Détails Scripts Perrine 2024

I. Récupération Automatique des pauses (#)

Dossier *Gold_Auto*

A. Script global (*detection_insertion_silences_conllu.ipynb*)

Ce script Python a pour objectif de traiter des fichiers de transcription, d'alignement et de phonétisation, en utilisant une série de notebooks Jupyter automatisés via la bibliothèque *papermill*. Chaque étape du traitement est encapsulée dans un notebook spécifique, avec des paramètres adaptés aux besoins des différentes tâches de traitement des données linguistiques. Voici une description détaillée de chaque étape :

1. Création du Tier Trans

La première étape consiste à créer un tier de transcription, indispensable pour l'utilisation de SPPAS (Software for the Phonetic and Phonological Analysis of Speech). Le script exécute le notebook *1-create_trans_tier.ipynb* en passant les paramètres nécessaires pour définir les répertoires d'entrée et de sortie.

- **Notebook** : 1-create_trans_tier.ipynb
- **Paramètres** :
    - *input_conllu_dir*: Répertoire des fichiers CONLL-U d'entrée.
    - *output_tg_dir*: Répertoire des fichiers TextGrid de sortie.
    - *input_wav_dir*: Répertoire des fichiers WAV d'entrée.

2. Utilisation de SPPAS

Cette étape utilise SPPAS pour générer des fichiers d'IPU, de tokenisation, d'alignement, de syllabification et de phonétisation. Le notebook *2-sppas_tg.ipynb* est exécuté avec les paramètres spécifiés.

- **Notebook** : 2-sppas_tg.ipynb
- **Paramètres** :
    - *input_tg_dir*: Répertoire des fichiers TextGrid d'entrée.
    - *input_pitchtier_dir*: Répertoire des fichiers PitchTier d'entrée.
    - *input_conllu_dir*: Répertoire des fichiers CoNLL-U d'entrée.
    - *ipu_script_sppas*: Indicateur pour utiliser le script IPU de SPPAS.
    - *ipu_script_webrtc*: Indicateur pour utiliser le script IPU de WebRTC.

3. Création du Tier Combined

Cette étape combine les tokens et les pauses en utilisant le notebook *3-get_combine_tier.ipynb*. Les paramètres définissent les répertoires d'entrée pour les fichiers TextGrid et PitchTier, et le répertoire de sortie pour les fichiers TSV.

- **Notebook** : 3-get_combine_tier.ipynb
- **Paramètres** :
    - *input_tg_dir*: Répertoire des fichiers TextGrid d'entrée.
    - *input_pitchtier_dir*: Répertoire des fichiers PitchTier d'entrée.
    - *output_tsv_dir*: Répertoire des fichiers TSV de sortie.

4. Reconstruction du Tier Trans à partir du Tier Combined

Cette étape consiste à reconstruire le tier de transcription à partir du tier combiné et à créer les tiers *Sent-Text* et *Sent-ID* nécessaires pour la correction des fichiers CoNLL-U. Le notebook *4-reconstruction_trans_merge.ipynb* est utilisé.

- **Notebook** : 4-reconstruction_trans_merge.ipynb
- **Paramètres** :
    - *input_conllu_dir*: Répertoire des fichiers CoNLL-U d'entrée.
    - *input_textgrid_dir*: Répertoire des fichiers TextGrid d'entrée.
    - *input_textgrid_dir_gold*: Répertoire des fichiers TextGrid "gold" d'entrée.
    - *output_merged_dir*: Répertoire des fichiers TextGrid fusionnésde sortie.

5. Correction des Fichiers CoNLL-U

Cette étape corrige les fichiers CoNLL-U en utilisant le notebook *5-update_conllu_pause.ipynb*, avec des paramètres définissant les répertoires d'entrée et de sortie pour les fichiers TextGrid et CoNLL-U.

- **Notebook** : 5-update_conllu_pause.ipynb
- **Paramètres** :
    - *input_textgrid_directory*: Répertoire des fichiers TextGrid d'entrée.
    - *input_conllu_directory*: Répertoire des fichiers CONLL-U d'entrée.
    - *conllu_output_directory*: Répertoire des fichiers CONLL-U de sortie.

6. Création des TextGrid SLAM

La dernière étape consiste à créer des fichiers TextGrid pour le projet SLAM en exécutant le notebook 6-create_slam_tg.ipynb. Les paramètres définissent les répertoires d'entrée pour les fichiers CONLL-U et TextGrid, et le répertoire de sortie pour les nouveaux fichiers TextGrid.

- **Notebook** : 6-create_slam_tg.ipynb
- **Paramètres** :
    - *input_conllu_dir*: Répertoire des fichiers CONLL-U d'entrée.
    - *input_textgrid_dir*: Répertoire des fichiers TextGrid d'entrée.
    - *output_tg_dir*: Répertoire des fichiers TextGrid de sortie.

Conclusion

Ce script permet d'automatiser le processus complexe de traitement des données linguistiques en utilisant des notebooks Jupyter et la bibliothèque papermill. Chaque étape est paramétrée pour garantir la cohérence et l'efficacité du flux de travail, depuis la création initiale des tiers de transcription jusqu'à la correction finale des fichiers CONLL-U et la génération des fichiers TextGrid SLAM.

B. Détails des scripts

1. *1-create_trans_tier.ipynb*

Ce script vise à convertir des fichiers CoNLL-U en fichiers TextGrid tout en prétraitant les données textuelles et en copiant les fichiers audio correspondants. Voici une explication détaillée de chaque étape :

1. Importation des modules

- **Importation des bibliothèques nécessaires** : Les modules os, re, shutil, subprocess, et sys sont importés pour les opérations sur les fichiers et le système.
- **Détermination des chemins** : Le chemin actuel (current_path) est récupéré, et le chemin vers le dossier 'scripts' est construit (scripts_path).
- **Ajout du chemin 'scripts' au sys.path** : Pour pouvoir importer des modules du dossier 'scripts', son chemin est ajouté à sys.path.
- **Importation des modules spécifiques** : Les fonctions et classes nécessaires de outils.conll3 et praatio.textgrid sont importées.

2. Fonction `convert_misc_to_dict`

La fonction `convert_misc_to_dict` transforme des informations diverses contenues dans une chaîne de caractères en un dictionnaire Python. Elle prend en paramètre une chaîne de caractères `misc` qui contient des données sous forme de paires clé=valeur, séparées par des barres verticales ('|'). La fonction commence par initialiser un dictionnaire vide `result_dict` destiné à stocker les paires clé-valeur extraites. Ensuite, elle divise la chaîne `misc` en utilisant le séparateur '|', ce qui produit une liste de paires. Si cette liste ne contient pas uniquement le caractère de soulignement '_', ce qui indiquerait l'absence de données utiles, la fonction traite chaque paire individuellement. Pour chaque paire, elle sépare la clé et la valeur en utilisant le signe égal '=' comme délimiteur, puis ajoute ces paires au dictionnaire `result_dict`. En fin de compte, la fonction retourne ce dictionnaire qui contient toutes les paires clé-valeur extraites de la chaîne d'origine.

3. Fonction `create_textgrid`

La fonction `create_textgrid` est conçue pour créer un fichier TextGrid à partir d'un fichier CoNLL-U, souvent utilisé pour représenter des arbres syntaxiques et des annotations de dépendance pour les phrases. La fonction prend deux paramètres : `file_path`, qui est le chemin vers le fichier CoNLL-U, et `output_textgrid_path`, qui est le chemin où le fichier TextGrid généré sera sauvegardé.

La fonction commence par initialiser un objet TextGrid à l'aide de la bibliothèque `textgrid` et crée une liste vide `entryList`. Elle extrait ensuite les arbres syntaxiques et leurs métadonnées du fichier CoNLL-U en utilisant la fonction `conllFile2trees`. Une variable `last_end_time` est initialisée à zéro pour suivre la fin de la dernière annotation temporelle, et une liste vide `intervals` est créée pour stocker les intervalles d'annotations.

Ensuite, la fonction itère sur chaque arbre (`tree`) dans la liste d'arbres extraits (`trees`). Pour chaque arbre, elle convertit l'arbre en une chaîne de caractères et recherche une correspondance pour le texte original de la phrase à l'aide d'une expression régulière. Si une correspondance est trouvée, elle divise la chaîne de texte en une liste de mots (`text_list`). Sinon, elle passe à l'arbre suivant.

La fonction extrait ensuite les informations de synchronisation et autres métadonnées de chaque token dans l'arbre en les stockant dans une liste `misc_list`. Les mots de l'arbre sont extraits et stockés dans la liste `words`.

Pour chaque mot, la fonction initialise des variables pour stocker le texte actuel (`current_text`) et les bornes temporelles (`current_xmin` et `current_xmax`). Elle vérifie ensuite si le premier mot de la première phrase a une annotation de début (`AlignBegin`). Si oui, elle met à jour `current_xmax` et ajoute un intervalle annoté pour la période allant de 0 à `current_xmax`.

La fonction continue en construisant les phrases à partir des mots, en excluant les signes de ponctuation. Elle met à jour les bornes temporelles et le texte actuel pour chaque phrase, en vérifiant les annotations de début et de fin pour les tokens dans les métadonnées. Les intervalles annotés sont ajoutés à la liste `intervals`.

Enfin, la fonction crée une nouvelle tierce d'intervalles (`IntervalTier`) avec les intervalles annotés, l'ajoute à l'objet TextGrid, et sauvegarde le TextGrid généré à l'emplacement spécifié par `output_textgrid_path`.

Cette fonction est donc un outil puissant pour convertir des fichiers de format CoNLL-U en fichiers TextGrid, facilitant ainsi l'analyse et la visualisation des annotations linguistiques temporelles.

4. Fonction `preprocess_text_list`

La fonction `preprocess_text_list` prend en entrée une liste de mots (`text_list`) et le nom d'un fichier (`file_name`). Elle remplace les contractions courantes dans la liste de mots par leurs formes développées. Les contractions sont des formes abrégées où deux mots sont combinés en un seul, souvent avec une apostrophe, comme "don't" pour "do not" ou "I'm" pour "I am". La fonction commence par créer une nouvelle liste vide `new_text_list` pour stocker les mots après le prétraitement. Ensuite, elle définit un dictionnaire `contractions` où chaque clé est une contraction et chaque valeur est une liste de mots représentant la forme développée de la contraction.

La fonction inclut des conditions pour adapter le traitement en fonction de `file_name`. Par exemple, pour certains fichiers comme "WAZP_03", "ENU_01", et "ONI_07", la contraction "it's" est incluse dans le dictionnaire des contractions. Pour d'autres fichiers comme "WAZA_02" et "PRT_11", la contraction "cannot" est ajoutée. Pour le fichier "KAD_10", "dat's" est également ajouté.

La fonction parcourt ensuite chaque mot dans `text_list`. Si le mot en minuscule est trouvé dans le dictionnaire des contractions, la forme développée de la contraction est ajoutée à `new_text_list`. Sinon, le mot original est ajouté à `new_text_list`. Finalement, la fonction renvoie la liste `new_text_list` avec toutes les contractions remplacées par leurs formes développées, normalisant ainsi le texte pour une analyse ultérieure.

5. Fonction `create_textgrid_taln`

Cette fonction est similaire à `create_textgrid`, mais elle effectue des prétraitements spécifiques sur les mots du fichier CoNLL-U, tels que la suppression des # et l'expansion des contractions. Cependant si on a déjà les fichiers CoNLL-U sans l'annotation des #, on peut directement utiliser la fonction `create_textgrid`. Le script `change_gold_in_non_gold.py` se trouvant dans le dossier outils, permet l'obtention de ces CoNLL-U.

La fonction `create_textgrid_taln` a pour but de créer un fichier TextGrid à partir d'un fichier formaté en CoNLL-U. Le TextGrid est un format de fichier utilisé en linguistique pour annoter des enregistrements audio avec des informations textuelles temporelles. Cette fonction prend en entrée le chemin d'un fichier CoNLL-U (`file_path`) et le chemin de sortie où le fichier TextGrid sera sauvegardé (`output_textgrid_path`). La fonction commence par créer un objet Textgrid vide. Ensuite, elle utilise la fonction `conllFile2trees` pour convertir le fichier CoNLL-U en une structure d'arbres syntaxiques.

La fonction traite chaque arbre syntaxique un par un. Pour chaque arbre, elle extrait la ligne de texte correspondant à l'arbre en utilisant une expression régulière. Cette ligne de texte est ensuite prétraitée avec la fonction `preprocess_text_list` pour gérer les contractions. La fonction construit ensuite une liste `misc_list` qui contient des informations supplémentaires sur chaque mot de l'arbre, telles que les alignements temporels. Ces informations sont converties en dictionnaires à l'aide de la fonction `convert_misc_to_dict`.

La fonction parcourt ensuite chaque mot de l'arbre. Si c'est le premier mot du premier arbre, elle enregistre l'alignement temporel de fin et l'ajoute à une liste d'intervalles `intervals`. Elle construit également une phrase en ajoutant chaque mot à une liste `sentence`. Lorsqu'elle atteint la fin de la phrase, elle supprime les signes de ponctuation de la phrase et crée une chaîne de texte `current_text`.

Pour chaque mot dans la liste de texte, la fonction vérifie l'alignement temporel de début et de fin à partir de `misc_list` et les convertit en secondes. Si ces alignements sont valides, elle les ajoute à la liste `intervals`. La fonction gère également les cas où une phrase contient un seul mot, ou où il y a une correspondance exacte entre le texte et la phrase.

Enfin, la fonction crée une `IntervalTier` (couche d'intervalle) dans l'objet Textgrid avec les intervalles collectés et sauvegarde l'objet Textgrid au chemin de sortie spécifié.

6. Préparation des paramètres et traitement des fichiers

- **Définition des chemins** : Les répertoires d'entrée et de sortie pour les fichiers CoNLL-U et WAV sont définis.
- **Création des répertoires de sortie** : Si le répertoire de sortie pour les fichiers TextGrid n'existe pas, il est créé.
- **Boucle sur les fichiers CoNLL-U** :
    - Cette section de code parcourt un répertoire contenant des fichiers CoNLL-U et génère des fichiers TextGrid correspondants tout en organisant les fichiers WAV associés dans des dossiers appropriés. La première tâche de ce script est de vérifier chaque fichier dans le répertoire `input_conllu_dir` pour voir s'il se termine par "*MG.conllu*" (fichier gold) ou "*M.conllu*" (fichier non gold). Si c'est le cas, il crée un chemin complet vers ce fichier.

      Ensuite, le script détermine le nom du dossier de sortie basé sur le nom du fichier sans extension. Si le nom de fichier commence par "ABJ", il prend les trois premières parties du nom, sinon, il prend les deux premières parties. Il vérifie ensuite si ce dossier de sortie existe, et le crée si nécessaire.

      Le nom du fichier TextGrid de sortie est généré à partir du nom de fichier sans extension et placé dans le dossier approprié. Si le dossier contenant le fichier TextGrid de sortie n'existe pas, il est créé.

      Le script appelle ensuite la fonction `create_textgrid` pour chaque fichier se terminant par "*M.conllu*" et la fonction `create_textgrid_taln` pour ceux se terminant par "*MG.conllu*", tout en excluant certains fichiers spécifiques basés sur leur nom. Si le fichier CoNLL-U se termine par "*M.conllu*", le script appelle la fonction `create_textgrid` pour générer le fichier TextGrid. Ensuite, il copie le fichier WAV correspondant du répertoire `input_wav_dir` vers le nouveau dossier créé.

      Pour les fichiers se terminant par "*_MG.conllu*", le script appelle la fonction `create_textgrid_taln`, qui est une variante spécifique pour traiter ces fichiers. Ensuite, il copie également le fichier WAV correspondant dans le nouveau dossier, sauf si le fichier destination existe déjà.

Ce script convertit donc des fichiers CoNLL-U en fichiers TextGrid tout en traitant les métadonnées et les alignements temporels nécessaires pour une synchronisation correcte des données textuelles avec les fichiers audio.

2. *2-sppas_tg.ipynb*

1. Importation des modules

- **Importation des bibliothèques nécessaires** : Les modules *sys*, *os*, *subprocess* sont importés pour les opérations sur les fichiers et le système.
- **Détermination du chemin de base** : Le chemin actuel (base_path) est récupéré.
- **Ajout du chemin `scripts/outils` au sys.path** : Pour pouvoir importer des modules du dossier `scripts/outils`, son chemin est ajouté à *sys.path*.
- **Importation des modules spécifiques** : Les fonctions et classes nécessaires de *modif_ipus_tier*, *webrtcvad_ipu*, *get_index* et *tqdm* sont importées.

2. Définition des paramètres

- **Définition des chemins d'entrée** : Les répertoires d'entrée pour les fichiers TextGrid (*input_tg_dir*), PitchTier (*input_pitchtier_dir*) et CoNLL-U (*input_conllu_dir*).
- **Choix du script à exécuter pour les IPU**: Les variables booléennes déterminent si le script SPPAS (*ipu_script_sppas*) ou WebRTC (*ipu_script_webrtc*) sera utilisé pour les IPU.

3. Traitement des fichiers

Ce script parcourt un répertoire donné `input_tg_dir` contenant des sous-répertoires de fichiers audio `.wav` et de fichiers TextGrid `.TextGrid`. Pour chaque sous-répertoire, le script identifie les fichiers audio et leurs TextGrid correspondants, puis exécute plusieurs commandes et scripts pour traiter ces fichiers. Voici une explication détaillée du script :

Le script commence par itérer sur chaque sous-répertoire dans `input_tg_dir` en utilisant `os.listdir`. Pour chaque sous-répertoire, il construit le chemin complet `subdir_path` et vérifie s'il s'agit d'un répertoire avec `os.path.isdir`.

Ensuite, le script initialise deux listes `wav_files` et `textgrid_files` pour stocker les noms des fichiers `.wav` et `.TextGrid` respectivement. Il parcourt tous les fichiers du sous-répertoire et ajoute les fichiers `.wav` à `wav_files` et les fichiers `.TextGrid` correspondant aux annotations `_M` ou `_MG` à `textgrid_files`.

Pour chaque fichier `.wav` dans `wav_files`, le script construit le nom du fichier TextGrid correspondant en remplaçant l'extension `.wav` par `.TextGrid`. Si ce fichier TextGrid existe dans `textgrid_files`, le script procède au traitement.

Le script commence par construire les chemins complets des fichiers `.wav` et `.TextGrid` en utilisant `os.path.join`.

Ensuite, selon que `ipu_script_sppas` ou `ipu_script_webrtc` est défini, le script exécute une commande pour obtenir les unités prosodiques inter-pause (IPU). Si `ipu_script_sppas` est défini, le script utilise `subprocess.run` pour exécuter un script SPPAS avec des paramètres spécifiques. Sinon, il appelle une fonction `process_file` avec le fichier TextGrid et le fichier `.wav`.

Le script construit ensuite le chemin du fichier IPU et le chemin du fichier TextGrid à partir des noms de fichiers `.wav` correspondants. Il construit également le chemin du fichier PitchTier à partir du répertoire `input_pitchtier_dir`.

