MedExtractor

Release 1.0.0

Fapra Gruppe 5

Jan 27, 2023

CONTENTS:

Fachpraktikum WS 22/23 Natural Language Processing (NLP) mit spaCy

CHAPTER

ONE

MEDEXTRACTOR - KONSOLENAPPLIKATION

Die Medextractor-Konsolenapplikation analysiert Texte und sucht darin nach Krankheiten und deren Symptomen und erstellt eine Wissensrepräsentation, die die gefundenen Krankheiten und Symptome miteinander in Beziehung setzt. Die Wissensrepräsentation wird im RDF (Resource Discription Framework)-Format gespeichert. Zusätzlich erstellt der Medextractor eine xml-Datei mit Daten für den Entity Linker von spaCy sowie eine Datei name.kb, in dem die erstellte Wissensbasis in Binärcode abgespeichert wird.

CHAPTER

TWO

KONFIGURATIONSDATEI CONFIG.JSON

Das Python-Modul, mit dem die Konsolenapplikation gestartet wird, ist die Datei: medextractor.py. Im selben Order von medextractor.py muss sich die Konfigurationsdatei config.json befinden.

Sollte es noch keine config.json Datei geben, wird beim Aufruf von medextractor.py eine Beispiel-Datei erzeugt, die anschließend vom Nutzer angepasst werden kann.

Die Konfigurationsdatei enthält folgende Informationen:

1. Pfad und Name der xml-Datei für den Export im RDF-Format
2. Pfad und Name der xml-Datei für den Export für den Entity Linker
3. Pfad und Name der KnowledgeBase Datei
4. Pfad zu dem Order, der die zu analysierenden Texte enthält
5. Spezifikation, ob die Knowledgebase Datei überschrieben werden soll (True oder False)
6. Pfad und name der .txt-Datei, die das Krankheiten-Vokabular enthält
7. Pfad und name der .txt-Datei, die das Symptome-Vokabular enthält

Die Pfade müssen relativ zu dem Order angegeben werden, in dem sich medextractor.py befindet. Alternativ können auch absolute Pfade angeben werden.

Es werden alle Textdateien (*.txt) analysiert, die sich in dem in der config.json-Datei angegebenen Ordner befinden. Die von Medextractor erzeugten xml- und Knowledgebase- Dateien enthalten ein über alle analysierten Texte akkumuliertes Ergebnis.

Wird festgelegt, dass die Knowledgebase-Datei nicht überschrieben werden soll, werden alle neu gefundenen Krankheit-Symptom-Beziehungen zu der vorhandenen Knowledgebase-Datei hinzugefügt.

CHAPTER

THREE

VOKABULAR-DATEIEN

Die Vokabulardateien sind einfache Dateien im csv-Format und enthalten Einträge der folgenden Art:

C0010051 coronary aneurysm DISEASE

Der Eintrag C0010051 ist der CUI (Concept Unique Identifier) aus der MetaMapLite-Datenbank. Der CUI ist als Referenz enthalten, wird aber nicht weiter vom Medextractor verwendet.

CHAPTER

FOUR

AUFRUF DES PROGRAMMS

4.1 Voraussetzungen

- Packages, die in requirements.txt aufgelistet sind, sind installiert (Installation aller Packages möglich mit dem Befehl `pip install -r requirements.txt`)

4.2 Aufruf

Das Programm wird gestartet, indem in die Windowseingabeaufforderung der Befehl

`python medextractor.py`

eingegeben wird.

Zu beachten ist, dass in der System-Path-Umgebungsvariable der Pfad zur (ggf. virtuellen) Umgebung des Python-Interpreters enthalten ist, in der spaCy installiert wurde. Ggf. sollte hierzu active.bat im Verzeichnis der virtuellen Umgebung der Python-Installation aufgerufen werden.

Da die Vokabulardateien umfangreich sind, dauert allein das Trainieren des Entity-Rulers typischerweise über eine Minute.

Nach Beendigung des Programms befinden sich die xml-Dateien mit der RDF-Repräsentation sowie die xml-Datei für den Entity Linker in dem in config.json angegebenen Ordner.

