PROJET ARGILES

GUIDE DEVELOPPEUR PYTHON

Les codes soigneusement commentés sont disponibles dans le dossier suivante :

https://github.com/zancap/ARGILES-Database-Management-and-NLP/tree/main/utils

Cette documentation technique est destinée aux développeurs souhaitant comprendre le fonctionnement des scripts suivants :

- get_audite.py
- stats.py
- extraction_info.py

Intégration avec PHP

Dans la guide développeur du site web du projet ARGILES (fournie) vous pouvez trouver toutes les informations nécessaires à la compréhension de l'intégration entre ces scripts python et le site web du projet.

1. Get_audite.py

Ce script permet de

D'extraire et d'afficher les réponses d'un élève spécifique pour un livre donné. Il repose sur le module extraction_info.py, qui assure la lecture et la structuration des fichiers XML. Il est conçu pour être exécuté aussi en ligne de commande avec des arguments, ce qui permet de récupérer les informations sur un élève sans passer par une interface graphique.

Intégration dans une interface web : Ce script est conçu pour être exécuté en ligne de commande mais on a ajouté des pages PHP (décrites dans la documentation PHP) pour qu'il puisse être utilisé via un serveur web et permettre une meilleure accessibilité.

Fonctionnalités principales

- ➔ Récupération des arguments passés en ligne de commande.
- ➔ Chargement des données XML via read_xmls().
- ➔ Vérification de l'existence de l'élève et du livre demandé.
- ➔ Extraction et affichage des réponses de l'élève, si disponibles.

Lecture des arguments

Le script commence par récupérer les arguments fournis lors de l'exécution.

```
args = sys.argv
args = [x.strip() for x in args]  # Nettoyage des arguments
```

Les arguments attendus sont :

- args[1] : ID de l'élève (exemple : 110).
- args[2] : Nom du livre concerné (exemple : "roman Sirius").

- args[3] *(optionnel)* : Chemin vers le dossier contenant les fichiers XML.

Si le chemin vers le dossier XML n'est pas fourni, le script utilise ./xmls/ par défaut :

```
if len(args) >= 4:
    xml_dir = args[3]
else:
    xml_dir = './xmls/'
```

Chargement des fichiers XML

Le script appelle read_xmls(xml_dir) pour extraire les informations contenues dans les fichiers XML.

```
document, groupes, audites = read_xmls(xml_dir)
```

Cela permet de charger trois dictionnaires contenant les documents, groupes et élèves analysés.

Vérification et extraction des réponses

Le script vérifie si l'ID de l'élève fourni est présent dans les données XML :

```
if len(args) >= 3 and args[1] in audites.keys():
```

Si l'ID est valide, il vérifie si l'élève a bien répondu à des questions sur le livre demandé :

```
if args[2] in audites[args[1]]["reponses"].keys():
    output = get_audite(args[1], args[2], xml_dir)
    print(output)
```

Si les réponses existent, elles sont extraites via get_audite() et affichées.

Si l'élève n'a pas répondu pour ce livre, le script affiche une liste des documents auxquels il a répondu :

```
print("Incorrect document\nPossibles documents :")
for key in audites[args[1]]["reponses"].keys():
    print('\t', key)
```

Si l'ID de l'élève est inconnu, un message d'erreur est affiché :

```
else:
    print("Unknown ID")
```

Améliorations possibles

- Gestion des erreurs améliorée : Actuellement, si les arguments ne sont pas bien fournis, le script peut générer des erreurs peu lisibles. Une validation plus stricte des entrées permettrait d'améliorer la robustesse.

2. Stats.py

Ce script est utilisé pour

Extraire et analyser les réponses des élèves sur un texte donné. Il prend en entrée une liste de documents et/ou de groupes d'élèves et génère des statistiques sur leurs réponses. Il repose sur le module extraction_info.py pour la lecture et la structuration des fichiers XML. Il est conçu pour être exécuté en ligne de commande avec des arguments représentant les documents et les groupes à analyser.

Fonctionnalités principales

- ➔ Récupération et nettoyage des arguments.
- ➔ Détection du chemin du répertoire contenant les fichiers XML.
- ➔ Chargement des données XML via read_xmls().
- ➔ Appel à la fonction stats() pour effectuer l'analyse des réponses des élèves.

Lecture des arguments

Le script commence par récupérer les arguments fournis lors de l'exécution et les nettoie en supprimant les espaces superflus.

```
args = sys.argv
args = [x.strip() for x in args]  # Nettoyage des arguments
```

Les arguments attendus sont :

- ❖ Un ou plusieurs noms de documents et/ou de groupes d'élèves (exemples : "roman Sirius", "CM2").
- ❖ *(Optionnel)* Un chemin vers le répertoire contenant les fichiers XML (exemple : "../xmls/").

Détection du répertoire des fichiers XML

Le script vérifie si le dernier argument est un chemin (contient un /). Si c'est le cas, il est utilisé comme répertoire pour les fichiers XML.

```
if '/' in args[-1]:
    xml_dir = args[-1]
    document, groupes, audites = read_xmls(xml_dir)
    stats(args[1:-1])  # Exécute les statistiques sur les autres arguments
```

Sinon, le script utilise le répertoire par défaut ../xmls/ :

```
else:
    xml_dir = '../xmls/'
    document, groupes, audites = read_xmls(xml_dir)
    stats(args[1:])  # Exécute les statistiques avec tous les arguments fournis
```

Cette approche permet au script d'être flexible et de s'adapter aux besoins des utilisateurs sans nécessiter de modification du code.