Ensuite, le script appelle une fonction `correct_silence_duration` pour modifier la tier IPU, corriger la durée des silences et supprimer les silences inférieurs à 0,1 seconde.

Après cela, le script exécute une autre commande pour obtenir les tokens, syllabes, transcriptions phonétiques et alignements en utilisant `subprocess.run` pour exécuter un script d'annotation SPPAS avec des paramètres spécifiques.

Enfin, le script appelle une fonction `get_index_function` avec le chemin du fichier CoNLL-U et le chemin du fichier palign TextGrid pour obtenir les indices de tokens.

Le script continue ce processus pour chaque fichier dans chaque sous-répertoire et affiche "Fin de l'exécution" une fois terminé.

En résumé, ce script permet de traiter des fichiers .wav et .TextGrid pour obtenir des informations telles que les IPU, les tokens, les syllabes, les transcriptions phonétiques et les alignements en utilisant principalement SPPAS

3. *3-get_combine_tier.ipynb*

1. Importation des modules

- **Importation des bibliothèques nécessaires** : Les modules *os*, *time*, *praatio.textgrid* et *tqdm* sont importés pour les opérations sur les fichiers, la gestion du temps, la manipulation des fichiers TextGrid et l'affichage des barres de progression.
- **Définition d'une variable epsilon (eps)**: Cette variable est utilisée pour définir une tolérance dans les calculs de temps. Notamment si un espace vide est inclus dans un interval IPU et qu'il est supérieur à la valeur eps donnée, alors on place un #.

2. Fonction `align_silence`

Cette fonction aligne les intervalles de silence dans le tier '*Combined*' du fichier TextGrid '*tokens_path*'. Elle trouve les intervalles de silence les plus proches et les remplace dans le tier '*Combined*', en éliminant les doublons et les chevauchements.

La fonction `align_silence` prend deux chemins de fichiers en entrée : `ipus_path` et `tokens_path`, qui pointent respectivement vers des fichiers TextGrid contenant des intervalles de silence et des intervalles de tokens. Cette fonction ajuste les intervalles de silence dans la tier "Combined" du fichier TextGrid des tokens pour aligner correctement les silences en fonction des intervalles de silence présents dans le fichier TextGrid des IPUs. La fonction commence par ouvrir les fichiers TextGrid spécifiés par `ipus_path` et `tokens_path` en utilisant `tgio.openTextgrid`. Les objets TextGrid résultants sont stockés dans `textgrid_ipus` et `textgrid_tokens`. Ensuite, elle récupère les tiers "IPUs" et "Combined" des objets TextGrid. `ipus_tier` contient les intervalles de silence, tandis que `combined_tier` contient les intervalles de tokens. Une copie des entrées du tier "Combined" est faite dans une nouvelle liste `new_combined_entries` pour la modification.

La boucle itère ensuite sur chaque intervalle dans `new_combined_entries`. Si l'intervalle a un label de "#", cela indique un silence, et la fonction cherche l'intervalle de silence le plus proche dans le tier "IPUs". Cette recherche est réalisée en calculant la distance absolue entre le début de l'intervalle "Combined" et chaque intervalle "IPUs" ayant le label "#". L'intervalle de silence le plus proche est trouvé en utilisant la fonction `min`. Un nouvel intervalle est créé avec les temps de début et de fin de l'intervalle de silence trouvé, et cet intervalle remplace l'intervalle original dans la liste `new_combined_entries`.

Pour éviter les doublons et les chevauchements d'intervalles, une nouvelle liste `unique_combined_entries` est créée. La fonction vérifie si un intervalle (début et fin) a déjà été vu en utilisant un ensemble `seen_intervals`. Si un intervalle est unique, il est ajouté à la liste `unique_combined_entries` et à l'ensemble `seen_intervals`. Finalement, les entrées du tier "Combined" sont mises à jour avec les nouvelles entrées dans `unique_combined_entries`. La fonction remplace ensuite le tier "Combined" dans `textgrid_tokens` avec la version mise à jour et retourne l'objet TextGrid modifié.

3. Fonction `detect_silence_in_sentence`

La fonction `detect_silence_in_sentence` détecte les intervalles de silence dans les phrases d'un fichier TextGrid spécifié par `sent_path` et les associe aux indices provenant d'un autre fichier TextGrid spécifié par `id_path`. Elle prend également un troisième fichier TextGrid, `syl_tok_path`, contenant les intervalles de syllabes et de tokens combiné aux silences. Cette fonction retourne une liste de tuples contenant les informations des phrases avec des intervalles de silence, et peut également écrire ces informations dans un fichier TSV optionnel spécifié par `output_tsv`.

La fonction commence par extraire le nom de base du fichier de phrases (`sent_path`) sans son extension et ouvre les trois fichiers TextGrid (`id_path`, `sent_path`, `syl_tok_path`) en utilisant la méthode `tgio.openTextgrid`. Ensuite, elle initialise une liste vide `phrases_with_hash` pour stocker les phrases contenant des intervalles de silence. La fonction parcourt les entrées du tier "*Combined*" du fichier des syllabes et tokens pour détecter les intervalles marqués par un "#", indiquant un silence. Pour chaque intervalle de silence détecté, elle récupère les temps de début (`xmin`) et de fin (`xmax`) et cherche l'intervalle de phrase correspondant dans le tier "*trans*" du fichier de phrases, c'est-à-dire un intervalle dont le temps de début est inférieur ou égal à `xmin` et le temps de fin est supérieur ou égal à `xmax`.

Ensuite, la fonction cherche les labels des indices précédents et suivants par rapport à l'intervalle de silence en parcourant en sens inverse et en sens normal les entrées du tier "*index*" du fichier des indices. Si aucun label précédent n'est trouvé, une anomalie est signalée. Si le label suivant n'est pas trouvé, le dernier label du tier "*index*" est pris comme valeur par défaut. Une fois les labels précédents et suivants trouvés, ainsi que le label de l'intervalle de phrase correspondant, la fonction ajoute ces informations sous forme de tuple dans la liste `phrases_with_hash`.

Enfin, bien que le code pour écrire les informations dans un fichier TSV soit présent mais commenté, la fonction retourne la liste `phrases_with_hash` contenant les tuples d'informations sur les phrases avec des intervalles de silence.

4. Fonction `create_tier`

La fonction `create_tier` est conçue pour créer un intervalle de type "*Combined*" à partir des intervalles "*IPUs*" et "*Tokens*" des fichiers TextGrid. Cette fonction prend trois paramètres: `ipus_path` qui est le chemin vers le fichier TextGrid contenant les intervalles de silence, `tokens_path` qui est le chemin vers le fichier TextGrid contenant les intervalles de tokens, et `tier` qui est le nom du tier à créer. La fonction renvoie un objet `IntervalTier` contenant les intervalles combinés.

La fonction commence par ouvrir les fichiers TextGrid spécifiés par `ipus_path` et `tokens_path`, et initialise un nouvel objet TextGrid pour stocker les intervalles combinés. Elle obtient les tiers des intervalles de silence ("*IPUs*") et des tokens à partir des fichiers TextGrid. Les intervalles de ces tiers sont extraits dans des listes `ipus_intervals` et `tokens_intervals` respectivement. Ensuite, quelques variables sont initialisées pour suivre les positions actuelles des intervalles de silence et de tokens, ainsi que pour gérer les situations spécifiques lors de la combinaison des intervalles.

La boucle principale traite chaque intervalle de token et de silence, en ajustant les intervalles en fonction des conditions spécifiques définies par les étiquettes et les positions des intervalles. Divers cas sont gérés, tels que l'insertion d'intervalles de silence lorsqu'un intervalle de token chevauche un intervalle de silence ou lorsque des espaces sont trouvés entre les intervalles. La logique inclut également la gestion des intervalles qui se chevauchent et la suppression des doublons pour assurer qu'il n'y a pas de chevauchement dans les intervalles combinés.

Enfin, après avoir traité tous les intervalles, la fonction vérifie si le premier intervalle commence à 0.0 et l'ajoute si nécessaire. Les intervalles finaux sont ensuite utilisés pour créer un nouvel intervalle de type "*Combined*" ou "*SyllSil*" en fonction du paramètre `tier`. Cette nouvelle tier est ensuite renvoyée par la fonction.

5. Fonction `save_textgrid`

La fonction `save_textgrid` est utilisée pour sauvegarder un fichier TextGrid modifié à un emplacement spécifié. Elle prend trois paramètres : `token_syl_tier`, qui est le tier contenant les intervalles de tokens ou de syllabes ; `syllsil_tier`, qui est le tier contenant les intervalles de silence ; et `output_path`, qui est le chemin où le fichier TextGrid doit être sauvegardé.

Dans cette fonction, un nouvel objet `Textgrid` est d'abord créé. Ensuite, les tiers fournis (`token_syl_tier` et `syllsil_tier`) sont ajoutés à cet objet TextGrid. Après avoir ajouté les tiers, le fichier TextGrid est sauvegardé à l'emplacement spécifié par `output_path`. Le fichier est enregistré au format "*long_textgrid*" et inclut les espaces vides.

Cette fonction est essentielle pour persister les modifications effectuées sur les tiers d'intervalles dans un fichier TextGrid, permettant ainsi de conserver et d'analyser les données de manière structurée.

6. Définition des paramètres et traitement des fichiers

- **Définition des chemins d'entrée et de sortie** : Les répertoires d'entrée pour les fichiers TextGrid (*input_tg_dir*) et PitchTier (*input_pitchtier_dir*), et le répertoire de sortie pour les fichiers TSV (*output_tsv_dir*) sont définis.
- **Création du répertoire de sortie** : Si le répertoire de sortie pour les fichiers TSV n'existe pas, il est créé.

7. Traitement des fichiers

La boucle principale commence par initialiser une liste vide `all_phrases_with_hash` pour stocker les informations sur les phrases contenant des intervalles de silence. Ensuite, pour chaque sous-répertoire de `input_tg_dir`, elle vérifie si le chemin est bien un répertoire. Si c'est le cas, trois listes vides sont initialisées pour stocker les noms des fichiers TextGrid contenant respectivement les intervalles d'IPU, d'identifiants et de phrases.

La boucle interne parcourt tous les fichiers du sous-répertoire courant et ajoute les fichiers appropriés aux listes correspondantes en fonction de leurs extensions de nom de fichier.

Après avoir remplis ces listes, une autre boucle interne traite chaque fichier IPU. Elle construit les chemins des fichiers TextGrid correspondants pour les identifiants, les phrases et les syllabes. Elle construit également le chemin du fichier PitchTier associé.

Si le fichier PitchTier existe, la fonction vérifie ensuite si les fichiers d'identifiants et de phrases associés existent également. Si ces fichiers existent, elle construit leurs chemins et imprime le chemin du fichier d'identifiants pour confirmation.

Ensuite, les fonctions `create_tier` et `save_textgrid` sont appelées pour créer les tiers combinés et syllabes alignés avec les silences, et pour les sauvegarder dans un nouveau fichier TextGrid. La fonction `align_silence` est ensuite appelée pour aligner les intervalles de silence, et `detect_silence_in_sentence` pour détecter les intervalles de silence dans les phrases et les ajouter à la liste `all_phrases_with_hash`.

Enfin, on écriture toutes les phrases contenant des silences dans un fichier TSV, cette partie du code est cependant commentée.

Ce script permet de combiner les intervalles de silence et de tokens/syllabes dans les fichiers TextGrid, en créant un nouveau tier '*Combined*' et en détectant les silences dans les phrases.

4. *4-reconstruction_trans_merge.ipynb*

Ce script combine les tokens du tiers "*Combined*" pour reformer le tier "*trans*" dans les fichiers TextGrid. Il effectue diverses tâches, notamment la division de longues phrases, la création de nouveaux tiers, la tokenisation et la syllabification, la fusion des fichiers TextGrid gold et non gold, et l'extraction et l'alignement des phrases et tokens.

1. Initialisation et importation des modules

- **Importation des bibliothèques** : Le script commence par importer les bibliothèques nécessaires pour manipuler les fichiers TextGrid, gérer les systèmes de fichiers, et exécuter des commandes externes.
- **Configuration des chemins** : Détermine le chemin actuel de travail et ajoute dynamiquement le chemin du dossier scripts à sys.path pour permettre l'importation de modules personnalisés.

2. Fonction `split_long_sentence`

La fonction `split_long_sentence` permet de diviser une phrase en plusieurs phrases de longueur maximale définie par `max_length`, facilitant ainsi la tokenisation et d'autres analyses par SPPAS. Elle prend une phrase et une longueur maximale comme paramètres et renvoie une liste de phrases divisées. Pour ce faire, elle parcourt les mots de la phrase d'origine, en construisant des phrases plus courtes jusqu'à ce que la longueur maximale soit atteinte, puis elle ajoute ces phrases courtes à la liste de résultats.

3. Fonction `create_new_trans`

La fonction `create_new_trans` utilise `split_long_sentence` pour reformer le tier `trans` à partir des tokens du tier `Combined` dans un fichier TextGrid. Elle prend le chemin du fichier TextGrid d'origine et le nom du nouveau fichier TextGrid à créer. Elle ouvre le fichier TextGrid d'origine et extrait les intervalles du tier `Combined`. Ensuite, elle parcourt ces intervalles, combinant les tokens en phrases tout en veillant à ce qu'aucune phrase ne dépasse 150 caractères et à ce que les intervalles soient corrigés en conséquence.

Pour chaque intervalle de `Combined`, si le label n'est pas `#`, elle ajoute le token à la phrase en cours de construction. Si un label `#` est rencontré, cela signifie la fin d'une phrase, alors elle divise cette phrase en phrases plus courtes si nécessaire et ajuste les intervalles temporels en conséquence. Les nouvelles phrases et leurs intervalles sont ajoutés à la liste des phrases reformées.

Ensuite, elle crée un nouvel objet TextGrid et y ajoute un nouveau tier `trans` contenant les phrases reformées et leurs intervalles corrigés. Enfin, le nouveau TextGrid est sauvegardé dans un fichier spécifié par le paramètre `new`.

4. Fonction `generate_tiers_selection_gold_non_gold_silence`

La fonction `generate_tiers_selection_gold_non_gold_silence` a pour but de générer une sélection de niveaux (tiers) à partir de fichiers TextGrid gold et non-gold (ou gold automatique) situés dans des répertoires spécifiques. Cette fonction permet de trier et de sélectionner les niveaux pertinents pour une analyse ultérieure. Elle prend deux paramètres : `directory`, qui est le chemin vers le répertoire contenant les fichiers TextGrid non-gold, et `directory_gold`, qui est le chemin vers le répertoire contenant les fichiers TextGrid gold.

La fonction commence par initialiser deux dictionnaires vides, `tiers_gold` et `tiers_non_gold`, pour stocker les niveaux des fichiers gold et non-gold respectivement. Elle parcourt ensuite tous les fichiers du répertoire spécifié par `directory`. Pour chaque fichier TextGrid non-gold, elle vérifie s'il se termine par "*MG-syll.TextGrid*" ou "*MG-palign.TextGrid*" et ajoute les niveaux correspondants ("*SyllAlign*" pour les syllabes alignées et "*TokensAlign*" pour les tokens alignés) au dictionnaire `tiers_non_gold`. Pour les autres fichiers TextGrid non-gold, qui ne contiennent pas certains sous-chaînes spécifiques indiquant des fichiers de phonétique, de tokens, fusionnés, etc., elle ajoute le niveau "*trans*" (transcriptions).

De même, la fonction parcourt tous les fichiers du répertoire spécifié par `directory_gold`. Pour chaque fichier TextGrid gold, elle vérifie s'il se termine par "*MG-id.TextGrid*" et ajoute le niveau "*index*" au dictionnaire `tiers_gold`. Pour les autres fichiers TextGrid gold, qui ne contiennent pas les mêmes sous-chaînes spécifiques que pour les fichiers non-gold, elle ajoute également le niveau "*trans*".

Enfin, la fonction retourne un tuple contenant les deux dictionnaires, `tiers_gold` et `tiers_non_gold`, qui regroupent les niveaux pertinents pour les fichiers gold et non-gold respectivement.

5. Fonction `merge_gold_non_gold`

La fonction `merge_gold_non_gold` a pour objectif de fusionner différents niveaux (tiers) provenant de fichiers TextGrid d'or (gold) et de non-or (non-gold) en un seul fichier TextGrid. Cette fonction prend plusieurs paramètres : `directory` (le chemin vers le répertoire contenant les fichiers TextGrid non-gold), `directory_gold` (le chemin vers le répertoire contenant les fichiers TextGrid gold), `tiers_gold` (un dictionnaire des niveaux obtenus à partir des fichiers gold), `tiers_non_gold` (un dictionnaire des niveaux obtenus à partir des fichiers non-gold), `base_name_file` (le nom du fichier de sortie), et `merged` (le dossier de destination pour le fichier fusionné, optionnel).

La fonction commence par créer un nouvel objet TextGrid vide nommé `merged_textgrid`. Elle définit ensuite un ordre de priorité pour les niveaux à ajouter dans le fichier fusionné : "*trans*", "*index*", "*trans*", "*TokensAlign*", et "*SyllAlign*". Un ensemble `added_tiers` est utilisé pour suivre les niveaux qui ont déjà été ajoutés afin d'éviter les doublons.

Dans un premier temps, la fonction ajoute le premier niveau "*trans*" provenant des fichiers gold. Pour chaque fichier gold, elle ouvre le fichier TextGrid, vérifie si le niveau "*trans*" existe et l'ajoute au fichier fusionné si ce n'est pas déjà fait.

Ensuite, la fonction ajoute les autres niveaux provenant des fichiers non-gold, toujours en suivant l'ordre de priorité défini. Pour chaque fichier non-gold, elle ouvre le fichier TextGrid, vérifie si le niveau spécifié existe, et l'ajoute au fichier fusionné s'il n'a pas encore été ajouté.

Après avoir ajouté les autres niveaux, la fonction ajoute le deuxième niveau "*trans*" provenant des fichiers non-gold, mais en le renommant "*transAuto*" pour éviter les conflits avec le premier niveau "*trans*" provenant des fichiers gold.

Enfin, la fonction enregistre le fichier TextGrid fusionné dans le répertoire spécifié par `merged` s'il est fourni, sinon elle l'enregistre dans le répertoire des fichiers non-gold. Le fichier fusionné est enregistré au format "*long_textgrid*" et inclut les espaces vides pour garantir une bonne lecture et interprétation des niveaux.

Cette approche permet de combiner efficacement et de manière organisée les niveaux pertinents provenant de sources différentes en un seul fichier, facilitant ainsi les analyses et les traitements ultérieurs.

6. Fonction `sppas_tokenisation_syllabification`

Cette fonction exécute un script SPPAS pour effectuer la tokenisation et la syllabification des intervalles dans un fichier TextGrid. Elle utilise une commande shell pour appeler SPPAS avec les paramètres nécessaires, y compris les fichiers TextGrid et WAV correspondants.