CHAPTER

FIVE

ENTITY-LINKER

Das Jupyter-notebook entity_linker_demo.ipynb demonstriert, wie die Daten aus der xml-Export-Datei gelesen und für das Training von Entity Ruler und Entity Linker verwendet werden. Findet der Entity Ruler in einem Text Symptome, dann ordnet der Entity Linker diesen Symptome dazugehörige Krankheiten zu.

CHAPTER

SIX

MEDEXTRACTOR

6.1 medextractor package

6.1.1 Subpackages

medextractor.dummy package

Submodules

medextractor.dummy.dummy module

class medextractor.dummy.dummy.**DummyKnowledgeExtractor**

Bases: *KnowledgeExtractorInterface*

get_knowledge_base()

class medextractor.dummy.dummy.**DummyPreprocessor**(*doc_name*)

Bases: *PreprocessingInterface*

get_preprocessed_text()

class medextractor.dummy.dummy.**DummyRDFSerialiser**(*knowledgebase*, *namespace*, *namespace_prefix*)

Bases: *RDFSerialiserInterface*

serialise_knowledgebase()

Module contents

medextractor.interfaces package

Submodules

medextractor.interfaces.interfaces module

class medextractor.interfaces.interfaces.**KnowledgeExtractorInterface**

Bases: ABC

abstract get_knowledge_base() → str

class medextractor.interfaces.interfaces.**PreprocessingInterface**(*doc_name*)

Bases: ABC

abstract get_preprocessed_text() → str

class medextractor.interfaces.interfaces.**RDFSerialiserInterface**(*knowledgebase*, *namespace*, *namespace_prefix*)

Bases: ABC

abstract serialise_knowledgebase(*output_path*) → str

abstract set_serialisation_format(*serialisation_format*)

Module contents

medextractor.knowledge package

Submodules

medextractor.knowledge.base module

class medextractor.knowledge.base.**KnowledgeBase**

Bases: object

The KnowledgeBase manages entities and relations.

Functions: add_relation(SemanticRelation) has_relation(SemanticRelation) -> bool give_entities(str) -> [] export_for_entity_linker(str) safe(str) load(str)

add_relation(*relation:* SemanticRelation) → None

Add a SemanticRelation into the KnowledgeBase.

@param relation: a SemanticRelation

add_training_example_to_relation(*relation:* SemanticRelation, *sent_text: str*) → None

Add a training sentence to a SemanticRelation

@param relation: a SemanticRelation @param sent_text: a training sentence

export_for_entity_linker(*file_name: str*)

Save the KnowledgeBase data as an xml file that can be used to train an entity linker.

@param alias: the name of the xml file

give_entities(*alias: str*) → []

Return a list of entities that are related to symptoms in SemanticRelations stored in the KnowledgeBase

@param alias: the name of a symptom

has_relation(*relation:* SemanticRelation) → bool

Return True if the relation is in the KnowledgeBase. Return False otherwise.

@param relation: a SemanticRelation

load(*file_name: str*)

Load the KnowledgeBase from file.

@param file_name: the name of the file

save(*file_name: str*) → None

Save the KnowledgeBase into a pickle file. If the KnowledgeBase does not contain any SemanticRelations, no file is saved.

@param file_name: the name of the file

medextractor.knowledge.entity module

class medextractor.knowledge.entity.**Entity**(*entity_name: str*, *entity_type:* EntityType)

Bases: object

An entity consisting of its name string and its EntityType

class medextractor.knowledge.entity.**EntityType**(*value*)

Bases: Enum

Types of entities that can be stored in the KnowledgeBase.

DISEASE SYMPTOM UNDEFINED

DISEASE = 1

SYMPTOM = 2

UNDEFINED = 3

medextractor.knowledge.relations module

class medextractor.knowledge.relations.**RelationType**(*value*)

Bases: Enum

Types of relations to be used in the KnowledgeBase.

