eslo_annotation.xml', 'w', encoding='utf-8') as
   ↪  file:
53     file.write(pretty_xml_str)
54
55 print("Annotation XML terminée.")
56
```