7. Fonction `correct_tokenisation`

La fonction `correct_tokenisation` a pour objectif de corriger les labels des tokens dans un fichier TextGrid en remplissant les labels vides avec un astérisque (*) et en comparant ces intervalles avec ceux de la tier `transAuto` pour mettre à jour les tokens si nécessaire. La fonction prend en paramètre `textgrid_file`, qui est le chemin vers le fichier TextGrid à corriger.

Tout d'abord, la fonction ouvre le fichier TextGrid en utilisant `tgio.openTextgrid` et obtient les tiers `TokensAlign` et `transAuto`. Ensuite, elle parcourt les entrées du tier `TokensAlign` et remplace les labels vides par des astérisques (*), en veillant à arrondir les valeurs de début et de fin des intervalles à quatre décimales pour garantir la précision.

Après cette première correction, la fonction filtre les intervalles valides en excluant ceux où l'heure de début est supérieure ou égale à l'heure de fin. Les entrées valides sont ensuite mises à jour dans le tier `TokensAlign` à l'aide de la méthode `replaceTier`.

Ensuite, la fonction compare les intervalles marqués d'un astérisque (*) avec ceux de `transAuto`. Pour chaque intervalle avec un astérisque, elle cherche un intervalle correspondant dans `transAuto` où les heures de début et de fin correspondent. Si un tel intervalle est trouvé et que son label contient un seul mot, l'astérisque est remplacé par ce mot. Si le label contient plusieurs mots, l'intervalle est divisé en plusieurs intervalles, chacun correspondant à un mot du label, en répartissant l'intervalle initial de manière équitable entre les nouveaux intervalles.

Si aucun intervalle correspondant n'est trouvé dans `transAuto`, l'intervalle avec l'astérisque est conservé tel quel. Enfin, les entrées finales sont utilisées pour mettre à jour le tier `TokensAlign`, et le fichier TextGrid est sauvegardé avec les modifications apportées.

Cette approche permet de corriger les labels vides dans les fichiers TextGrid en utilisant les informations disponibles dans un autre tier, tout en veillant à maintenir la cohérence des intervalles et la précision des labels.

8. Paramètres

Définir les chemins d'entrée et de sortie pour les fichiers CoNLL-U (*input_conllu_dir*) et TextGrid (*input_textgrid_dir*, *input_textgrid_dir_gold*) , et créer le répertoire de sortie (*output_merged_dir*) s'il n'existe pas.

9. Reconstruction du tier trans

Cette partie permet de parcourir de manière récursive les sous-répertoires d'un répertoire spécifié, `input_textgrid_dir`, à la recherche de fichiers TextGrid se terminant par "*-syl_tok.TextGrid*". Pour chaque fichier trouvé, il effectue une série d'opérations. D'abord, il construit le chemin complet du fichier TextGrid en utilisant `os.path.join(root, file)`. Ensuite, il crée un nouveau répertoire nommé "*new*" dans le même répertoire que le fichier actuel, s'il n'existe pas déjà. Ce nouveau répertoire est destiné à stocker les nouvelles versions des fichiers TextGrid et les copies des fichiers WAV correspondants.

Une fois le répertoire "*new*" créé, le script utilise la fonction `create_new_trans` pour créer une nouvelle version du fichier TextGrid, qui est sauvegardée dans le répertoire "*new*" avec un suffixe "*-new.TextGrid*". Après la création de ce nouveau fichier, il est renommé pour remplacer le suffixe "*-new.TextGrid*" par "*.TextGrid*", alignant ainsi le nom du nouveau fichier avec celui de l'ancien fichier tout en indiquant qu'il s'agit d'une version mise à jour.

Ensuite, le script cherche le fichier WAV correspondant au fichier TextGrid en remplaçant le suffixe "*-syl_tok.TextGrid*" par "*.wav*". Si le fichier WAV existe, il est copié dans le répertoire "*new*". Si le fichier WAV n'est pas trouvé, un message est imprimé pour signaler l'absence du fichier WAV correspondant.

10. Correction de la tokenisation

Comme précédemment, pour chaque sous-répertoire trouvé, il vérifie s'il contient des fichiers TextGrid se terminant par "_M.TextGrid" ou "*MG.TextGrid*". Si de tels fichiers existent, le script procède à leur tokenisation et syllabification en utilisant la fonction `sppas_tokenisation_syllabification`.

Pour chaque fichier TextGrid trouvé, le script construit le chemin complet du fichier en utilisant `os.path.join(subdir_path, file)`. Ensuite, il construit le chemin complet du fichier WAV correspondant en remplaçant l'extension "*.TextGrid*" par "*.wav*" dans le nom du fichier TextGrid. Si le fichier WAV correspondant existe, le script imprime un message indiquant qu'il va procéder à la tokenisation et syllabification des fichiers TextGrid et

WAV. Ensuite, il appelle la fonction `sppas_tokenisation_syllabification` avec les chemins des fichiers TextGrid et WAV comme arguments pour effectuer ces opérations.

Si le fichier WAV correspondant n'est pas trouvé, le script imprime un message d'avertissement signalant l'absence du fichier WAV. Ce processus permet de s'assurer que chaque fichier TextGrid est correctement tokenisé et syllabisé en fonction de son fichier audio correspondant, facilitant ainsi les analyses et les traitements ultérieurs des données.

11. Fusion des tiers trans originel, TokensAlign et SyllAlign, et transAuto

Cette partie du script vise à fusionner les fichiers TextGrid "gold" et "non-gold" (gold auto) à partir de répertoires spécifiques. Il commence par parcourir récursivement les sous-répertoires du répertoire principal `input_textgrid_dir` pour identifier ceux contenant des fichiers "non-gold" dans un sous-répertoire nommé "new". Pour chaque sous-répertoire identifié, il vérifie l'existence d'un répertoire correspondant dans un autre répertoire principal, `input_textgrid_dir_gold`, où se trouvent les fichiers "gold", bien que le répertoire peut être le même que celui de `input_textgrid_dir`, cette distinction est faite si on prend les vrai fichiers "gold" ayant les annotations des "#". Lorsque ces répertoires sont trouvés, le script appelle d'abord la fonction `generate_tiers_selection_gold_non_gold_silence` pour extraire les niveaux nécessaires des fichiers "gold" et "non-gold" (gold auto). Ensuite, il utilise la fonction `merge_gold_non_gold` pour fusionner les niveaux extraits, créant ainsi un fichier TextGrid combiné qui inclut les informations pertinentes de chaque source. À la fin du processus, le script affiche un message indiquant que l'opération est terminée.

12. Correction de la tokenisation finale

Corrige les tokens vides dans les fichiers TextGrid fusionnés en utilisant les informations du tier transAuto. Utilise la fonction `correct_tokenisation`.

13. Fonction `extraction_sentences_conllu`

La fonction `extraction_sentences_conllu` permet d'extraire les phrases d'un fichier au format CoNLL-U et de les organiser de manière structurée dans un dictionnaire. Lorsqu'elle est exécutée, elle commence par ouvrir le fichier spécifié par le chemin donné en paramètre. Ensuite, elle initialise une variable pour stocker l'identifiant de chaque phrase. En parcourant le fichier ligne par ligne, la fonction vérifie si la ligne contient un identifiant de phrase, identifié par le préfixe `# sent_id =`. Lorsqu'un tel identifiant est trouvé, il est extrait et le nom du fichier est déduit de l'identifiant. Si ce nom de fichier n'existe pas encore dans le dictionnaire de sortie, une nouvelle entrée est créée pour ce fichier. Lorsque la fonction rencontre une ligne contenant le texte de la phrase, précédée par `# text =`, le texte est extrait et ajouté à la liste associée à l'identifiant de la phrase dans le dictionnaire. Ce processus se répète pour toutes les phrases du fichier. Une fois toutes les lignes traitées, la fonction retourne le dictionnaire structuré, où chaque clé est le nom du fichier et chaque valeur est un autre dictionnaire contenant les phrases extraites, indexées par leur identifiant.

14. Fonction `extraction_tokens_textgrid`

La fonction `extraction_tokens_textgrid` est conçue pour extraire les tokens d'un fichier TextGrid. Elle prend en entrée le chemin vers un fichier TextGrid, et elle commence par vérifier si ce fichier existe. Si le fichier n'est pas trouvé, une exception `FileNotFoundError` est levée, avec un message indiquant l'absence du fichier spécifié.

Une fois le fichier localisé, la fonction ouvre le fichier TextGrid en utilisant la bibliothèque `praatio` et vérifie si le niveau de tiers nommé "*TokensAlign*" est présent dans le fichier. Si ce niveau est absent, la fonction lève une exception `ValueError` en signalant que le niveau requis n'a pas été trouvé.

Si le niveau "*TokensAlign*" est présent, la fonction procède à l'extraction des tokens. Elle parcourt chaque intervalle du niveau "*TokensAlign*", extrayant le label du token, ainsi que les temps de début et de fin associés à cet intervalle. Ces informations sont ensuite ajoutées à une liste sous forme de tuples, où chaque tuple contient le label du token, le temps de début, et le temps de fin. Enfin, la fonction retourne cette liste de tokens extraits.

15. Fonction `check_special_characters`

La fonction `check_special_characters` est conçue pour vérifier si un token extrait d'un fichier CoNLL-U contient des caractères spéciaux par rapport à un token extrait d'un fichier TextGrid. Cette fonction prend en entrée quatre paramètres : le token de la phrase (`sent_token`) extrait du fichier CoNLL-U, le token du niveau de tiers (`tier_token`) extrait du fichier TextGrid, l'index du token dans la liste des tokens du niveau de tiers (`tier_index`), et la liste des tokens du niveau de tiers (`tier`).

Le processus commence par initialiser une variable `move_step` à zéro, qui servira à indiquer combien de tokens doivent être déplacés pour corriger le token. La fonction examine ensuite diverses conditions basées sur la présence de caractères spéciaux, tels que les apostrophes ou les points, et ajuste le token en conséquence.

Si le token de la phrase contient un apostrophe et que le token du niveau de tiers ne le contient pas, plusieurs conditions sont vérifiées pour déterminer comment ajuster le token. Par exemple, si le token est "*DO*" et que les tokens suivants dans la liste sont "*N*" et "*T*", le code ajoute les tokens et l'apostrophe appropriée pour corriger le token.

Des ajustements similaires sont effectués pour les cas de ponctuation, comme les points et les tirets. Si le token de la phrase contient un point, des conditions spécifiques permettent d'ajouter ou de modifier le token du niveau de tiers pour refléter correctement la ponctuation.

Après avoir effectué les ajustements nécessaires, la fonction renvoie le token modifié ainsi que le nombre de tokens à déplacer (`move_step`) pour appliquer la correction. Cette fonction permet ainsi d'aligner plus précisément les tokens entre les fichiers CoNLL-U et TextGrid en tenant compte des caractères spéciaux et des règles spécifiques de correction.

16. Fonction `get_alignement_informations`

La fonction `get_alignement_informations` vise à comparer les tokens d'un fichier TextGrid avec les phrases d'un fichier CoNLL-U pour récupérer les informations nécessaires à l'alignement de chaque token correspondant pour chaque phrase. Elle prend en paramètre un dictionnaire contenant les phrases extraites du fichier CoNLL-U (`conllu`), une liste de tokens extraits du fichier TextGrid (`tier_tok`), et le nom du fichier (`filename`).

Pour commencer, la fonction convertit le nom du fichier en enlevant son extension, puis vérifie si ce nom est présent dans le dictionnaire CoNLL-U. Si ce n'est pas le cas, une erreur est levée. Ensuite, des variables sont initialisées pour garder la trace des indices des tokens, des temps de fin précédents (`prev_xmax`) et de l'index de la phrase.

La fonction parcourt ensuite chaque phrase du fichier CoNLL-U. Pour chaque phrase, elle extrait le texte de la phrase et les tokens, puis initialise une liste pour stocker les informations des tokens. Pour chaque token de la phrase, elle vérifie s'il correspond à un token du TextGrid. Si le token contient des caractères spéciaux comme des apostrophes, la fonction `check_special_characters` est appelée pour effectuer les ajustements nécessaires.

Si le token de la phrase et celui du TextGrid correspondent, la fonction calcule les temps d'alignement (`alignbegin` et `alignend`) et les ajoute aux informations du token. Si le token est une ponctuation, les temps d'alignement sont définis par rapport au temps de fin du token précédent. Si un token est remplacé par un `#` dans le TextGrid, les temps d'alignement sont définis en conséquence, avec `X` indiquant des valeurs non alignées.

La fonction gère également les cas où des tokens consécutifs doivent être combinés ou ajustés, notamment lorsqu'un token du TextGrid est suivi d'un espace ou d'une ponctuation. À la fin de chaque phrase, les informations des tokens, y compris les temps d'alignement et les identifiants des mots, sont ajoutées au dictionnaire des phrases.

Les phrases alignées sont ensuite stockées dans le dictionnaire `sentences` avec les informations des tokens et les nouvelles phrases alignées. Ce processus permet de créer un alignement précis des tokens entre les fichiers CoNLL-U et TextGrid, en tenant compte des caractères spéciaux et des ponctuations.

17. Fonction `save_alignment_results`

La fonction `save_alignment_results` permet de sauvegarder les résultats d'alignement dans un fichier au format CSV. Cette fonction prend deux paramètres : le premier, `sentences`, est un dictionnaire où chaque clé représente un identifiant de phrase (`sent_id`) et chaque valeur est un autre dictionnaire contenant des informations sur la phrase, à savoir le texte original (`sent_text`) et le texte modifié (`new_sent_text`). Le second paramètre, `output_file`, est le chemin du fichier CSV dans lequel les résultats doivent être enregistrés. Lors de l'exécution, la fonction ouvre le fichier en mode écriture, en s'assurant de gérer correctement les nouvelles lignes pour éviter les erreurs de formatage. Elle écrit ensuite les en-têtes de colonne : `sent_id`, `sent_text`, et `new_sent_text`. Pour chaque phrase, la fonction extrait les informations nécessaires et les enregistre dans le fichier CSV, ligne par ligne, en veillant à ce que chaque enregistrement soit correctement aligné avec les en-têtes définis. Cette procédure garantit que les données d'alignement sont stockées de manière claire et structurée, facilitant ainsi leur analyse ultérieure.

18. Fonction `update_textgrid`

La fonction `update_textgrid` permet de mettre à jour un fichier TextGrid avec des informations d'alignement de phrases et de mots. Cette fonction prend deux paramètres : `sentences`, un dictionnaire contenant des données d'alignement pour chaque phrase, et `input_file`, le chemin du fichier TextGrid à mettre à jour. La fonction commence par ouvrir le fichier TextGrid en mode lecture à l'aide de la bibliothèque `praatio`, tout en incluant les intervalles vides pour garantir que toutes les sections du fichier sont prises en compte. Ensuite, elle initialise quatre listes vides pour stocker les informations des différents niveaux d'intervalle : `sent_idx_tier` pour les identifiants de phrase, `sent_text_tier` pour le texte des phrases, `word_id_tier` pour les identifiants de mots, et `word_text_tier` pour le texte des mots.

La fonction utilise une liste de ponctuations pour filtrer les tokens non pertinents. Pour chaque entrée dans le dictionnaire `sentences`, elle extrait les informations de début et de fin de l'alignement de la phrase, puis vérifie si ces valeurs sont valides (c'est-à-dire que le début est antérieur à la fin). Si c'est le cas, elle ajoute les informations d'alignement de la phrase aux listes appropriées. La phrase est ensuite divisée en mots, et chaque token est comparé aux mots de la phrase pour s'assurer qu'il correspond correctement. Les alignements de mots sont ajoutés aux listes `word_id_tier` et `word_text_tier`, en vérifiant que les valeurs d'alignement sont valides.

Après avoir traité toutes les phrases et mots, la fonction crée quatre niveaux d'intervalle (`IntervalTier`) pour les identifiants de phrase, le texte des phrases, les identifiants de mots, et le texte des mots, et les ajoute au fichier TextGrid. Enfin, le fichier est sauvegardé sous le même nom avec le format "*long_textgrid*" pour conserver la structure détaillée des données, incluant les espaces vides. Cette procédure assure que le fichier TextGrid reflète avec précision les alignements des phrases et des mots, facilitant ainsi leur analyse et leur utilisation ultérieure.

19. Fonction `build_textgrid_index`

La fonction `build_textgrid_index` est conçue pour créer un index des fichiers TextGrid situés dans un répertoire spécifié et pour construire un dictionnaire de correspondance entre les noms des fichiers et leurs chemins complets. Elle prend un paramètre, `base_path`, qui est le chemin vers le répertoire contenant les fichiers TextGrid. La fonction commence par initialiser un dictionnaire vide nommé `textgrid_index`, qui sera utilisé pour stocker les correspondances entre les noms de fichiers et leurs chemins. En utilisant la fonction `os.walk`, la fonction parcourt récursivement tous les sous-répertoires et fichiers présents dans le répertoire spécifié par `base_path`. Pour chaque fichier trouvé, elle vérifie si le nom du fichier se termine par `"-merged.TextGrid"`, indiquant qu'il s'agit d'un fichier TextGrid pertinent pour l'indexation.

Lorsque cette condition est remplie, la fonction extrait le nom du fichier en supprimant la partie après le premier tiret dans le nom du fichier à l'aide de la méthode `split`. Elle utilise ce nom de fichier (sans la partie après le tiret) comme clé dans le dictionnaire `textgrid_index` et associe cette clé au chemin complet du fichier, obtenu par la fonction `os.path.join`. Après avoir indexé tous les fichiers pertinents, la fonction renvoie le dictionnaire `textgrid_index`, offrant ainsi une structure de données efficace pour accéder rapidement aux fichiers TextGrid basés sur leurs noms. Cette méthode permet de gérer et d'organiser les fichiers TextGrid de manière systématique, facilitant leur utilisation ultérieure dans des traitements ou des analyses.

20. Construction de l'index et traitement final

Le bloc de code commence par construire un index des fichiers TextGrid à l'aide de la fonction `build_textgrid_index`, en fournissant le chemin du répertoire contenant les fichiers TextGrid fusionnés (`output_merged_dir`). Ce dictionnaire `textgrid_index` associe les noms de fichiers TextGrid à leurs chemins complets. Ensuite, le code parcourt récursivement les fichiers dans le répertoire d'entrée CONLLU (`input_conllu_dir`) à l'aide de `os.walk`. Pour chaque fichier dont le nom se termine par `_MG.conllu` ou `_M.conllu`, le chemin complet est construit et le nom du fichier est utilisé pour déterminer le nom associé du fichier TextGrid. Si le fichier CONLLU commence par "ABJ", les trois premiers segments du nom de fichier sont utilisés pour construire le nom du fichier TextGrid ; sinon, seuls les deux premiers segments sont pris en compte. Ce nom est utilisé pour générer le chemin potentiel du fichier TextGrid dans le répertoire de sortie.