HAS_SYMPTOM = 2

IS_SYMPTOM_OF = 1

NO_RELATION = 3

medextractor.knowledge.semantics module

class medextractor.knowledge.semantics.**SemanticRelation**(*entity_1:* Entity, *entity_2:* Entity, *relation_type:* RelationType, *training_sample: Optional[str] = None*)

Bases: object

A semantic relation between two entities connected by a value of RelationType.

Additionally, training samples can be saved that resulted in this semantic relation.

add_training_sample(*training_sample: str*)

Adds the training_sample into the list of training_samples.

training_sample: string
The text sample/sentence to be added

None

contains_training_sample(*training_sample: str*) → bool

Checks whether the training_sample given is already included in the list of training samples.

training_sample: string
A text sample/sentence

true, if training_sample is contained in the list, false otherwise

Module contents

knowledge package

contains the modules: base entity relations semantics

This is the form of a docstring.

It can be spread over several lines.

medextractor.knowledge_extractor package

Submodules

medextractor.knowledge_extractor.knowledge_extractor module

class medextractor.knowledge_extractor.knowledge_extractor.**KnowledgeExtractor**(*config:* ConfigManager)

Bases: *KnowledgeExtractorInterface*

KnowledgeExtractor searches a text string for entities and for relations between these entities

call2(*text*)

export_for_entity_linker()

Exports all entities, aliases and example sentences into an xml-File. The data is prepared for easy import into spaCy's Entity Linker. The xml-File is human readable and allows reviewing the data that will be used by the Entity Linker. Path and filename are defined in config.json.

None

None

get_knowledge_base()

Returns the knowledgebase that contains all entities and sample sentences. Samples sentences can be used for training statistical models (e.g. Entity Linker)

None

KnowledgeBase

process_texts()

Analyzes all text documents in the folder specified in config.json

None

None

saveKB(**args*)

Saves the database persistently. Optionally, path and file name are given as a string parameter when calling this function. If no path and file name are given, the function will use the path and file name in attribute self._knowledgebase_filename.

(optional) file_name (string)

None

set_context(*context*)

This function allows defining a context. The context is described by named entities included in the Entity Ruler (self._ruler). These entities will be added to the set of entities when searching for disease/symptom relations between entities.

context: {} (set of spacy.Spans = Entities of Entity Ruler)

None

Module contents

knowledge_extractor package

contains the knowledge_extractor module

This is the form of a docstring.

It can be spread over several lines.

medextractor.preprocessor package

Submodules

medextractor.preprocessor.preprocessor module

class medextractor.preprocessor.preprocessor.**RuleBasedPreprocessor**(*doc_name*)

Bases: *PreprocessingInterface*

get_preprocessed_text() → str

pysbd_sentence_boundaries()

Module contents

medextractor.rdf package

Submodules

medextractor.rdf.RDFSerialiser module

class medextractor.rdf.RDFSerialiser.**RDFSerialiser**(*knowledgebase*, *namespace*, *namespace_prefix*)

Bases: *RDFSerialiserInterface*

knowledgebase_to_graph()

serialise_knowledgebase(*output_path*)

set_serialisation_format(*serialisation_format*)

medextractor.rdf.graphmanager module

class medextractor.rdf.graphmanager.**GraphManager**(*namespace_prefix*, *namespace_uri*)

Bases: object

add_disease(*disease*)

add_symptom(*disease*, *symptom*)

get_serialized_graph(*output_path*, *serialization_format='pretty-xml'*)

Module contents

6.1.2 Submodules

6.1.3 medextractor.config_manager module

class medextractor.config_manager.**ConfigManager**

Bases: object

6.1.4 medextractor.manual_rdf_graph module

medextractor.manual_rdf_graph.**add_symptom**(*symptom_str: str*)

6.1.5 medextractor.medextractor module

6.1.6 medextractor.ruler_creator module

class medextractor.ruler_creator.**RulerCreator**

Bases: object

load()

save() → None

6.1.7 Module contents

Main Medextractor

This is the form of a docstring.

It can be spread over several lines.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

I

K

L

M

N

P

R

S

U