Chargement des fichiers XML

Le script charge les fichiers XML en appelant read_xmls(xml_dir), ce qui renvoie trois dictionnaires contenant les documents, groupes et élèves analysés.

```
document, groupes, audites = read_xmls(xml_dir)
```

Ces données sont ensuite utilisées pour identifier les réponses des élèves concernées.

Exécution de l'analyse statistique

Une fois les données XML chargées, le script appelle la fonction stats() en lui passant les documents et groupes sélectionnés.

```
stats(args[1:-1])  # Si un chemin XML a été spécifié
stats(args[1:])  # Sinon
```

La fonction stats() est responsable de l'analyse des réponses des élèves :

- Identification des mots les plus fréquents.
- Catégorisation des réponses selon leur valence (positive, négative).
- Comparaison entre différents groupes d'élèves si plusieurs ID sont fournis.

Améliorations possibles

- Validation et gestion des erreurs : Actuellement, si les arguments fournis sont incorrects ou si les fichiers XML ne contiennent pas les informations attendues, le script risque de générer des erreurs difficiles à interpréter. Ajouter une gestion des erreurs plus robuste améliorerait la fiabilité du script.
- Possibilité de choisir différentes méthodes d'analyse : Ajouter des options pour sélectionner le type d'analyse (ex. fréquence des mots, analyse sémantique) permettrait une utilisation plus flexible du script.

3. Extraction_info.py

Avec ce script vous pouvez

- ➔ Charger et analyser des fichiers XML contenant les réponses d'élèves à des questionnaires sur des œuvres littéraires.
- ➔ Organiser les données sous forme de dictionnaires exploitables.
- ➔ Extraire les réponses des élèves en fonction d'un ID spécifique.
- ➔ Générer des statistiques lexicales à partir des réponses collectées.

Le script repose principalement sur xml.etree.ElementTree pour le parsing XML et SpaCy/NLTK pour le traitement linguistique des données textuelles.

Fonctionnalités principales

Le script est organisé autour de 3 grandes fonctions :

1. Lecture des fichiers XML et structuration des données (read_xmls())
2. Extraction des réponses d'un élève spécifique (get_audite())
3. Analyse lexicale des réponses (stats())

Détails des Fonctions

read_xmls(xml_dir = "../xmls/")

Objectif : Charger et structurer les données à partir des fichiers XML.

Entrée : xml_dir *(str)* : Chemin du dossier contenant les fichiers XML.

Sortie : 3 dictionnaires représentant les documents, groupes et audités :

- ❖ document : Informations sur les livres analysés (titre, ID, questions, groupes).
- ❖ groupes : Informations sur les groupes d'élèves (ID, document associé, liste des élèves).

- ❖ audites : Informations détaillées sur chaque élève (ID, groupe, réponses classées par type de question).

Le script : vérifie l'existence du dossier XML; parcourt chaque fichier XML, lit son contenu et extrait les informations; remplit les dictionnaires document, groupes et audites pour structurer les données et enfin retourne ces trois dictionnaires.

Exemple de structure :

```
document = {
   "Sirius": {
      "id": 1,
      "questions": {"evocation": {...}, "like": {...}},
      "groupes": ["Groupe1", "Groupe2"]
   }
}
```

get_audite(id_audite:str, manga:str, xml_dir='./xmls/')

Objectif : Récupérer les réponses d'un élève spécifique pour un livre donné.

Entrée :

- id_audite *(str)* : Identifiant de l'élève.
- manga *(str)* : Nom du livre concerné.
- xml_dir *(str, optionnel)* : Chemin du dossier contenant les fichiers XML.

Sortie : Chaîne de texte formatée contenant les réponses de l'élève, séparées par |.

Le script: charge les données XML via read_xmls(); vérifie si l'ID de l'élève et le livre sont bien présents dans la base de données; extrait les réponses de l'élève et les formate en une chaîne de texte structurée.

Exemple de sortie :

evocation : Aventure mystère | like : J'aime l'univers post-apocalyptique | dislike : Trop de descriptions |

stats(*args)

Objectif : Extraire des statistiques lexicales sur les réponses des élèves.

Entrée : args *(list)* : Liste contenant un document ou un groupe spécifique à analyser.

Sortie : Impression des statistiques lexicales dans le terminal.

Ce script : charge les données XML via read_xmls(); détermine les livres et groupes à analyser; applique un prétraitement NLP sur les réponses des élèves (spacy, nltk); exclut les stopwords pour ne garder que les mots significatifs; trie les mots les plus fréquents et affiche les résultats sous forme tabulaire.

Exemple de sortie dans la console :

--- evocation ---		--- like ---		--- dislike ---		--- expectation ---	
mystère	15	aventure	20	lent	12	plus d'action	10
futur	12	suspense	18	compliqué	8	un autre héros	5

Améliorations possibles

- Gestion des erreurs : Actuellement, certaines erreurs (fichiers mal formatés, valeurs manquantes) sont seulement imprimées dans la console. Une meilleure gestion des exceptions améliorerait la robustesse.
- Optimisation des performances : Le traitement NLP peut être coûteux en ressources. Une option pour stocker les résultats en cache éviterait d'avoir à retraiter les mêmes fichiers à chaque exécution.
- Affichage graphique des résultats : Générer des visualisations pour faciliter l'analyse des données.