Le code recherche ensuite le fichier TextGrid correspondant dans l'index en utilisant le nom du fichier comme clé. Si un fichier TextGrid correspondant est trouvé, la fonction `extraction_sentences_conllu` est appelée pour extraire les phrases du fichier CONLLU et les stocker dans un dictionnaire `conllu_dict`. Les tokens sont extraits du fichier TextGrid à l'aide de la fonction `extraction_tokens_textgrid`. Ensuite, la fonction `get_alignement_informations` est utilisée pour obtenir les informations d'alignement entre les phrases et les tokens. Ces informations sont ensuite mises à jour dans le fichier TextGrid à l'aide de la fonction `update_textgrid`, et le dictionnaire `all_sentences` est mis à jour avec les nouvelles données.

Enfin, le code imprime un message confirmant que le fichier TextGrid a été mis à jour. Les lignes commentées à la fin permettent que les résultats d'alignement soient sauvegardés dans un fichier CSV et que la fin du processus soit notifiée. Le code est conçu pour automatiser le processus de mise à jour et d'alignement des fichiers TextGrid à partir des données CONLLU, tout en fournissant des points de contrôle pour vérifier le bon déroulement des opérations.

5. *5-update_conllu_pause.ipynb*

1. Importation des modules et configuration des chemins

- **Importation des bibliothèques** : *re* pour les expressions régulières, *os* pour la manipulation des chemins de fichiers, *sys* pour ajouter des chemins au système, et *praatio* pour la gestion des fichiers TextGrid.
- **Configuration des chemins :** Définit les chemins actuels et ceux des scripts, et ajoute dynamiquement le chemin des scripts au système afin de pouvoir importer des modules personnalisés (*conll3*).

2. Fonction `load_textgrid`

La fonction `load_textgrid` permet de charger et de récupérer les données d'un fichier TextGrid au format dictionnaire. Le chemin du fichier TextGrid est fourni en argument à la fonction. Le fichier est ouvert à l'aide de la bibliothèque `tgio`, et le nom du fichier, sans l'extension `.TextGrid`, est utilisé comme clé principale dans le dictionnaire `sentences`.

La fonction commence par vérifier l'existence des niveaux de tiers nécessaires dans le fichier TextGrid, à savoir "*Sent-ID*", "*Sent-Text*", "*Word-ID*" et "*Word-Text*". Ces tiers contiennent respectivement les identifiants des phrases, le texte des phrases, les identifiants des mots et le texte des mots. Si tous ces niveaux de tiers sont présents, la fonction procède à leur récupération.

Si l'argument `alignment` est défini sur `True`, la fonction extrait les informations d'alignement des mots dans chaque phrase. Pour chaque phrase, identifiée par `sent_id` et son texte associé `sent_text`, un sous-dictionnaire est créé pour contenir les mots associés à cette phrase. Pour chaque mot, les informations extraites comprennent l'identifiant du mot (`word_id`), ses coordonnées temporelles (`start`, `end`) et son texte (`word_text`). Ces informations sont stockées dans un sous-dictionnaire sous la clé "words" pour chaque phrase.

Si `alignment` est défini sur `False`, le dictionnaire ne contient que les identifiants des phrases et leur texte associé, sans les détails des mots.

Enfin, la fonction retourne le dictionnaire `sentences` contenant toutes les phrases et, si applicable, les mots alignés pour chaque fichier TextGrid.

3. Fonction `update_sent_text_conllu`

La fonction `update_sent_text_conllu` permet de modifier les textes des phrases dans un fichier CoNLL-U en se basant sur les informations fournies dans un dictionnaire. Les paramètres de la fonction sont le chemin du fichier CoNLL-U (`conllu`), un dictionnaire contenant les phrases à mettre à jour (`sentences`), et le chemin vers le fichier de sortie où le fichier CoNLL-U mis à jour sera enregistré (`out`).

La fonction commence par lire le contenu du fichier CoNLL-U. Le contenu est stocké sous forme de chaîne de caractères et est divisé en sections correspondant à chaque phrase. Le nom du fichier CoNLL-U, sans l'extension `.conllu`, est utilisé pour déterminer le nom du fichier TextGrid associé, en ajoutant le suffixe "-merged".

Pour chaque phrase dans le fichier CoNLL-U, la fonction vérifie si elle contient un identifiant de phrase (`sent_id`). Si l'identifiant est présent et correspond à une clé dans le dictionnaire `sentences`, la fonction extrait le nouveau texte de la phrase à partir du dictionnaire. Ensuite, elle remplace l'ancien texte de la phrase dans le fichier CoNLL-U par le nouveau texte. Les anciennes et nouvelles versions du texte des phrases sont enregistrées dans les dictionnaires `old_sent` et `new_sent`, respectivement.

Enfin, la fonction écrit le contenu mis à jour dans un nouveau fichier CoNLL-U à l'emplacement spécifié par le paramètre `out`. Les phrases mises à jour sont regroupées et séparées par des doubles sauts de ligne. La fonction retourne deux dictionnaires : `old_sent`, qui contient les anciens textes des phrases, et `new_sent`, qui contient les nouveaux textes mis à jour. Cette fonction est utile pour synchroniser les fichiers CoNLL-U avec les mises à jour

apportées aux textes des phrases dans des fichiers associés.

4. Fonction `compare_and_replace`

La fonction `compare_and_replace` sert à harmoniser les annotations de tokens entre deux ensembles : les anciens et les nouveaux. Elle prend en entrée deux dictionnaires : `old_tokens` et `new_tokens`, où les clés représentent les indices des tokens et les valeurs sont les tokens eux-mêmes. L'objectif est de mettre à jour les annotations nouvelles pour refléter les annotations anciennes lorsque les tokens ont été modifiés, et d'ajouter les tokens qui ne figuraient pas dans les anciennes annotations.

La fonction commence par créer un dictionnaire `old_token_occurrences` pour suivre les occurrences des tokens dans les anciennes annotations, en les stockant par token converti en majuscules. Cela permet de rechercher rapidement les indices des tokens anciens. Ensuite, elle itère à travers les nouveaux tokens pour les comparer avec les anciens.

Pour chaque nouveau token, la fonction vérifie s'il existe dans les anciens tokens. Si c'est le cas, elle cherche un match en comparant le contexte des tokens voisins (les tokens précédents et suivants). Si un token ancien est trouvé avec le même contexte que le nouveau token, le nouveau token est remplacé par l'ancien dans le dictionnaire `new_tokens_dic`.

Si le token nouveau ne correspond pas exactement à un ancien token, la fonction tente de gérer les différences de format, comme la présence de `~` dans les tokens. Elle compare les tokens après avoir supprimé ces caractères pour trouver une correspondance possible en se basant également sur le contexte.

Si aucun match n'est trouvé, le nouveau token est conservé tel quel dans le dictionnaire de sortie. À la fin du processus, la fonction retourne le dictionnaire mis à jour `new_tokens_dic`, qui contient les tokens corrigés et ajoutés. Cette fonction est utile pour aligner les annotations de tokens entre différentes versions de données ou pour corriger des erreurs dans les nouvelles annotations en se basant sur des annotations antérieures.

5. Fonction `update_conllu_id`

La fonction `update_conllu_id` est conçue pour mettre à jour les indices des tokens et ajuster les indices des têtes dans un fichier CoNLL-U. Elle prend trois paramètres : le chemin du fichier CoNLL-U à mettre à jour (`conllu_path`), le chemin du fichier de sortie pour sauvegarder le CoNLL-U mis à jour (`output_path`), et un dictionnaire contenant les phrases anciennes (`old_sent`). La fonction commence par lire le fichier CoNLL-U et divise son contenu en phrases distinctes. Chaque phrase est traitée pour extraire les métadonnées et les lignes de tokens.

La fonction construit un dictionnaire des anciens tokens, où les clés sont les identifiants des phrases et les valeurs sont des dictionnaires associant des indices de tokens à leurs mots. Elle fait de même pour les nouvelles phrases en comparant les phrases anciennes et nouvelles. Si l'identifiant de la phrase est présent dans les deux dictionnaires, la fonction compare les tokens anciens et nouveaux pour identifier les modifications.

Pour chaque token dans les nouvelles phrases, la fonction vérifie si le token existe dans les anciens tokens ou s'il est manquant. Elle utilise la fonction `compare_and_replace` pour harmoniser les nouveaux tokens avec les anciens en remplaçant les tokens modifiés et en ajoutant les nouveaux tokens manquants. Ensuite, elle ajuste les indices des tokens et les indices des têtes (qui indiquent les relations de dépendance entre les tokens) pour tenir compte des changements dans les indices des tokens.

Enfin, la fonction reformate les lignes de tokens corrigées et les écrit dans le fichier de sortie. Elle gère également les erreurs potentielles, telles que des indices hors limites ou des indices de tête incorrects, en affichant des messages d'erreur si nécessaire. Cette fonction est essentielle pour assurer la cohérence entre les annotations de tokens dans les fichiers CONLL-U lorsqu'il y a eu des modifications ou des mises à jour des annotations.

6. Fonction `correct_alignements`

La fonction `correct_alignements` est conçue pour ajuster les alignements dans un fichier CoNLL-U en utilisant les alignements fournis dans un dictionnaire. Le but est de mettre à jour les alignements des tokens dans le fichier CoNLL-U afin qu'ils correspondent aux informations d'alignement spécifiées.

Pour commencer, la fonction lit le contenu du fichier CoNLL-U et le divise en phrases distinctes. Chaque phrase est ensuite analysée pour extraire les métadonnées et les lignes de tokens. Pour chaque phrase, l'identifiant de la phrase est extrait et utilisé pour déterminer le nom du fichier correspondant, basé sur une convention spécifique.

Ensuite, pour chaque token de la phrase, la fonction cherche les alignements dans le dictionnaire en utilisant l'identifiant de la phrase et l'identifiant du token. Les alignements des tokens sont corrigés en fonction des informations fournies dans le dictionnaire, en mettant à jour les valeurs de début et de fin des tokens. Si un token est trouvé dans le dictionnaire, il est mis à jour avec les informations d'alignement appropriées. Si le token est un signe de ponctuation, il reçoit des valeurs d'alignement par défaut basées sur la dernière position connue.

Enfin, les lignes corrigées pour chaque phrase sont réécrites dans le fichier CoNLL-U. Les phrases pour lesquelles les alignements n'ont pas pu être corrigés sont laissées inchangées. Le fichier CoNLL-U est sauvegardé avec les modifications apportées, remplaçant le fichier original.

En résumé, la fonction `correct_alignements` ajuste les informations d'alignement dans un fichier CoNLL-U pour les rendre cohérentes avec les alignements fournis dans un dictionnaire, en tenant compte des ponctuations et des tokens manquants ou modifiés.

7. Fonction `adjust_alignments_delete_x`

La fonction `adjust_alignments_delete_x` est utilisée pour ajuster les alignements dans un fichier CoNLL-U en remplaçant les valeurs 'X' par des valeurs valides. Voici comment elle fonctionne :

La fonction commence par ouvrir et lire le fichier CoNLL-U, le divisant en phrases distinctes. Pour chaque phrase, elle analyse les lignes pour extraire les informations d'alignement des tokens. Les valeurs d'alignement sont extraites des colonnes spécifiées, et les valeurs 'X' (indiquant des alignements inconnus) sont converties en `None`.

Les valeurs de début et de fin des alignements sont ensuite ajustées. La fonction vérifie les alignements manquants et les remplit en utilisant les valeurs disponibles des alignements précédents ou suivants, ou en les ajustant en fonction des alignements connus. Elle s'assure également que les valeurs de début ne sont pas supérieures aux valeurs de fin et ajuste les alignements pour éviter les incohérences.

Pour chaque token dans chaque phrase, la fonction ajuste les alignements en remplaçant les valeurs 'X' par les valeurs calculées. Les alignements sont ensuite mis à jour dans les lignes des tokens. Les lignes ajustées sont ensuite reformatées et ajoutées à la liste des données ajustées.

Enfin, le fichier CoNLL-U est réécrit avec les données ajustées, remplaçant le fichier original. La fonction assure que toutes les valeurs d'alignement sont valides et cohérentes, en supprimant les valeurs 'X' et en les remplaçant par des valeurs calculées à partir des tokens environnants.

8. Paramètres et boucle principale

Dans cette dernière étape, le script définit ensuite les chemins des répertoires d'entrée (*input_textgrid_directory*, *input_conllu_directory*) et de sortie (*conllu_output_directory*) pour les fichiers TextGrid et CoNLL-U.

Pour traiter les fichiers CONLL-U, le script commence par vérifier l'existence du répertoire de sortie pour les fichiers CONLL-U. Si ce répertoire n'existe pas, il est créé à l'aide de `os.makedirs()`. Ensuite, le script parcourt récursivement le répertoire d'entrée des fichiers CONLL-U à l'aide de `os.walk()`. Pour chaque fichier se terminant par "*_MG.conllu*" ou "*_M.conllu*", il détermine le nom du fichier TextGrid associé en fonction du préfixe du nom de fichier CONLL-U. Si le fichier commence par "ABJ", le nom du fichier TextGrid est composé des trois premiers segments du nom de fichier CoNLL-U, tandis que pour les autres fichiers, il est constitué des deux premiers

segments.

Le chemin vers le fichier TextGrid est ensuite construit en joignant le répertoire d'entrée des fichiers TextGrid avec le nom du fichier et le suffixe approprié. Si le fichier TextGrid existe, le script procède à plusieurs étapes de traitement. Tout d'abord, il charge les données du fichier TextGrid en appelant la fonction `load_textgrid()`. Ensuite, il met à jour les textes dans le fichier CoNLL-U en utilisant les informations du TextGrid à l'aide de `update_sent_text_conllu()` et réindexe les tokens avec `update_conllu_id()`. Le script charge de nouveau le fichier TextGrid, cette fois avec l'option d'alignement activée, et corrige les alignements dans le fichier CoNLL-U en appelant `correct_alignements()`. La fonction `adjust_alignments_delete_x()` est commentée, mais elle pourrait être incluse si nécessaire pour ajuster les alignements finaux.

Cette approche garantit que les fichiers CoNLL-U sont correctement mis à jour avec les informations des fichiers TextGrid et que les alignements sont correctement ajustés. Le traitement est effectué uniquement pour les fichiers CoNLL-U ayant un fichier TextGrid associé, assurant ainsi que les opérations sont effectuées uniquement sur les fichiers pertinents.

6. *6-create_slam_tg.ipynb*

1. Importation des modules et configuration des chemins

Le script commence par importer les modules nécessaires : *os* pour la manipulation des chemins de fichiers, *subprocess* pour exécuter des commandes système, *tqdm* pour afficher une barre de progression, *papermill* pour exécuter des notebooks Jupyter, et *praatio* pour travailler avec des fichiers TextGrid. Il définit également les chemins des répertoires d'entrée et de sortie pour les fichiers nécessaires.

2. Fonction `extraction_sentences_conllu`

La fonction `extraction_sentences_conllu` extrait les phrases d'un fichier CoNLL-U et organise ces phrases dans un dictionnaire structuré. Elle prend en paramètre le chemin d'accès au fichier CoNLL-U et retourne un dictionnaire où chaque clé est le nom du fichier et chaque valeur est un autre dictionnaire qui associe les identifiants de phrases à leurs textes respectifs.

Le processus commence par ouvrir le fichier CoNLL-U en mode lecture. Le fichier est parcouru ligne par ligne. Lorsqu'une ligne commence par "# sent_id =", la fonction extrait l'identifiant de la phrase et le nom du fichier associé, en utilisant la première partie de l'identifiant avant le double soulignement ("__"). Si ce nom de fichier n'existe pas encore dans le dictionnaire, il est ajouté comme une nouvelle clé avec une valeur qui est un dictionnaire vide.

Lorsque la ligne commence par "# text =", la fonction extrait le texte de la phrase et l'ajoute à la liste des phrases associées à l'identifiant de phrase courant dans le dictionnaire. Chaque identifiant de phrase est utilisé comme clé dans le sous-dictionnaire du fichier correspondant. Si l'identifiant de phrase n'existe pas encore pour le fichier, une nouvelle entrée est créée avec une liste vide qui sera remplie avec le texte de la phrase.

Finalement, le dictionnaire structuré est retourné, permettant d'accéder aux phrases extraites et à leur organisation par fichier et par identifiant de phrase.

3. Fonction `extraction_tokens_textgrid`

La fonction `extraction_tokens_textgrid` permet d'extraire les tokens d'un fichier TextGrid en spécifiant le chemin d'accès au fichier en tant que paramètre. Elle renvoie une liste de tokens extraits, où chaque token est représenté par un tuple contenant son étiquette, son temps de début et son temps de fin.

La fonction commence par vérifier si le fichier TextGrid existe à l'emplacement spécifié. Si le fichier est introuvable, elle lève une exception `FileNotFoundError` avec un message d'erreur approprié. Si le fichier est présent, il est ouvert en utilisant la fonction `tgio.openTextgrid`, qui est capable de lire les données du fichier, y compris les intervalles vides, grâce à l'argument `includeEmptyIntervals=True`.

Ensuite, la fonction vérifie la présence d'un tier spécifique nommé "*TokensAlign*" dans le fichier TextGrid. Si ce tier n'est pas trouvé, une exception `ValueError` est levée, signalant l'absence de ce tier dans le fichier. Si le tier est présent, il est récupéré et assigné à la variable `tokens_tier`.

La fonction parcourt alors chaque entrée dans le tier "*TokensAlign*". Chaque entrée est un intervalle contenant une étiquette (label), un temps de début (`xmin`) et un temps de fin (`xmax`). Ces informations sont extraites et regroupées dans un tuple, puis ajoutées à une liste appelée `tokens`.

Enfin, la fonction retourne la liste de tuples, chaque tuple représentant un token extrait avec ses informations de temps.

4. Fonction `extraction_syllabes_textgrid`

Même principe mais pour l'extraction des syllables. Elle prend en paramètre le chemin d'accès au fichier TextGrid (`textgrid_file`) et retourne une liste de syllabes, où chaque syllabe est représentée par un tuple contenant son étiquette, son temps de début, et son temps de fin.

La fonction débute par une vérification pour s'assurer que le fichier TextGrid existe à l'emplacement spécifié. Si le fichier n'existe pas, une exception `FileNotFoundError` est levée, accompagnée d'un message d'erreur indiquant que le fichier est introuvable.

Une fois le fichier validé, il est ouvert en utilisant la fonction `tgio.openTextgrid`, avec l'argument `includeEmptyIntervals=True` pour s'assurer que même les intervalles vides sont inclus dans la lecture des données.

Après l'ouverture du fichier, la fonction vérifie la présence d'un tier nommé "*SyllAlign*" dans le fichier TextGrid. Si ce tier est absent, la fonction lève une exception `ValueError`, informant que le tier requis n'a pas été trouvé.

Si le tier est présent, il est récupéré à l'aide de la méthode `getTier` et assigné à la variable `syllabes_tier`. La fonction parcourt ensuite chaque entrée dans ce tier. Chaque entrée représente un intervalle contenant une étiquette (`label`), un temps de début (`xmin`), et un temps de fin (`xmax`). Ces informations sont extraites et regroupées dans un tuple, puis ajoutées à une liste appelée `syllabes`.

En fin de traitement, la fonction retourne la liste `syllabes`, contenant les tuples avec les informations temporelles pour chaque syllabe extraite du fichier TextGrid.

