Documentation Technique - Projet Histoires

Documentation Technique - Projet Jeux de Mots - Chaines de triplet de narration
==========

Fichier : story_test.py

Classe StoryTest

    def __init__(self, test_file: str = "tests.txt")
        Initialise le testeur avec un fichier contenant des histoires a trous.

    def run_all_tests(self)
        Execute tous les tests definis dans le fichier.
        Pour chaque histoire, predit les lignes manquantes.

    def _run_and_print(self, title: str, story_lines: List[str])
        Affiche chaque histoire avec les lignes predites a la place des "?".

Fichier : story_generator.py

Classe StoryGenerator

    def __init__(self, input_file: str = "histoires.txt", output_dir: str = "data/stories")
        Initialise les dependances, le repertoire de sortie et charge les outils.

    def load_and_store_stories(self)
        Charge les histoires depuis le fichier texte, les segmente et les stocke apres validation.

    def _create_and_add_story(self, domain: str, sentences: List[str], story_id: int)
        Cree une structure d'histoire avec des factoides, identifie chaque element et stocke l'histoire.

    def check_story_consistency(self, story: Dict[str, Any], force: bool = False, ignore: bool = False) -> bool
        Verifie la validite des termes et relations avec JDM. Peut forcer ou ignorer les erreurs.

Fichier : story_generalizer.py

Classe StoryGeneralizer

def __init__(...)
Initialise le client JDM, l'extracteur et le validateur d'histoires.

def _get_generalizations1(self, term: str) -> list
Tente de generaliser un terme en comparant a une liste filtree de candidats pertinents.

def _get_generalizations(self, term: str) -> list
Recherche les hyperonymes d'un terme a partir des relations r_isa.

def generalize_factoid_story(...)
Applique une generalisation semantique sur les sujets et objets d'une histoire si les relations restent valides.

def _rebuild_factoid_sentence(f: Dict[str, str]) -> str
Construit une phrase a partir d'un factoide.

Fonctions globales :

def test_all_stories()
Applique la generalisation a toutes les histoires presentes.

def test_story(story_id, save_story: bool = False)
Applique la generalisation a une histoire specifique et affiche les changements.

Fichier : story_database.py

Classe statique StoryDatabase

def initialize(storage_dir: str = "data/stories")
Charge tous les fichiers necessaires (histoires, factoides, termes, relations).

def add_story(story: Dict, generalized: bool = False)
Ajoute une histoire et ses factoides a la base de donnees et les enregistre.

def _save_story_to_file(story: Dict)
Enregistre une histoire sous forme JSON.

def _save_all_factoids()
Sauvegarde tous les factoides en fichier.

def save_valid_terms(), save_valid_relations()
Sauvegarde les termes et relations valides manuellement.

def load_valid_terms(), load_valid_relations()
Charge les fichiers de validation.

def is_valid_term(term: str), is_valid_relation(...)
Verifie qu'un terme ou relation est valide(e).

def add_valid_term(term: str), add_valid_relation(...)
Ajoute manuellement un terme ou une relation comme valide.

def get_factoid_by_id(...), find_factoid(...)
Recherche des factoides par ID ou attributs.

def list_factoids(), get_story(...), list_stories()
Fonctions utilitaires pour acceder aux donnees.

def load_all_stories(), load_all_factoids()
Chargement brut depuis les fichiers disques.

Fichier : factoid_extractor.py

Classe FactoidExtractor

def __init__(...)
Initialise les repertoires et charge les factoides existants.

def _extract_factoid_components(sentence: str) -> Dict[str, Any]
Extrait les composants formels d'une phrase etiquetee.

def extract_factoids_from_story(story: Dict[str, Any]) -> List[Dict[str, Any]]
Extrait tous les factoides d'une histoire donnee.

def extract_all_factoids() -> List[Dict[str, Any]]
Applique l'extraction a toutes les histoires sauvegardees.

Fichier : factoid_predict_1.py

Classe FactoidPredict1

def __init__(...)
Initialise les poids, le nombre d'echantillons et les dependances.

def _factoid_text(factoid: Dict) -> str
Construit une phrase textuelle a partir d'un factoide structure.

def predict_missing(story_lines: List[str]) -> List[str]
Predit les lignes manquantes d'une histoire a partir du contexte precedent/suivant.

def check_terms(term1: str, term2: str, type: str) -> bool
Verifie si deux termes correspondent (actuellement par egalite stricte).

def check_pattern1 a check_pattern4(...)
Detecte differentes structures/repetitions logiques entre les factoides.

def _predict_from_previous(prev_line: str, all_subjects: List[str]) -> str
Predit une ligne manquante a partir de la ligne precedente.

def _predict_from_next(next_line: str, all_subjects: List[str]) -> str
Predit une ligne manquante a partir de la ligne suivante.

Fichier : jdm_client.py

Classe JDMCache

def __init__(...)
Initialise un systeme de cache (en memoire et sur disque).

def get(...), set(...)
Permet de stocker et recuperer des reponses d'API localement.

Classe JDMClient

def __init__(...)
Initialise le client JDM, configure le cache et les identifiants de relations.

def get_node_by_name(node_name: str) -> Dict[str, Any]
Recupere les informations sur un mot donne via JDM.

def get_relations_from(...), get_relations_to(...), get_relations_from_to(...)
Recupere les relations sortantes, entrantes ou specifiques entre deux noeuds.

def get_relation_types() -> List[Dict[str, Any]]
Recupere les types de relations disponibles.

def has_relation(source: str, target: str, relation_name: str) -> bool
Verifie l'existence d'une relation directionnelle donnee dans JDM.

Fichier : main.py

Ce fichier constitue le point d'entree principal du programme. Il fournit une interface en ligne de commande (CLI) permettant d'executer diverses fonctions liees aux histoires et aux factoides.

Fonctions principales :

def importer_histoires(input_file="histoires.txt")
Utilise le StoryGenerator pour importer des histoires depuis un fichier texte, les analyser et les stocker.

def generaliser_histoires()
Lance la generalisation de toutes les histoires via la fonction test_all_stories du module story_generalizer.

def tester_histoire(story_id)
Applique un test de generalisation sur une histoire specifique sans la sauvegarder.

def lister_histoires()
Initialise la base de donnees et affiche la liste de tous les identifiants d'histoires stockees.

def afficher_histoire(story_id)
Affiche tous les factoides d'une histoire identifiee, en format lisible avec les roles (sujet, predicat, etc.).

def prediction_incomplete()
Interface interactive pour entrer une histoire ligne par ligne, avec des trous marques par "?".
Utilise FactoidPredict1 pour completer automatiquement les lignes manquantes.

def prediction_incomplete_from_file()
Demande un chemin vers un fichier d'histoire contenant des "?" a predire, puis execute les predictions via StoryTest.

Bloc principal :

if __name__ == "__main__"

Analyse les arguments de la ligne de commande a l'aide d'argparse.

Les sous-commandes disponibles sont :

- import : pour charger et enregistrer des histoires depuis un fichier
- generalize : pour appliquer la generalisation a toutes les histoires
- list : pour afficher les IDs d'histoires disponibles
- predict : pour completer une histoire interactive a trous
- predict-from-file : pour completer une histoire a partir d'un fichier
- show <id> : pour afficher le contenu d'une histoire
- test <id> : pour tester la generalisation d'une histoire