5. Fonction `check_special_characters`

La fonction `check_special_characters` est conçue pour gérer les caractères spéciaux dans les tokens afin d'assurer leur alignement et leur cohérence. Elle prend en entrée un token de phrase (`sent_token`), un token du tier (`tier_token`), l'indice du token dans la liste (`tier_index`), et la liste des tokens du tier (`tier`). Son rôle principal est de vérifier la présence de caractères comme les apostrophes, les points et les tirets dans le `sent_token` et d'ajuster le `tier_token` pour correspondre à la forme attendue.

Pour commencer, la fonction examine si des apostrophes sont présentes dans le `sent_token` et, si elles ne se trouvent pas dans le `tier_token`, elle applique diverses règles de transformation. Par exemple, elle traite les contractions telles que "don’t" ou "can’t", en ajoutant les parties manquantes pour aligner le `tier_token` avec le `sent_token`. Elle gère aussi les possessifs et les formes contractées, comme dans le cas de "it’s" ou "I’m", en modifiant le `tier_token` pour qu'il reflète correctement la forme contractée.

Ensuite, la fonction se penche sur les points présents dans le `sent_token`. Lorsqu’un point est observé dans le `sent_token` mais absent du `tier_token`, elle ajuste ce dernier pour inclure les points nécessaires. Par exemple, si le `sent_token` est "2.5" et que le `tier_token` est "2", elle modifie le `tier_token` en "2.5" pour correspondre au format attendu. Elle s'assure également que les nombres avec des points, tels que "100.9", sont correctement formés en combinant les parties pertinentes du `tier`.

Enfin, la fonction gère les tirets en ajustant le `tier_token` lorsque le `sent_token` en contient. Elle combine les tokens du tier avec des tirets lorsque cela est nécessaire, suivant des règles spécifiques. Par exemple, si le `sent_token` est "UN-AFRICAN" et que le `tier_token` est "UN", elle modifie le `tier_token` en "UN-AFRICAN" pour refléter la composition correcte. Elle applique des règles pour des cas complexes tels que les noms composés ou les termes techniques, assurant ainsi que le `tier_token` reflète fidèlement le `sent_token`.

La fonction retourne le `tier_token` modifié et le nombre de tokens traités (`move_step`), facilitant ainsi la synchronisation et l'alignement précis entre les tokens de la phrase et ceux du tier.

6. Fonction `get_alignement_informations`

La fonction `get_alignement_informations` est conçue pour comparer les tokens extraits d'un fichier TextGrid avec les phrases provenant d'un fichier CoNLL-U afin d'aligner chaque token de manière appropriée. Elle reçoit trois arguments principaux : un dictionnaire `conllu` qui contient les phrases et leurs identifiants, une liste `tier_tok` des tokens extraits du fichier TextGrid avec leurs coordonnées de temps, et une liste `tier_syll` des syllabes avec leurs coordonnées de temps. En outre, elle prend le nom du fichier comme paramètre.

La fonction commence par définir une liste de signes de ponctuation courants qui seront traités séparément. Ensuite, elle initialise diverses variables pour suivre l'état actuel du processus d'alignement, telles que les indices des tokens dans le fichier TextGrid et les informations sur les phrases. Pour chaque phrase dans le dictionnaire `conllu`, la fonction construit une liste de tokens et prépare une liste vide pour les informations d'alignement.

Pour chaque token dans la phrase, la fonction tente de trouver une correspondance dans les tokens du fichier TextGrid. Si une correspondance est trouvée, elle enregistre les informations d'alignement (début et fin) pour ce token. Si le token est de la ponctuation, il est aligné en utilisant les coordonnées du token précédent. En cas de remplacement par '#', la fonction gère le cas en alignant le token en utilisant les coordonnées des tokens voisins ou en le marquant avec des valeurs "X" pour indiquer une absence d'alignement précis. Les cas spéciaux, comme les combinaisons de tokens ou les abréviations, sont également pris en compte avec des traitements spécifiques.

Si un token n'est pas trouvé dans les tokens du fichier TextGrid, il est ajouté avec des valeurs "X" pour les coordonnées. À la fin de chaque phrase, les informations sur les syllabes sont alignées si elles se trouvent à l'intérieur des limites de la phrase. Enfin, les informations sur la phrase, y compris le texte de la phrase, les informations d'alignement des tokens, et les informations sur les syllabes, sont stockées dans un dictionnaire pour chaque phrase. La fonction retourne ce dictionnaire après avoir traité toutes les phrases.

En résumé, cette fonction gère l'alignement des tokens et des syllabes entre les fichiers de texte et les fichiers de tiers, en prenant en compte les particularités des tokens, de la ponctuation et des abréviations pour fournir des informations d'alignement précises pour chaque phrase.

7. Fonction `save_alignment_results`

La fonction `save_alignment_results` permet de sauvegarder les résultats d'alignement dans un fichier CSV. Elle prend deux arguments : `sentences`, un dictionnaire contenant les informations d'alignement pour chaque phrase, et `output_file`, le chemin du fichier où les résultats doivent être enregistrés.

Le processus commence par ouvrir le fichier de sortie en mode écriture, avec un délimiteur de nouvelle ligne pour garantir une bonne mise en forme du fichier CSV. Un objet `writer` est créé à l'aide du module `csv`, et une ligne d'en-tête est écrite dans le fichier pour définir les noms des colonnes : "*sent_id*", "*sent_text*", "*new_sent_text*", "*token_info*", et "*syll_info*".

Ensuite, pour chaque phrase dans le dictionnaire `sentences`, la fonction écrit une ligne dans le fichier CSV contenant les informations suivantes : l'identifiant de la phrase (`sent_id`), le texte original de la phrase (`sent_text`), le texte de la phrase après traitement (`new_sent_text`), les informations sur les tokens alignés (`token_info`), et les informations sur les syllabes alignées (`syll_info`).

Chaque ligne du fichier CSV est composée des données associées à une phrase, permettant ainsi d'exporter les résultats d'alignement sous un format structuré et facile à manipuler. La fonction ne retourne rien ; son seul objectif est de produire un fichier CSV contenant les résultats d'alignement.

8. Fonction `create_textgrid_slam`

La fonction `create_textgrid_slam` génère un fichier TextGrid à partir des informations d'alignement pour l'annotation SLAM. Elle commence par initialiser un objet `Textgrid` pour stocker les différentes couches d'annotation. Ensuite, elle définit une liste de ponctuations à exclure et prépare plusieurs listes pour les niveaux d'annotation, tels que les identifiants de phrases, les textes de phrases, les IDs des mots, les textes des mots, la tokenisation, et les syllabes.

Pour chaque phrase dans le dictionnaire d'entrée, elle extrait les informations nécessaires, convertit les temps d'alignement en secondes, et remplit les listes avec les intervalles appropriés. La fonction traite aussi les chevauchements à l'aide d'une fonction dédiée pour s'assurer que les annotations ne se chevauchent pas de manière incorrecte. Une fois les données préparées, elle crée des objets `IntervalTier` pour chaque type

d'annotation, les ajoute à l'objet TextGrid, puis enregistre le fichier au chemin spécifié, en utilisant le format "*long_textgrid*" et en incluant les espaces vides.

9. Fonction `build_textgrid_index`

La fonction `build_textgrid_index` est conçue pour parcourir un répertoire et créer un index des fichiers TextGrid. Elle prend en entrée le chemin du répertoire de base contenant les fichiers TextGrid. En utilisant `os.walk`, elle explore récursivement tous les sous-répertoires et fichiers présents dans ce répertoire. Pour chaque fichier trouvé, elle vérifie si le nom se termine par "*-merged.TextGrid*". Si c'est le cas, elle extrait le nom de base du fichier en supprimant la partie après le premier tiret, puis elle ajoute ce nom de base comme clé dans un dictionnaire `textgrid_index`, avec le chemin complet du fichier comme valeur associée. Finalement, la fonction retourne ce dictionnaire, permettant ainsi d'obtenir un accès rapide aux fichiers TextGrid indexés par leurs noms de base.

10. Paramètres et boucle principale

Le script définit les chemins des répertoires d'entrée et de sortie pour les fichiers CoNLL-U et TextGrid. Il construit ensuite l'index des fichiers TextGrid en appelant `build_textgrid_index`. Pour chaque fichier CoNLL-U dans le répertoire d'entrée, il génère le nom du fichier TextGrid correspondant et vérifie son existence dans l'index. Si le fichier TextGrid existe, il extrait les phrases du fichier CoNLL-U, les tokens et syllabes du fichier TextGrid, et obtient les informations d'alignement en appelant `get_alignement_informations`. Les résultats sont ensuite mis à jour dans un fichier TextGrid de sortie en appelant `create_textgrid_slam`.

Cette structure de script permet d'assurer que les annotations des fichiers CoNLL-U sont synchronisées avec les alignements des fichiers TextGrid, en gérant les modifications de texte et les alignements de manière cohérente et systématique.

II. Extraction des Données Prosodique et Mise à Jour des CoNLL-U

Dossier *Prosodic_Data*

1. *fill_conllus_prosody.ipynb*

Je détaille seulement les fonctions que j'ai rajouté au script déjà existant.

1. Importation des modules et configuration des chemins

- **Importation des bibliothèques** : *glob* pour la recherche de fichiers selon des motifs, *math* pour des calculs mathématiques, os pour la manipulation des chemins de fichiers, *re* pour les expressions régulières, *numpy* pour les calculs numériques, *tgt* pour travailler avec les fichiers TextGrid, *parselmouth* pour traiter les fichiers audio, et *cmudict* de NLTK pour le dictionnaire de prononciation.
- **Configuration des chemins :** Définit les chemins des répertoires d'entrée et de sortie pour les fichiers nécessaires.

2. Fonction `extract_trees_and_metadata`

La fonction `extract_trees_and_metadata` extrait des arbres syntaxiques et leurs métadonnées à partir d'un fichier au format CoNLL. Elle prend en entrée le chemin vers le fichier (`file_path`). La fonction commence par utiliser `conllFile2trees` pour extraire les arbres syntaxiques du fichier et obtient le nom du fichier à partir du chemin en utilisant `os.path.basename(file_path)`. Elle initialise ensuite une liste vide `metadata` pour stocker les métadonnées de chaque arbre. Pour chaque arbre, elle convertit l'arbre en chaîne de caractères, cherche l'identifiant de la phrase (`sent_id`) à l'aide d'une expression régulière, et recueille les mots de l'arbre. Les métadonnées, incluant l'arbre, l'identifiant de la phrase et les mots, sont ajoutées à la liste `metadata`. Finalement, la fonction retourne une tuple contenant la liste des arbres (`trees`), le nom du fichier (`file_name`) et la liste des métadonnées (`metadata`).

3. Fonction `extract_values_from_pitchtier`

La fonction `extract_values_from_pitchtier` extrait les valeurs de fréquence fondamentale (pitch) d'un fichier pitchtier en prenant en entrée une liste de lignes du fichier (`file_content`) et deux entiers représentant le début (`align_begin`) et la fin (`align_end`) de l'intervalle d'alignement. La fonction initialise deux listes vides, `values` et `numbers`, pour stocker respectivement les valeurs de pitch et les numéros associés. Elle parcourt chaque ligne du fichier, vérifiant si la ligne contient "number =" ou "value =". Lorsqu'une ligne contient "number =", la fonction extrait et convertit le numéro en un flottant. Lorsqu'une ligne contient "value =", la fonction extrait et convertit la valeur en un flottant et vérifie si le numéro extrait précédemment se situe dans l'intervalle d'alignement spécifié. Si c'est le cas, la valeur et le numéro sont ajoutés aux listes `values` et `numbers`. Enfin, la fonction retourne les listes `values` et `numbers`.

4. Fonction `is_number`

Cette fonction vérifie si une chaîne peut être convertie en nombre flottant.

5. Fonction `extract_pitchtier_infos`

La fonction `extract_pitchtier_infos` prend en entrée le début et la fin de l'intervalle d'alignement pour la phrase (`sent_begin`, `sent_end`), un arbre de tokens (`tok_tree`), et le chemin vers un fichier pitchtier (`pitchtier_file`). Elle commence par convertir les valeurs de début et de fin d'alignement en secondes si elles sont valides. Ensuite, elle lit le contenu du fichier pitchtier et utilise la fonction `extract_values_from_pitchtier` pour extraire les valeurs et les numéros de fréquence fondamentale correspondant à l'intervalle d'alignement de la phrase. La fonction construit ensuite un dictionnaire des caractéristiques des syllabes (`misc_dict`) à partir des informations du token tree. Pour chaque syllabe (jusqu'à huit), elle extrait les valeurs de début et de fin d'alignement si elles existent, les convertit en secondes, et utilise la fonction `extract_values_from_pitchtier` pour obtenir les valeurs et numéros de fréquence fondamentale pour chaque intervalle d'alignement de syllabe. Ces informations sont stockées dans un dictionnaire `Syls_infos`. Enfin, la fonction compile les informations de fréquence fondamentale pour la phrase dans un dictionnaire `sent_pitchtier_infos` et retourne les deux dictionnaires, `Syls_infos` et `sent_pitchtier_infos`.

6. Fonction `semitones_between`

Cette fonction calcule le nombre de demi-tons entre deux fréquences.

7. Fonction `frequency_from_semitones`

La fonction `frequency_from_semitones` prend en entrée deux paramètres : `frequency`, qui représente une fréquence de base en Hertz, et `semitones`, qui représente un nombre de demi-tons. La formule utilisée pour calculer la fréquence résultante repose sur le principe que chaque demi-ton correspond à une multiplication de la fréquence par la racine douzième de 2 (environ 1.059463). La fonction élève 2 à la puissance du nombre de demi-tons divisé par 12, puis multiplie le résultat par la fréquence de base pour obtenir la nouvelle fréquence. La fonction retourne cette fréquence en Hertz. Ainsi, elle permet de déterminer la fréquence correspondant à un certain nombre de demi-tons au-dessus de la fréquence initiale (ici 2).

8. Fonction `hertz_semiton_data`

La fonction `hertz_semiton_data` calcule la fréquence fondamentale moyenne (Mean F0) et les demi-tons (semitones) entre les syllabes et la phrase pour un token donné. Elle prend en entrée trois dictionnaires : `syls_infos` contenant les valeurs de fréquence des syllabes, `sent_pitchtier_infos` contenant les valeurs de fréquence de la phrase, et `token_data` représentant les données du token. La fonction commence par calculer la moyenne des fréquences fondamentales de la phrase. Ensuite, elle parcourt chaque syllabe, calcule la moyenne des fréquences fondamentales de chaque syllabe et détermine les demi-tons entre cette moyenne et celle de la phrase, en utilisant les fonctions `semitones_between` et `frequency_from_semitones`. Les résultats sont stockés dans `syls_infos`. La moyenne des fréquences de la phrase est également ajoutée à `sent_pitchtier_infos`. Si le `token_data` contient "root", la moyenne des fréquences de la phrase est ajoutée sous la clé `UtteranceMeanF0` dans `sent_pitchtier_infos`. Enfin, la fonction retourne les dictionnaires mis à jour `syls_infos` et `sent_pitchtier_infos`.

9. Fonction `slope_data`

La fonction `slope_data` prend en entrée un dictionnaire `syls_infos` contenant des informations sur les syllabes, notamment les temps (`numbers`) et les fréquences (`values`). Pour chaque syllabe, la fonction vérifie si les listes de temps et de fréquences ne sont pas vides. Si elles contiennent des données, elle extrait les temps et les fréquences de début et de fin pour calculer les coordonnées (`coordonnee`), le seuil de glissando (`seuil_glissando`), et les demi-tons entre les fréquences de début et de fin (`semiton`) en utilisant la fonction `semitones_between`. La pente (`slope`) est déterminée en comparant les demi-tons au seuil de glissando : si les demi-tons absolus sont supérieurs au seuil et positifs, la pente est "Rise"; si les demi-tons absolus sont supérieurs au seuil et négatifs, la pente est "Fall"; si les demi-tons et le seuil sont tous deux nuls, la pente est "X"; sinon, la pente est "Flat". Si les listes de temps ou de fréquences sont vides, les coordonnées, les demi-tons et la pente sont marqués comme "X". Enfin, la fonction met à

jour `syls_infos` avec les nouvelles informations calculées et retourne ce dictionnaire mis à jour.

10. Fonction `calculate_amplitudes`

La fonction `calculate_amplitudes` prend en entrée le chemin vers un fichier audio (`audio_file`), un temps de début (`start_time`) et un temps de fin (`end_time`). Elle commence par s'assurer que le chemin du fichier audio est une chaîne de caractères, même s'il a été passé sous forme de liste. Ensuite, elle charge le fichier audio en utilisant la bibliothèque `parselmouth`. La fonction vérifie que les temps de début et de fin se situent dans la durée totale du fichier audio pour éviter des erreurs de dépassement. Elle extrait ensuite le segment audio correspondant à l'intervalle de temps spécifié. Ce segment est converti en intensité pour obtenir les valeurs d'amplitude. La fonction calcule ensuite l'amplitude maximale (`max_amp`) et l'amplitude moyenne (`avg_amp`) du segment extrait. Ces valeurs sont arrondies à trois décimales et retournées. Si une exception survient à n'importe quelle étape, la fonction gère l'erreur en renvoyant des amplitudes de 0 pour les deux mesures.

11. Fonction `amplitude_data`

La fonction `amplitude_data` prend en entrée un dictionnaire `syls_infos` contenant des informations sur les syllabes, un dictionnaire `sent_infos` contenant des informations sur la phrase, et le chemin vers un fichier audio (`audio_file`). Elle commence par initialiser les variables `token_start_time` et `token_end_time` à `None`. Ensuite, elle parcourt chaque syllabe dans `syls_infos` pour vérifier la présence des clés `Alignbegin` et `Alignend`, qui indiquent les temps de début et de fin de l'alignement de la syllabe. Si ces clés sont présentes, la fonction met à jour les temps de début et de fin du token si nécessaire, puis appelle la fonction `calculate_amplitudes` avec les temps de début et de fin de la syllabe pour calculer les amplitudes maximale et moyenne. Les résultats sont stockés dans `syl_data` sous les clés `MaxAmplitude` et `AvgAmplitude`. Si le calcul échoue, un message d'erreur est affiché pour la syllabe concernée. Après avoir traité toutes les syllabes, la fonction vérifie si les temps de début et de fin du token ont été définis, puis appelle `calculate_amplitudes` pour calculer les amplitudes maximale et moyenne pour l'ensemble du token. Les résultats sont stockés dans `sent_infos` sous les clés `TokenMaxAmplitude` et `TokenAvgAmplitude`. Si le calcul échoue, un message d'erreur est affiché pour le token. Enfin, la fonction retourne les dictionnaires mis à jour `syls_infos` et `sent_infos`.

12. Paramètres et boucle principale

Le script définit les chemins des répertoires d'entrée et de sortie pour les fichiers CoNLL-U, TextGrid, pitchtier et audio. Il crée les dossiers de sortie si nécessaire.

On commence par initialiser un dictionnaire de prononciation (`cmudict.dict()`) pour gérer les prononciations en fonction de la langue choisie. Il définit plusieurs variables pour les différents niveaux de tiers de syllabes et de mots, tels que `syl_tier` pour le contour global, `syl_tier2` pour le contour local, `word_tier` pour l'ID numérique du token, `word_text_tier` pour le texte du token, et `syllable_text_tier` pour les transcriptions syllabiques. Ensuite, il utilise `glob.glob` pour lister les fichiers correspondants dans les dossiers spécifiés : `slam_files` pour les fichiers TextGrid, `conll_infiles` pour les fichiers CoNLL-U dans le dossier "gold", `pitchtier_infiles` pour les fichiers PitchTier, et `audio_infiles` pour les fichiers audio au format WAV. Le dossier de sortie pour les nouveaux fichiers CoNLL-U est défini par `conll_outfolder`. Le script vérifie l'existence de ce dossier et le crée s'il n'existe pas. De plus, un dictionnaire `feature_rename_dict` est initialisé pour renommer les caractéristiques si nécessaire lors d'une exécution ultérieure du script.

13. Fonction `main`

La fonction `main` commence par itérer sur chaque fichier SLAM dans `slam_files`, en triant les fichiers par ordre alphabétique. Pour chaque fichier SLAM, elle extrait le nom de base du fichier pour trouver les fichiers correspondants dans `pitchtier_infiles` et `audio_infiles`. Si aucun fichier correspondant n'est trouvé, le script continue avec le fichier SLAM suivant. Ensuite, la fonction extrait les annotations prosodiques du fichier SLAM en

utilisant `extractProsodicAnnotation` et initialise une liste `conllu_outfiles` pour stocker les fichiers CoNLL-U traités.

Pour chaque fichier CoNLL-U dans `conll_infiles` correspondant au nom de base, elle extrait les arbres syntaxiques et les métadonnées avec `extract_trees_and_metadata`. La fonction parcourt chaque token dans chaque arbre pour construire un dictionnaire de fonctionnalités `feature_dico` à partir de `misc`. Les fonctionnalités obsolètes ou à renommer sont supprimées ou mises à jour en fonction de `feature_rename_dict`. Si le token a des annotations prosodiques, celles-ci sont ajoutées au `feature_dico`. La fonction génère une chaîne de fonctionnalités à partir de `feature_dico` et met à jour la propriété `misc` du token. Les arbres mis à jour sont ensuite sauvegardés dans des fichiers CoNLL-U.

Après cette première étape, pour chaque fichier CoNLL-U traité, la fonction extrait à nouveau les arbres syntaxiques et les métadonnées. Pour chaque token, elle calcule les débuts et fins d'alignement de la phrase. Les informations prosodiques sont extraites pour chaque token non ponctué à partir des fichiers pitchtier et audio, en utilisant les fonctions `extract_pitchtier_infos`, `hertz_semiton_data`, `slope_data` et `amplitude_data`. Les résultats sont ajoutés au `feature_dico` de chaque token et les arbres mis à jour sont sauvegardés dans les fichiers CoNLL-U d'origine. Enfin, la fonction `main` est appelée avec les listes de fichiers d'entrée et de sortie appropriées pour exécuter l'ensemble du traitement.

Cette structure de script permet d'assurer que les annotations prosodiques des fichiers TextGrid sont synchronisées avec les informations de pitch et d'amplitude des fichiers pitchtier et audio, et qu'elles sont intégrées de manière cohérente et systématique dans les fichiers CoNLL-U.

III. Comparaison Gold et Gold Auto

Dossier *Prosodic_Data*

1. *metrics_gold_auto.ipynb*

1. Importation des modules et configuration des chemins

- **Importation des bibliothèques** : glob pour la recherche de fichiers selon des motifs, os pour la manipulation des chemins de fichiers, pandas pour la manipulation des données, collections pour les structures de données avancées, matplotlib pour la visualisation des données, pdfkit pour la conversion des fichiers HTML en PDF, tqdm pour les barres de progression et outils.conll3 pour des fonctions spécifiques au traitement des fichiers CoNLL-U.
- **Configuration des chemins** : Définit également le chemin vers wkhtmltopdf pour pdfkit.

2. Fonction `dico2FeatureString`

La fonction `dico2FeatureString` prend en entrée un dictionnaire `dico` contenant des caractéristiques sous forme de paires clé-valeur. Elle crée une liste de chaînes où chaque chaîne est une paire clé-valeur du dictionnaire, jointe par un signe égal (`=`). Cette liste est ensuite jointe par des barres verticales (`|`) pour former une seule chaîne de caractères qui représente toutes les caractéristiques du dictionnaire. Cette chaîne est alors retournée par la fonction. En somme, cette fonction convertit les entrées du dictionnaire en une représentation textuelle standardisée utilisée dans les fichiers CONLLU pour décrire des caractéristiques supplémentaires des tokens.

3. Fonction `build_feature_dico`

La fonction `build_feature_dico` prend en entrée une chaîne de caractères `misc_features_string` qui contient des caractéristiques CONLLU sous forme de paires clé-valeur séparées par des barres verticales (`|`). Elle initialise un dictionnaire vide `feature_dico`. Ensuite, elle divise la chaîne d'entrée en plusieurs sous-chaînes en utilisant la barre verticale comme séparateur. Pour chaque sous-chaîne résultante, la fonction divise à nouveau la sous-chaîne en une clé et une valeur en utilisant le signe égal (`=`) comme séparateur. Ces paires clé-valeur sont ajoutées au dictionnaire `feature_dico`. La fonction retourne ensuite ce dictionnaire, permettant d'accéder facilement aux caractéristiques individuelles du token.

4. Fonction `extract_trees_and_metadata`

La fonction `extract_trees_and_metadata` lit un fichier CoNLL-U et extrait les arbres syntaxiques (phrases) et les métadonnées. Elle initialise des listes et des dictionnaires pour stocker les métadonnées, les phrases et les silences. En ouvrant le fichier CoNLL-U, la fonction lit chaque ligne et la nettoie des espaces superflus. Si une ligne commence par un caractère "#", elle est considérée comme une métadonnée. Les métadonnées sont stockées dans un dictionnaire `current_metadata`, où les clés et les valeurs sont séparées par le signe "=". Si une ligne contient un token de la phrase, les informations du token sont extraites et stockées dans un dictionnaire `current_token`. Si le token n'est pas une ponctuation, il est ajouté à la liste `current_sentence`. Si le token est une ponctuation représentant un silence (indiqué par le caractère "#"), il est ajouté à la liste `current_silence`. Lorsque la fonction rencontre une ligne vide, cela indique la fin d'une phrase ou d'un silence. La phrase ou le silence actuel est alors ajouté à la liste correspondante (`sentences` ou `silences`), et les listes temporaires sont réinitialisées.

5. Fonction `count_syllable_features`

La fonction `count_syllable_features` prend en entrée deux listes de phrases, `gold_sentences` et `auto_sentences`. Pour chaque phrase, elle itère sur les tokens des phrases gold et auto. Si le token n'est pas une ponctuation, la fonction extrait les caractéristiques `misc` des tokens gold et auto, et les convertit en dictionnaires via `build_feature_dico`. Elle compte ensuite les occurrences de formes syllabiques spécifiques et les onsets externes dans les phrases gold et auto, en excluant certaines caractéristiques non pertinentes. Les résultats sont stockés dans des dictionnaires `gold_syllable_form_count` et `auto_syllable_form_count` pour les syllabes, et `gold_external_onset_count` et `auto_external_onset_count` pour les onsets externes. La fonction calcule également le total des syllabes pour les phrases gold et auto.

Après avoir comptabilisé les syllabes et les onsets externes, elle combine les résultats dans des dictionnaires `combined_count` et `combined_onsets`, en unissant les clés des deux sources de données (gold et auto). Ces dictionnaires sont ensuite convertis en DataFrames pandas pour une présentation tabulaire des résultats, avec des colonnes pour les comptages des syllabes et des onsets dans les phrases gold et auto. La fonction renvoie finalement ces DataFrames ainsi que les totaux des syllabes pour les phrases gold et auto.

6. Fonction `count_mono_syllable_features`

La fonction `count_mono_syllable_features` prend en entrée deux listes de phrases, `gold_sentences` et `auto_sentences`. Elle initialise un dictionnaire imbriqué `syllable_form_count` pour stocker les comptages des caractéristiques des monosyllabes, ainsi que des compteurs pour le total des syllabes dans les phrases gold et auto. Pour chaque phrase dans les listes d'entrée, la fonction itère sur les tokens correspondants des phrases gold et auto. Si le token n'est pas une ponctuation, la fonction extrait les caractéristiques `misc` des tokens gold et auto, et les convertit en dictionnaires via `build_feature_dico`. Elle vérifie ensuite si le token est un monosyllabe (valeur de `SyllableCount` égale à '1'). Pour les monosyllabes, la fonction incrémente le compteur correspondant dans `syllable_form_count` pour chaque caractéristique pertinente, excluant certaines caractéristiques spécifiques telles que `MeanF0`, `SemitonesFromUtteranceMean`, `Semiton`, etc. La fonction met à jour les totaux des syllabes pour les phrases gold et auto en conséquence.

7. Fonction `slam`

Cette fonction compte les caractéristiques SLAM dans les phrases "gold" et "auto". Elle extrait les caractéristiques globales et locales des syllabes et les stocke dans des dictionnaires.

La fonction `slam` prend en entrée deux listes de phrases annotées, `gold_sentences` et `auto_sentences`, et compte différentes caractéristiques prosodiques pour chacune. Elle initialise plusieurs dictionnaires `defaultdict` pour stocker les comptages des caractéristiques comme `AvgHeightGlo`, `AvgHeightLoc`, `PitchRangeGlo`, `PitchRangeLoc`, `Glo`, et `Loc` pour les phrases gold et auto. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si le token n'est pas une ponctuation, la fonction extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, elle compte les occurrences des différentes caractéristiques prosodiques pour les phrases gold et auto, en mettant à jour les dictionnaires de comptages respectifs.

Après avoir comptabilisé les caractéristiques prosodiques, la fonction combine les résultats pour chaque type de caractéristique en un dictionnaire `combined_count`. Elle utilise cet ensemble de données combiné pour créer des DataFrames pandas qui présentent les comptages des caractéristiques pour les phrases gold et auto. Pour chaque type de caractéristique (`AvgHeightGlo`, `AvgHeightLoc`, `PitchRangeGlo`, `PitchRangeLoc`, `Glo`, et `Loc`), elle crée un DataFrame distinct avec les comptages correspondants et renomme les colonnes pour une meilleure lisibilité. Enfin, la fonction retourne ces DataFrames.

8. Fonction `slope`

La fonction `slope` compte les caractéristiques de pente dans les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto). Elle suit un processus similaire aux fonctions précédentes, avec quelques étapes spécifiques pour les caractéristiques de pente.

La fonction `slope` commence par initialiser des dictionnaires `defaultdict` pour stocker les comptages des caractéristiques de pente globales (`SlopeGlo`), locales (`SlopeLoc`), et générales (`Slope`) pour les phrases gold et auto. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si le token n'est pas une ponctuation, elle extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, elle compte les occurrences des différentes caractéristiques de pente pour les phrases gold et auto, en mettant à jour les dictionnaires de comptages respectifs.

Après avoir comptabilisé les caractéristiques de pente, la fonction combine les résultats pour chaque type de caractéristique en un dictionnaire `combined_count`. Elle utilise cet ensemble de données combiné pour créer des DataFrames pandas qui présentent les comptages des caractéristiques pour les phrases gold et auto. Pour chaque type de caractéristique (`SlopeGlo`, `SlopeLoc`, et `Slope`), elle crée un DataFrame distinct et renomme les colonnes pour une meilleure lisibilité. Enfin, elle retourne ces DataFrames.

9. Fonction `count_total_tokens`

Cette fonction compte le nombre total de tokens dans les phrases "gold" et "auto", en excluant les tokens de ponctuation.

10. Fonction `count_matching_number_syllables`

La fonction `count_matching_number_syllables` compte le nombre de tokens dans les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto) ayant le même nombre de syllabes.

La fonction `count_matching_number_syllables` commence par initialiser un compteur `matching_syllable_count` à zéro et calcule le nombre total de tokens dans les phrases gold et auto en utilisant la fonction `count_total_tokens`. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si le token n'est pas une ponctuation, elle extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, elle obtient le nombre de syllabes pour les tokens gold et auto à partir des dictionnaires de caractéristiques. Si les deux tokens ont des valeurs de `SyllableCount` et que ces valeurs sont égales, le compteur `matching_syllable_count` est incrémenté. Enfin, la fonction retourne un dictionnaire contenant le nombre de tokens avec un nombre de syllabes correspondant (`MatchingSyllableCount`) et le nombre total de tokens (`TotalTokens`).

11. Fonction `count_matching_form_syllables`

La fonction `count_matching_form_syllables` compte le nombre de tokens dans les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto) ayant des formes syllabiques correspondantes.

La fonction `count_matching_form_syllables` initialise un compteur `matching_form_syllable_count` à zéro et calcule le nombre total de tokens dans les phrases gold et auto en utilisant la fonction `count_total_tokens`. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si le token n'est pas une ponctuation, elle extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, elle obtient les formes syllabiques des tokens gold et auto en filtrant les clés du dictionnaire qui commencent par `Syl` et en excluant certaines caractéristiques spécifiques comme `SyllableCount`, `MeanF0`, `SemitonesFromUtteranceMean`, etc. Si les listes de formes syllabiques pour les deux tokens sont égales, le compteur `matching_form_syllable_count` est incrémenté. Enfin, la fonction retourne un dictionnaire contenant le nombre de tokens avec des formes syllabiques correspondantes (`MatchingSyllableCount`) et le nombre total de tokens (`TotalTokens`).

Cette méthode permet de comparer de manière précise les formes syllabiques entre les annotations manuelles et automatiques, fournissant des informations sur la cohérence des annotations syllabiques.

12. Fonction `count_matching_token_alignment_features`

La fonction `count_matching_token_alignment_features` compte le nombre de tokens dans les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto) ayant des caractéristiques d'alignement correspondantes.

La fonction `count_matching_token_alignment_features` initialise un compteur `matching_alignment_count` à zéro et calcule le nombre total de tokens dans les phrases gold et auto en utilisant la fonction `count_total_tokens`. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si le token n'est pas une ponctuation, elle extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, elle obtient les caractéristiques d'alignement `AlignBegin` et `AlignEnd` pour les tokens gold et auto. Si les valeurs de `AlignBegin` et `AlignEnd` pour les deux tokens sont égales, le compteur `matching_alignment_count` est incrémenté. Enfin, la fonction retourne un dictionnaire contenant le nombre de tokens avec des caractéristiques d'alignement correspondantes (`MatchingTokenAlignmentCount`) et le nombre total de tokens (`TotalTokens`).

13. Fonction `count_alignment_differences`

La fonction `count_alignment_differences` compte les différences d'alignement entre les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto) en fonction d'intervalles de 10 ms.

La fonction `count_alignment_differences` initialise un dictionnaire `matching_counts` pour stocker les comptages de correspondances d'alignement pour des intervalles de temps allant de 0 à 300 ms par incréments de 10 ms. Elle calcule également le nombre total de tokens dans les phrases gold et auto en utilisant la fonction `count_total_tokens`. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si le token n'est pas une ponctuation, elle extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, elle obtient les caractéristiques d'alignement `AlignBegin` et `AlignEnd` pour les tokens gold et auto, en les convertissant en entiers si elles ne sont pas "X". Si toutes les valeurs d'alignement sont valides, elle calcule les différences absolues entre les alignements de début et de fin pour les tokens gold et auto. Pour chaque différence calculée, elle incrémente le compteur correspondant dans le dictionnaire `matching_counts` en fonction de l'intervalle auquel la différence appartient. Enfin, la fonction retourne un dictionnaire contenant les comptages de correspondances d'alignement par intervalle (`MatchingCountsByInterval`) et le nombre total de tokens (`TotalTokens`).

14. Fonction `count_matching_slam_similarity`

La fonction `count_matching_slam_similarity` compte le nombre de tokens ayant des caractéristiques SLAM correspondantes dans les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto).

La fonction `count_matching_slam_similarity` commence par initialiser des compteurs pour les différentes caractéristiques SLAM : `matching_avgheightglo_count`, `matching_avgheightloc_count`, `matching_pitchrangeglo_count`, `matching_pitchrangeloc_count`, `matching_glo_count`, et `matching_loc_count`. Elle calcule également le nombre total de tokens dans les phrases gold et auto en utilisant la fonction `count_total_tokens`. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si le token n'est pas une ponctuation, elle extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, pour chaque caractéristique SLAM, elle vérifie si les valeurs de la caractéristique dans les tokens gold et auto correspondent. Si elles correspondent, elle incrémente le compteur correspondant. Enfin, la fonction retourne un dictionnaire contenant les comptages de correspondance pour chaque caractéristique SLAM et le nombre total de tokens.

15. Fonction `count_matching_slope_similarity`

La fonction `count_matching_slope_similarity` compte le nombre de tokens ayant des caractéristiques de pente correspondantes dans les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto).

La fonction `count_matching_slope_similarity` commence par initialiser des compteurs pour les différentes caractéristiques de pente : `matching_slopeglo_count`, `matching_slopeloc_count`, et `matching_slope_count`. Elle calcule également le nombre total de tokens dans les phrases gold et auto en utilisant la fonction `count_total_tokens`. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si le token n'est pas une ponctuation, elle extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, pour chaque caractéristique de pente (`SlopeGlo`, `SlopeLoc`, et `Slope`), elle vérifie si les valeurs de la caractéristique dans les tokens gold et auto correspondent. Si elles correspondent, elle incrémente le compteur correspondant. Enfin, la fonction retourne un dictionnaire contenant les comptages de correspondance pour chaque caractéristique de pente et le nombre total de tokens.

16. Fonction `calculate_percentage`

La fonction `calculate_percentage` calcule le pourcentage d'une partie par rapport à un tout. Si le tout (`whole`) est différent de zéro, elle retourne le résultat arrondi à deux décimales. Sinon, elle retourne zéro pour éviter une division par zéro.

17. Fonction `format_count_and_percentage`

La fonction `format_count_and_percentage` prend un compte et un pourcentage, et les formate dans une chaîne de caractères avec le compte suivi du pourcentage entre parenthèses.

18. Fonction `convert_defaultdict_to_dict`

La fonction `convert_defaultdict_to_dict` convertit un `defaultdict` en un dictionnaire régulier, en utilisant le constructeur `dict`.

19. Fonction `sum_counters`

La fonction `sum_counters` prend une liste de compteurs (`Counter`) et les additionne pour produire un compteur total, en utilisant la méthode `update` pour mettre à jour le compteur total avec les valeurs de chaque compteur dans la liste.

20. Fonction `count_silence_tokens`

La fonction `count_silence_tokens` compte le nombre total de tokens de silence dans les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto) séparément.

La fonction `count_silence_tokens` commence par initialiser deux compteurs, `total_silence_gold` et `total_silence_auto`, à zéro. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si un token dans les phrases gold a une étiquette de ponctuation (`'PUNCT'` dans `tag`) et contient le caractère "#" dans le texte (`'t'`), le compteur `total_silence_gold` est incrémenté. De même, si un token dans les phrases auto a une étiquette de ponctuation et contient le caractère "#", le compteur `total_silence_auto` est incrémenté. Enfin, la fonction retourne les deux compteurs, indiquant le nombre total de tokens de silence dans les phrases gold et auto respectivement.

21. Fonction `count_matching_silence_alignment`

La fonction `count_matching_silence_alignment` compte le nombre de tokens de silence ayant des caractéristiques d'alignement correspondantes dans les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto).

La fonction `count_matching_silence_alignment` commence par initialiser un compteur `matching_silence_alignment_count` à zéro et calcule le nombre total de tokens de silence dans les phrases gold et auto en utilisant la fonction `count_silence_tokens`. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si un token dans les phrases gold ou auto a une étiquette de ponctuation (`'PUNCT'` dans `tag`) et contient le caractère "#" dans le texte (`'t'`), la fonction extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, elle obtient les caractéristiques d'alignement `AlignBegin` et `AlignEnd` pour les tokens gold et auto. Si les valeurs de `AlignBegin` et `AlignEnd` pour les deux tokens sont égales, le compteur `matching_silence_alignment_count` est incrémenté. Enfin, la fonction retourne un dictionnaire contenant le nombre de tokens de silence avec des caractéristiques d'alignement correspondantes (`MatchingSilenceAlignmentCount`), ainsi que le nombre total de tokens de silence dans les phrases gold (`TotalSilenceTokensGold`) et auto (`TotalSilenceTokensAuto`).

22. Fonction `count_silence_alignment_duration`

La fonction `count_silence_alignment_duration` compte la durée d'alignement des tokens de silence entre les phrases annotées manuellement (gold) et les phrases annotées automatiquement (auto).

La fonction `count_silence_alignment_duration` initialise des dictionnaires pour stocker les comptages de la durée d'alignement des tokens de silence par intervalles de 10 ms allant de 0 à 700 ms, pour les phrases gold (`gold_counts`) et auto (`auto_counts`). Elle calcule également le nombre total de tokens de silence dans les phrases gold et auto en utilisant la fonction `count_silence_tokens`. Pour chaque paire de phrases gold et auto, elle itère sur les tokens correspondants. Si un token contient le caractère "#" dans le texte (`'t'`), elle extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires via `build_feature_dico`. Ensuite, elle obtient les caractéristiques d'alignement `AlignBegin` et `AlignEnd` pour les tokens gold et auto, en les convertissant en entiers si elles ne sont pas "X". Si les valeurs d'alignement sont valides, elle calcule la durée d'alignement pour les tokens gold et auto. Pour chaque durée calculée, elle incrémente le compteur correspondant dans le dictionnaire `gold_counts` ou `auto_counts` en fonction de l'intervalle auquel la durée appartient. Enfin, la fonction retourne un dictionnaire contenant les comptages de la durée d'alignement par intervalle pour les phrases gold (`GoldCountsByInterval`) et auto (`AutoCountsByInterval`), ainsi que le nombre total de tokens de silence dans les phrases gold (`TotalSilenceTokensGold`) et auto (`TotalSilenceTokensAuto`).

23. Fonction `plot_features`

La fonction `plot_features` trace un graphique en barres pour les caractéristiques d'un DataFrame donné. Elle commence par filtrer les lignes où les valeurs de `Comptage Gold` ou `Comptage Auto` sont supérieures à 150, puis elle trace un graphique en utilisant ces valeurs. Le graphique est enregistré sous forme de fichier PNG et affiché.

24. Fonction `update_global_counts`

La fonction `update_global_counts` met à jour un dictionnaire de comptages globaux en ajoutant les valeurs d'un DataFrame de caractéristiques. Pour chaque ligne du DataFrame, elle incrémente les comptages globaux correspondants.

25. Fonction `update_global_counts_mono`

La fonction `update_global_counts_mono` est similaire mais traite un dictionnaire imbriqué pour les formes monosyllabiques, en s'assurant que les clés existent avant d'ajouter les valeurs.

26. Fonction `convert_monosyllableform_defaultdict_to_df`

La fonction `convert_monosyllableform_defaultdict_to_df` convertit un dictionnaire imbriqué en un DataFrame, en créant une liste de dictionnaires contenant les tokens, les formes et les comptages, puis en convertissant cette liste en DataFrame.

27. Fonction `convert_dict_to_df`

La fonction `convert_dict_to_df` convertit un dictionnaire en DataFrame, en utilisant les clés du dictionnaire comme index et les valeurs comme colonnes, puis en réinitialisant l'index.La fonction `convert_dict_to_df` convertit un dictionnaire en DataFrame, en utilisant les clés du dictionnaire comme index et les valeurs comme colonnes, puis en réinitialisant l'index.

28. Fonction `save_html`

La fonction `save_html` enregistre une liste de DataFrames en tant que fichier HTML, ajoutant des titres pour chaque DataFrame. Elle convertit ensuite le fichier HTML en PDF, bien que cette conversion soit commentée dans le code fourni.

29. Fonction `main`

La fonction principale du script, main, traite des fichiers annotés manuellement (gold) et des fichiers annotés automatiquement (auto) pour extraire et comparer diverses caractéristiques prosodiques. Il commence par initialiser un dictionnaire `counts_dicts` pour stocker les compteurs globaux de diverses caractéristiques prosodiques et de silence. Chaque compteur est initialisé avec une valeur par défaut appropriée. Ensuite, pour chaque fichier gold, il cherche le fichier auto correspondant. Si un fichier correspondant est trouvé, il extrait les arbres syntaxiques et les silences des fichiers gold et auto. Pour chaque paire de fichiers, le script utilise plusieurs fonctions auxiliaires pour compter les caractéristiques suivantes : nombre de syllabes correspondantes, formes de syllabes correspondantes, caractéristiques d'alignement des tokens, similitudes des caractéristiques SLAM, similitudes des caractéristiques de pente, différences d'alignement, correspondances des silences et durée d'alignement des silences. Les résultats de correspondance pour chaque fichier traité sont stockés dans une liste `results`, avec les pourcentages de correspondance et les totaux pour chaque caractéristique. Le script met ensuite à jour les compteurs globaux pour chaque caractéristique, en s'assurant que les valeurs sont correctement agrégées. Enfin, il génère des rapports et des graphiques pour visualiser les résultats, en enregistrant les données sous forme de fichiers HTML et PDF. Cette méthode permet de comparer précisément les annotations manuelles et automatiques, en fournissant des informations détaillées sur la précision et la cohérence des annotations prosodiques.

30. Appel de la fonction main

Le script appelle la fonction main avec les fichiers d'entrée CoNLL-U et les dossiers de sortie.

2. *aux_verb_distribution_gold_auto.ipynb*

1. Importation des modules

- **Importation des modules nécessaires** : *os* et *glob* pour la gestion des fichiers et répertoires, *pandas* pour la manipulation des données, *conll3* pour l'utilisation des fonctions spécifiques aux fichiers CoNLL-U, *defaultdict* et *Counter* pour les structures de données utiles, *HTML* et *display* pour afficher les résultats de manière interactive, *tqdm* pour afficher des barres de progression, *matplotlib.pyplot* pour la création de graphiques, *pdfkit* pour convertir des fichiers en PDF.
- **Configuration de Matplotlib**: Le style '*ggplot*' est utilisé pour les graphiques.

2. Fonction `dico2FeatureString`

La fonction `dico2FeatureString` prend en entrée un dictionnaire `dico` contenant des caractéristiques sous forme de paires clé-valeur. Elle crée une liste de chaînes où chaque chaîne est une paire clé-valeur du dictionnaire, jointe par un signe égal (`=`). Cette liste est ensuite jointe par des barres verticales (`|`) pour former une seule chaîne de caractères qui représente toutes les caractéristiques du dictionnaire. Cette chaîne est alors retournée par la fonction. En somme, cette fonction convertit les entrées du dictionnaire en une représentation textuelle standardisée utilisée dans les fichiers CONLLU pour décrire des caractéristiques supplémentaires des tokens.

3. Fonction `build_feature_dico`

La fonction `build_feature_dico` prend en entrée une chaîne de caractères `misc_features_string` qui contient des caractéristiques CONLLU sous forme de paires clé-valeur séparées par des barres verticales (`|`). Elle initialise un dictionnaire vide `feature_dico`. Ensuite, elle divise la chaîne d'entrée en plusieurs sous-chaînes en utilisant la barre verticale comme séparateur. Pour chaque sous-chaîne résultante, la fonction divise à nouveau la sous-chaîne en une clé et une valeur en utilisant le signe égal (`=`) comme séparateur. Ces paires clé-valeur sont ajoutées au dictionnaire `feature_dico`. La fonction retourne ensuite ce dictionnaire, permettant d'accéder facilement aux caractéristiques individuelles du token.

4. Fonction `extract_trees_and_metadata`

La fonction `extract_trees_and_metadata` extrait les arbres syntaxiques et les métadonnées d'un fichier CoNLL-U.

La fonction `extract_trees_and_metadata` lit un fichier CONLL et extrait les phrases sous forme d'arbres syntaxiques ainsi que les métadonnées associées. Elle initialise des structures de données pour stocker les métadonnées actuelles (`current_metadata`), la phrase en cours de traitement (`current_sentence`) et toutes les phrases (`sentences`). En ouvrant le fichier, la fonction lit chaque ligne et la nettoie des espaces superflus. Si une ligne commence par un caractère "#", elle est considérée comme une métadonnée. Les métadonnées sont stockées dans un dictionnaire où les clés et les valeurs sont séparées par le signe "=". Si une ligne contient un token, les informations du token sont extraites et stockées dans un dictionnaire `current_token`. Si le token n'est pas une ponctuation, il est ajouté à la liste `current_sentence`. Lorsque la fonction rencontre une ligne vide, cela indique la fin d'une phrase. La phrase en cours est alors ajoutée à la liste `sentences`, et `current_sentence` est réinitialisée pour traiter la phrase suivante. À la fin du fichier, toute phrase restante est ajoutée à la liste `sentences`. La fonction retourne finalement la liste des phrases extraites.

5. Fonction `SLAM_mono`

La fonction `SLAM_mono` extrait les informations prosodiques (SLAM) pour les verbes et auxiliaires monosyllabiques dans les phrases annotées manuellement (gold) et automatiquement (auto).

La fonction `SLAM_mono` prend en entrée deux listes de phrases annotées, `gold_sentences` et `auto_sentences`, et un argument optionnel `token_form` pour spécifier la forme du token à analyser. Elle initialise plusieurs dictionnaires `defaultdict` pour stocker les comptages des caractéristiques prosodiques globales et locales telles que `AvgHeightGlo`, `AvgHeightLoc`, `PitchRangeGlo`, `PitchRangeLoc`, `Glo`, et `Loc`, en distinguant les annotations gold et auto. Les compteurs totaux pour les verbes et auxiliaires monosyllabiques sont également initialisés à zéro.

Pour chaque paire de phrases gold et auto, la fonction itère sur les tokens correspondants. Si les tokens ne sont pas des ponctuations et qu'ils sont tous deux soit des auxiliaires soit des verbes ayant la même forme (`t`), la fonction extrait les caractéristiques `misc` des tokens et les convertit en dictionnaires à l'aide de `build_feature_dico`. Si le paramètre `token_form` est spécifié, la fonction ne considère que les tokens ayant cette forme spécifique. Elle vérifie ensuite si les caractéristiques d'étiquettes pertinentes (comme `Syl1AvgHeightGlo`, `Syl1AvgHeightLoc`, etc.) sont présentes dans les dictionnaires de caractéristiques des tokens gold et auto.

Si le token gold est monosyllabique (`SyllableCount` égale à 1), la fonction incrémente les compteurs globaux et locaux appropriés pour les verbes ou auxiliaires dans les annotations gold, en fonction des valeurs des caractéristiques extraites. De même, elle incrémente les compteurs correspondants pour les annotations auto. La fonction continue ce processus pour tous les tokens des phrases. Enfin, elle retourne les dictionnaires de comptages des caractéristiques prosodiques ainsi que les compteurs totaux de verbes et auxiliaires monosyllabiques dans les annotations gold et auto.

Cette méthode permet de comparer les annotations manuelles et automatiques pour des caractéristiques prosodiques spécifiques, fournissant des informations détaillées sur la précision et la cohérence des annotations.

6. Fonction `Slope_mono`

La fonction `Slope_mono` extrait les informations de pente (slope) pour les verbes et auxiliaires monosyllabiques dans les phrases annotées manuellement (gold) et automatiquement (auto).

La fonction `Slope_mono` prend en entrée deux listes de phrases annotées, `gold_sentences` et `auto_sentences`, ainsi qu'un argument optionnel `token_form` pour spécifier la forme du token à analyser. Elle initialise plusieurs dictionnaires `defaultdict` pour stocker les comptages des caractéristiques de pente globales et locales (`SlopeGlo`, `SlopeLoc`, `Slope`), en distinguant les annotations gold et auto. Les compteurs totaux pour les verbes et auxiliaires monosyllabiques sont également initialisés à zéro.

Pour chaque paire de phrases gold et auto, la fonction itère sur les tokens correspondants. Si les tokens ne sont pas des ponctuations et qu'ils sont tous deux soit des auxiliaires soit des verbes ayant la même forme (`t`), la fonction vérifie si le paramètre `token_form` est spécifié. Si c'est le cas, elle ne considère que les tokens ayant cette forme spécifique. Elle extrait ensuite les caractéristiques `misc` des tokens et les convertit en dictionnaires à l'aide de `build_feature_dico`. La fonction vérifie la présence des étiquettes pertinentes (comme `Syl1SlopeGlo`, `Syl1SlopeLoc`, `Syl1Slope`) dans les dictionnaires de caractéristiques des tokens gold et auto.

Si le token gold est monosyllabique (`SyllableCount` égale à 1), la fonction incrémente les compteurs globaux et locaux appropriés pour les verbes ou auxiliaires dans les annotations gold, en fonction des valeurs des caractéristiques extraites. De même, elle incrémente les compteurs correspondants pour les annotations auto. La fonction continue ce processus pour tous les tokens des phrases.

Enfin, la fonction retourne les dictionnaires de comptages des caractéristiques de pente ainsi que les compteurs totaux de verbes et auxiliaires monosyllabiques dans les annotations gold et auto.

7. Fonction `sum_counters`

La fonction `sum_counters` additionne une liste de compteurs (`Counter`) pour produire un compteur total, en utilisant la méthode `update` pour ajouter les valeurs de chaque compteur à un compteur global.

8. Fonction `convert_defaultdict_to_dict`

La fonction `convert_defaultdict_to_dict` convertit un `defaultdict` imbriqué en un dictionnaire régulier en utilisant une compréhension de dictionnaire, ce qui permet de manipuler plus facilement les données dans des structures de données standards.

9. Fonction `convert_nested_dict_to_df`

La fonction `convert_nested_dict_to_df` convertit un dictionnaire imbriqué en un DataFrame, combinant les comptages pour les formes de tokens pour les verbes (`VERB`) et auxiliaires (`AUX`) sur la même ligne pour chaque forme. Elle initialise les données pour chaque forme et met à jour les comptages en fonction du type de POS (partie du discours).

10. Fonction `convert_nested_dict_to_df_with_percent`

La fonction `convert_nested_dict_to_df_with_percent` est similaire, mais elle ajoute des pourcentages pour chaque comptage en fonction des totaux globaux des verbes et auxiliaires dans les annotations gold et auto.

11. Fonction `convert_dict_to_df`

La fonction `convert_dict_to_df` convertit un dictionnaire en DataFrame en utilisant les clés comme index et les valeurs comme colonnes, puis réinitialise l'index pour une manipulation plus facile.

12. Fonction `plot_features`

La fonction `plot_features` trace les caractéristiques d'un DataFrame donné, en séparant visuellement les comptages pour les verbes (`VERB`) et auxiliaires (`AUX`). Elle filtre les données pour ne conserver que les valeurs de comptage supérieures à une limite spécifiée (`limite`). Les barres pour les verbes et auxiliaires sont tracées côte à côte, avec des largeurs de barres et des positions ajustées pour une séparation claire. Les étiquettes sont ajoutées aux barres pour afficher les valeurs de comptage et les formes de tokens. La fonction ajuste les positions des étiquettes x pour être au milieu des groupes de barres et enregistre le graphique sous forme de fichier PNG.

13. Fonction `global_result`

La fonction `global_result`, calcule les résultats globaux pour les caractéristiques SLAM et Slope des verbes et auxiliaires monosyllabiques dans les phrases annotées manuellement (gold) et automatiquement (auto).

La fonction `global_result` prend en entrée deux listes de fichiers annotés (gold et auto) et calcule les résultats globaux pour les caractéristiques SLAM et Slope des verbes et auxiliaires monosyllabiques. Elle initialise plusieurs dictionnaires `defaultdict` pour stocker les comptages globaux des caractéristiques `AvgHeightGlo`, `AvgHeightLoc`, `PitchRangeGlo`, `PitchRangeLoc`, `Glo`, `Loc`, `SlopeGlo`, `SlopeLoc`, et `Slope`, en distinguant les annotations gold et auto. Les compteurs totaux pour les verbes et auxiliaires monosyllabiques sont également initialisés à zéro.

Pour chaque fichier gold, la fonction recherche le fichier auto correspondant et extrait les phrases annotées pour les deux fichiers. Ensuite, elle utilise les fonctions `SLAM_mono` et `Slope_mono` pour calculer les caractéristiques SLAM et Slope pour les verbes et auxiliaires monosyllabiques dans les phrases. Les résultats sont ajoutés aux compteurs globaux, en incrémentant les valeurs appropriées pour chaque caractéristique et chaque type de partie du discours (verbe ou auxiliaire).

Les totaux globaux pour les verbes et auxiliaires monosyllabiques dans les annotations gold et auto sont également mis à jour. Une fois toutes les paires de fichiers traitées, la fonction convertit les dictionnaires de comptages en DataFrames en ajoutant les pourcentages correspondants pour chaque comptage en fonction des totaux globaux. Ces DataFrames sont ensuite enregistrés sous forme de fichiers CSV.

Enfin, la fonction trace les caractéristiques en utilisant la fonction `plot_features`, qui génère des graphiques pour visualiser les comptages des caractéristiques SLAM et Slope. Les graphiques sont enregistrés sous forme de fichiers PNG.

14. Fonction `global_go_result`

La fonction `global_go_result` calcule les résultats globaux pour les caractéristiques SLAM et Slope des verbes et auxiliaires monosyllabiques ayant la forme "go" dans les phrases annotées manuellement (gold) et automatiquement (auto).

La fonction `global_go_result` prend en entrée deux listes de fichiers annotés (gold et auto) et calcule les résultats globaux pour les caractéristiques SLAM et Slope des verbes et auxiliaires monosyllabiques ayant la forme "go". Elle initialise plusieurs dictionnaires `defaultdict` pour stocker les comptages globaux des caractéristiques `AvgHeightGlo`, `AvgHeightLoc`, `PitchRangeGlo`, `PitchRangeLoc`, `Glo`, `Loc`, `SlopeGlo`, `SlopeLoc`, et `Slope`, en distinguant les annotations gold et auto. Les compteurs totaux pour les verbes et auxiliaires monosyllabiques sont également initialisés à zéro.

Pour chaque fichier gold, la fonction recherche le fichier auto correspondant et extrait les phrases annotées pour les deux fichiers. Ensuite, elle utilise les fonctions `SLAM_mono` et `Slope_mono` pour calculer les caractéristiques SLAM et Slope pour les verbes et auxiliaires monosyllabiques ayant la forme "go" dans les phrases. Les résultats sont ajoutés aux compteurs globaux, en incrémentant les valeurs appropriées pour chaque caractéristique et chaque type de partie du discours (verbe ou auxiliaire).

Les totaux globaux pour les verbes et auxiliaires monosyllabiques ayant la forme "go" dans les annotations gold et auto sont également mis à jour. Une fois toutes les paires de fichiers traitées, la fonction convertit les dictionnaires de comptages en DataFrames en ajoutant les pourcentages correspondants pour chaque comptage en fonction des totaux globaux. Ces DataFrames sont ensuite enregistrés sous forme de fichiers CSV.

Enfin, la fonction trace les caractéristiques en utilisant la fonction `plot_features`, qui génère des graphiques pour visualiser les comptages des caractéristiques SLAM et Slope. Les graphiques sont enregistrés sous forme de fichiers PNG.

15. Fonction `aux_mono_slam`

La fonction `aux_mono_slam` récupère les caractéristiques SLAM pour les auxiliaires monosyllabiques.

Comme précédemment, la fonction `aux_mono_slam` prend en entrée deux listes de phrases annotées, `gold_sentences` et `auto_sentences`, et extrait les caractéristiques SLAM pour les auxiliaires monosyllabiques. Elle initialise plusieurs dictionnaires `defaultdict` pour stocker les comptages des caractéristiques `AvgHeightGlo`, `AvgHeightLoc`, `PitchRangeGlo`, `PitchRangeLoc`, `Glo`, et `Loc`, en distinguant les annotations gold et auto. Pour chaque paire de phrases dans les listes `gold_sentences` et `auto_sentences`, la fonction vérifie d'abord si les phrases sont identiques en termes de texte en les convertissant en minuscules et en les comparant. Si les phrases sont identiques, elle procède à l'analyse des tokens individuels.

Pour chaque token, si le token est un auxiliaire (tag `AUX`) et n'est pas une ponctuation (`PUNCT`), la fonction extrait les caractéristiques SLAM du champ `misc` du token. Si le token est monosyllabique (indiqué par `SyllableCount` égal à `1`), la fonction met à jour les comptages des caractéristiques correspondantes dans les dictionnaires `defaultdict`. Cela est fait séparément pour les annotations gold et auto. Les caractéristiques prises en compte incluent `Syl1AvgHeightGlo`, `Syl1AvgHeightLoc`, `Syl1PitchRangeGlo`, `Syl1PitchRangeLoc`, `Syl1Glo`, et `Syl1Loc`.

Une fois toutes les paires de phrases analysées, la fonction retourne les dictionnaires de comptages pour chaque caractéristique SLAM.

16. Fonction `aux_mono_slope`

La fonction `aux_mono_slope` est semblable à la précédente mais extrait les caractéristiques de la pente (Slope) pour les auxiliaires monosyllabiques.

La fonction `aux_mono_slope` prend en entrée deux listes de phrases annotées, `gold_sentences` et `auto_sentences`, et extrait les caractéristiques de la pente (Slope) pour les auxiliaires monosyllabiques. Elle initialise plusieurs dictionnaires `defaultdict` pour stocker les comptages des caractéristiques `SlopeGlo`, `SlopeLoc`, et `Slope`, en distinguant les annotations gold et auto. Pour chaque paire de phrases dans les listes `gold_sentences` et `auto_sentences`, la fonction vérifie d'abord si les phrases sont identiques en termes de texte en les convertissant en minuscules et en les comparant. Si les phrases sont identiques, elle procède à l'analyse des tokens individuels.

Pour chaque token, si le token est un auxiliaire (tag `AUX`) et n'est pas une ponctuation (`PUNCT`), la fonction extrait les caractéristiques de la pente du champ `misc` du token. Si le token est monosyllabique (indiqué par `SyllableCount` égal à `1`), la fonction met à jour les comptages des caractéristiques correspondantes dans les dictionnaires `defaultdict`. Cela est fait séparément pour les annotations gold et auto. Les caractéristiques prises en compte incluent `Syl1SlopeGlo`, `Syl1SlopeLoc`, et `Syl1Slope`.

Une fois toutes les paires de phrases analysées, la fonction retourne les dictionnaires de comptages pour chaque caractéristique de la pente.

17. Fonction `convert_mono_defaultdict_to_df`

La fonction `convert_mono_defaultdict_to_df` prend en paramètre un dictionnaire imbriqué de type `defaultdict` et le convertit en un `DataFrame` Pandas. Ce dictionnaire contient des informations sur les caractéristiques des tokens (mots) dans les phrases annotées. Chaque clé du dictionnaire externe représente un token, et chaque clé du dictionnaire interne représente une forme du token (une variante spécifique). Les valeurs sont des sous-dictionnaires qui contiennent les comptages de la caractéristique pour les annotations gold et auto. La fonction parcourt toutes les entrées du dictionnaire imbriqué, extrait les informations pertinentes (token, forme, comptages gold et auto), et les ajoute à une liste sous forme de dictionnaires. Cette liste est ensuite convertie en un `DataFrame` Pandas, qui est retourné par la fonction.

18. Fonction `convert_mono_defaultdict_to_df_with_percent`

La fonction `convert_mono_defaultdict_to_df_with_percent` convertit également un dictionnaire imbriqué de type `defaultdict` en un `DataFrame` Pandas, mais elle inclut en plus le pourcentage des comptages par rapport au total. Les paramètres incluent le dictionnaire imbriqué et un tuple contenant les totaux des annotations gold et auto. La fonction parcourt chaque entrée du dictionnaire imbriqué et calcule les pourcentages pour les comptages gold et auto en fonction des totaux respectifs. Ces valeurs, combinées avec les comptages absolus, sont ajoutées à une liste sous forme de dictionnaires, qui est ensuite convertie en un `DataFrame`. Ce `DataFrame` est retourné par la fonction.

19. Fonction `plot_token`

La fonction `plot_token` génère des graphiques à barres pour les caractéristiques de chaque token et les enregistre sous forme de fichiers PNG. Les paramètres incluent un `DataFrame`, le nom de la caractéristique, le dossier de sortie pour les fichiers PNG, la longueur de l'axe x, et une limite pour filtrer les caractéristiques à afficher. La fonction crée une colonne combinée `Token_Form` dans le `DataFrame`, filtre les entrées selon la limite spécifiée, et génère un graphique à barres pour les comptages gold et auto. Les barres sont annotées avec leurs valeurs, et le graphique est sauvegardé sous forme de fichier PNG dans le dossier de sortie.

20. Fonction `most_common_form`

La fonction `most_common_form` identifie la forme la plus courante pour chaque token dans un `DataFrame`. Elle regroupe les données par token et trouve les entrées avec les comptages maximaux pour les annotations gold et auto. Les résultats sont renommés pour préparer la fusion, et les deux `DataFrame` (pour gold et auto) sont fusionnés sur la colonne `Token`. Le tableau final, contenant les formes les plus courantes pour chaque token avec leurs comptages respectifs, est retourné par la fonction.

21. Fonction `aux_result`

La fonction `aux_result` calcule les résultats globaux pour les caractéristiques SLAM et Slope des auxiliaires monosyllabiques dans les phrases annotées manuellement (gold) et automatiquement (auto).

La fonction `aux_result` prend en entrée deux listes de fichiers de phrases annotées, `gold_files` et `auto_files`, et calcule les caractéristiques SLAM et Slope pour les auxiliaires monosyllabiques. Elle initialise plusieurs dictionnaires `defaultdict` pour stocker les comptages des différentes caractéristiques, telles que `AvgHeightGlo`, `AvgHeightLoc`, `PitchRangeGlo`, `PitchRangeLoc`, `Glo`, `Loc`, `SlopeGlo`, `SlopeLoc`, et `Slope`, séparément pour les annotations gold et auto.

Pour chaque fichier dans `gold_files`, la fonction extrait les phrases annotées et trouve le fichier correspondant dans `auto_files` basé sur le nom du fichier. Ensuite, elle appelle les fonctions `aux_mono_slam` et `aux_mono_slope` pour extraire les caractéristiques SLAM et Slope des phrases correspondantes. Les résultats de ces fonctions sont utilisés pour mettre à jour les dictionnaires globaux de comptage des caractéristiques.

Une fois que toutes les phrases ont été analysées, la fonction convertit les dictionnaires `defaultdict` en DataFrames. Chaque DataFrame contient les comptages des caractéristiques pour chaque forme de token, et les résultats les plus fréquents pour chaque token sont sauvegardés dans des fichiers CSV. Les DataFrames sont ensuite utilisés pour générer des graphiques des caractéristiques, qui sont sauvegardés sous forme de fichiers PNG.

La fonction utilise des sous-fonctions comme `convert_mono_defaultdict_to_df` et `convert_mono_defaultdict_to_df_with_percent` pour convertir les dictionnaires en DataFrames, et `plot_token` pour tracer les graphiques des caractéristiques. Elle utilise également la fonction `most_common_form` pour trouver les formes les plus fréquentes pour chaque token.

22. Fonction `SLAM_mono_aux_verb`

La fonction `SLAM_mono_aux_verb` a pour but d'extraire les caractéristiques SLAM (Syl1AvgHeightGlo, Syl1AvgHeightLoc, Syl1PitchRangeGlo, Syl1PitchRangeLoc, Syl1Glo, Syl1Loc) pour les verbes monosyllabiques et les auxiliaires dans des phrases annotées, appelées respectivement `gold_sentences` et `auto_sentences`. Ces caractéristiques sont calculées uniquement lorsque le verbe est précédé par un auxiliaire (AUX -> VERB) ou un autre verbe (VERB -> VERB). Les résultats sont retournés sous forme de dictionnaires imbriqués, indiquant le nombre de fois où chaque caractéristique apparaît dans les phrases annotées (gold et auto).

La fonction commence par initialiser plusieurs dictionnaires de comptage imbriqués (`defaultdict`), chacun étant configuré pour stocker les caractéristiques SLAM spécifiques des tokens, à la fois pour les annotations "Gold" et "Auto". Les variables `total_g_sent` et `total_a_sent` comptent le nombre total de paires de tokens (gold et auto) traitées.

Ensuite, la fonction boucle à travers chaque paire de phrases annotées (gold et auto). Pour chaque paire de phrases, elle vérifie si les phrases sont identiques en termes de texte (indépendamment de la casse). Si les phrases sont identiques, elle procède à une itération sur chaque paire de tokens adjacents dans les phrases (à l'exclusion des tokens de ponctuation). Pour chaque paire de tokens adjacents, elle vérifie si le premier token est un verbe ou un auxiliaire et si le deuxième token est un verbe. Si c'est le cas, elle extrait les caractéristiques `misc` de chaque token et les transforme en dictionnaires de caractéristiques.

La fonction vérifie ensuite si toutes les caractéristiques requises sont présentes pour les deux tokens dans les annotations gold et auto. Si les tokens et leurs caractéristiques correspondent, elle augmente les compteurs pour chaque caractéristique SLAM dans les dictionnaires correspondants. Elle distingue également les paires VERB->VERB et AUX->VERB pour les annotations gold et auto.

Enfin, la fonction retourne les dictionnaires de comptage remplis, ainsi que les comptes totaux pour les paires VERB->VERB et AUX->VERB dans les annotations gold et auto. Ces résultats permettent d'analyser et de comparer les caractéristiques prosodiques des verbes et auxiliaires monosyllabiques dans une structure particulière dans les phrases annotées.

23. Fonction `Slope_mono_aux_verb`

Similaire à SLAM_mono_aux_verb, cette fonction se concentre sur l'extraction des informations de pente pour les verbes monosyllabiques précédés par un auxiliaire ou un autre verbe.

La fonction `Slope_mono_aux_verb` a pour objectif d'extraire les caractéristiques de pente (Slope) pour les auxiliaires et les verbes monosyllabiques dans les phrases annotées, appelées respectivement `gold_sentences` et `auto_sentences`. Cette extraction se fait uniquement lorsque le verbe est précédé par un auxiliaire (AUX -> VERB) ou un autre verbe (VERB -> VERB). Les caractéristiques de pente considérées sont `Syl1SlopeGlo`, `Syl1SlopeLoc` et `Syl1Slope`. Les résultats sont retournés sous forme de dictionnaires imbriqués, indiquant le nombre de fois où chaque caractéristique apparaît dans les phrases annotées (gold et auto).

La fonction commence par initialiser plusieurs dictionnaires de comptage imbriqués (`defaultdict`), chacun étant configuré pour stocker les caractéristiques de pente spécifiques des tokens, à la fois pour les annotations "Gold" et "Auto". Les variables `total_g_sent` et `total_a_sent` comptent le nombre total de paires de tokens (gold et auto) traitées.

Ensuite, la fonction boucle à travers chaque paire de phrases annotées (gold et auto). Pour chaque paire de phrases, elle vérifie si les phrases sont identiques en termes de texte (indépendamment de la casse). Si les phrases sont identiques, elle procède à une itération sur chaque paire de tokens adjacents dans les phrases (à l'exclusion des tokens de ponctuation). Pour chaque paire de tokens adjacents, elle vérifie si le premier token est un verbe ou un auxiliaire et si le deuxième token est un verbe. Si c'est le cas, elle extrait les caractéristiques `misc` de chaque token et les transforme en dictionnaires de caractéristiques.

La fonction vérifie ensuite si toutes les caractéristiques requises sont présentes pour les deux tokens dans les annotations gold et auto. Si les tokens et leurs caractéristiques correspondent, elle augmente les compteurs pour chaque caractéristique de pente dans les dictionnaires correspondants. Elle distingue également les paires VERB->VERB et AUX->VERB pour les annotations gold et auto.

Enfin, la fonction retourne les dictionnaires de comptage remplis, ainsi que les comptes totaux pour les paires VERB->VERB et AUX->VERB dans les annotations gold et auto.

24. Fonction `plot_aux_verb`

La fonction `plot_aux_verb` génère des graphiques en barres pour visualiser les caractéristiques des verbes et des auxiliaires monosyllabiques présents dans un DataFrame. Cette fonction prend en paramètres un DataFrame (`df`), le nom de la caractéristique (`feature_name`), le dossier de sortie pour les fichiers PNG (`png_outfolder`), une limite pour filtrer les caractéristiques (`limite`) et la longueur de l'axe des x (`x_length`). Le DataFrame est filtré pour ne conserver que les lignes où les comptes sont supérieurs à la limite spécifiée. La fonction crée ensuite des graphiques en barres pour les comptes "Gold" et "Auto" des paires VERB-VERB et AUX-VERB, en les séparant visuellement pour faciliter la comparaison. Les barres sont annotées avec leurs valeurs respectives, et le graphique final est enregistré en tant que fichier PNG dans le dossier spécifié.

25. Fonction `convert_mono_aux_verb_defaultdict_to_df`

La fonction `convert_mono_aux_verb_defaultdict_to_df` convertit un `defaultdict` imbriqué en DataFrame pour les verbes et les auxiliaires monosyllabiques. Le `defaultdict` contient des comptes de caractéristiques pour différentes paires de tokens (VERB-VERB et AUX-VERB). La fonction itère à travers chaque paire de tokens dans le `defaultdict`, construisant un dictionnaire de données où chaque entrée représente une forme spécifique avec ses comptes "Gold" et "Auto" pour les paires VERB-VERB et AUX-VERB. Ce dictionnaire est ensuite converti en DataFrame, qui peut être utilisé pour des analyses ou des visualisations ultérieures.

26. Fonction `convert_mono_aux_verb_defaultdict_to_df_with_percent`

La fonction `convert_mono_aux_verb_defaultdict_to_df_with_percent` est similaire à la fonction précédente, mais elle ajoute des pourcentages aux comptes des caractéristiques. Elle prend en paramètres un `defaultdict` imbriqué contenant les comptes de caractéristiques et un tuple contenant les comptes totaux pour les paires VERB-VERB et AUX-VERB (gold et auto). La fonction itère à travers chaque paire de tokens dans le `defaultdict`, calculant les pourcentages correspondants pour chaque forme. Ces pourcentages sont ajoutés aux comptes "Gold" et "Auto" dans le dictionnaire de données. Le dictionnaire est ensuite converti en DataFrame, permettant une analyse plus détaillée des proportions des caractéristiques pour chaque forme.

27. Fonction `aux_result_aux_verb`

La fonction `aux_result_aux_verb` est conçue pour analyser et comparer les caractéristiques prosodiques de verbes monosyllabiques précédant soit un verbe auxiliaire (AUX -> VERB) soit un autre verbe (VERB -> VERB) dans des phrases annotées manuellement (gold) et automatiquement (auto). Elle prend en paramètres deux listes de fichiers contenant ces annotations, l'une pour les phrases gold et l'autre pour les phrases annotées automatiquement. La fonction commence par initialiser plusieurs dictionnaires à valeurs par défaut (`defaultdict`) pour stocker les comptes de diverses caractéristiques prosodiques, telles que `AvgHeightGlo`, `AvgHeightLoc`, `PitchRangeGlo`, `PitchRangeLoc`, `Glo`, `Loc`, `SlopeGlo`, `SlopeLoc`, et `Slope`.

Ensuite, pour chaque fichier de phrases gold, elle trouve le fichier correspondant dans les phrases automatiques, extrait les arbres et les métadonnées des phrases, et s'assure que les nombres de phrases dans les deux fichiers correspondent. Si le nombre de phrases ne correspond pas, un message d'erreur est affiché. La fonction `SLAM_mono_aux_verb` est utilisée pour extraire les caractéristiques SLAM des phrases, et `Slope_mono_aux_verb` pour extraire les caractéristiques de pente (Slope). Les comptes obtenus sont ensuite ajoutés aux dictionnaires globaux.

La fonction additionne les comptes totaux des caractéristiques SLAM et Slope pour les paires VERB-VERB et AUX-VERB. Elle convertit ensuite les dictionnaires imbriqués en DataFrames, ajoutant des pourcentages aux comptes pour fournir un contexte supplémentaire. Ces DataFrames sont ensuite sauvegardés en fichiers CSV. Enfin, les données sont converties en DataFrames sans pourcentages pour être utilisées dans des graphiques. La fonction `plot_aux_verb` est appelée pour créer et sauvegarder des graphiques illustrant les comptes des différentes caractéristiques prosodiques.