MedExtractor

Release 1.0.0

Fapra Gruppe 5

Mar 16, 2023

CONTENTS:

CHAPTER

ONE

FACHPRAKTIKUM WS 22/23 - NATURAL LANGUAGE PROCESSING (NLP) MIT SPACY

CHAPTER

TWO

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

THREE

ORDNERSTRUKTUR

- **docs**: mit Sphinx erstellte Dokumentation des MedExtractor-Programms, mit Öffnen der html-Dateien in docs/build kommt man zu einer ansprechenden Dokumentation des Programms
- **medextractor**: enthält die Programme, die für die Erstellung der Wissensbasis zuständig sind
  - **preprocessor**: Programm, das einen gegebenen Text vorverarbeitet, sodass es in weiteren Schritten besser verarbeitet werden kann
  - **knowledge**: verarbeitet vorverarbeiteten Text, extrahiert Krankheiten und dazugehörige Symptome und speichert sie in einer Knowledgebase
  - **rdf**: Serialisiert die erstellte Knowledgebase und speichert sie als xml-Datei im RDF-Format
- **resources**: alle Ressourcen, die zum Ausführen des Programms benötigt werde, sowie Ausgabedateien
  - **to_analyze**: Texte, die vom Programm analysiert und ausgewertet werden können
  - **training_data**: Vokabular-Dateien (siehe Abschnitt unten)
- **test**: Testprogramme

CHAPTER

FOUR

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 muss.

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

FIVE

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

SIX

AUFRUF DES PROGRAMMS

6.1 Voraussetzungen

- eine Python Version 3.6-3.8 (empfohlen und getestet: 3.8) muss installiert sein (SpaCy ist noch nicht kompatibel mit Python >3.8)
- Packages, die in requirements.txt aufgelistet sind, sind installiert (Installation aller Packages möglich mit dem Befehl `pip install -r requirements.txt`)

6.2 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 activate.bat im Verzeichnis der virtuellen Umgebung der Python-Installation aufgerufen werden.

Da die Vokabulardateien umfangreich sind, kann allein das Trainieren des Entity-Rulers (je nach Rechner) eine Minute übersteigen.

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

SEVEN

ENTITY-LINKER

Das Jupyter-Notebook entity_linker_demo.ipynb (zu finden im Ordner NLP_63458_WS22/notebooks/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

EIGHT

MEDEXTRACTOR

8.1 medextractor package

8.1.1 Subpackages

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.

Parameters
relation (SemanticRelation) –

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

Add a training sentence to a SemanticRelation

Parameters

- **relation** (SemanticRelation) –
- **sent_text** (*str*) – 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.

Parameters
file_name (*str*) – the name of the xml file

get_entities(*alias: str*) → []

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

Parameters
alias (*str*) – the name of a symptom

Return type
list of Entity

has_relation(*relation:* SemanticRelation) → bool

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

Parameters
relation (SemanticRelation) –

load(*file_name: str*)

Load the KnowledgeBase from file.

Parameters
file_name (*str*) – 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.

Parameters
file_name (*str*) – 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 = 1

SYMPTOM = 2

UNDEFINED = 3

medextractor.knowledge.knowledge_extractor module

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

Bases: object

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

analyze_linguistically(*text*)

Method that finds entities in a given text and outputs them on the command line together with part-of-speech tags and the syntactic dependency within the sentence

Parameters
text (*string*) – The text string to be analyzed by the method

Return type
None

`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.

Parameters
None –

Return type
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)

Parameters
None –

Return type
KnowledgeBase

`is_related(entity1, entity2, sent)`

Returns relation type of entity1 and entity2. If both entities are found to be unrelated, RelationType.NO_RELATION is returned.

Parameter sent is not used because this function currently only implements a very simple relation check without analyzing the syntax of the sentence. Such analysis could be added at a later stage.

At the moment is_related() just checks whether entity1 is a disease and whether entity2 is a symptom. Thus possible results are only RelationType.NO_RELATION and RelationType.HAS_SYMPTOM.

Parameters

- **entity1** (*spacy.Span*) –
- **entity2** (*spacy.Span*) –
- **sent** (*spacy.Span*) –

Return type
RelationType (Enum)

`process_texts()`

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

Parameters
None –

Return type
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._config.knowledgebase_filename.

Parameters
file_name (*string optional*) –

Return type
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.

Parameters
context (*{} (set of spacy.Spans = Entities of Entity Ruler)*) –

Return type
None

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.

Parameters
training_sample (*string*) – The text sample/sentence to be added

Return type
None

contains_training_sample(*training_sample: str*) → bool

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

Parameters
training_sample (*string*) – A text sample/sentence

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

Module contents

medextractor.preprocessor package

Submodules

medextractor.preprocessor.preprocessor module

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

Bases: object

get_preprocessed_text() → str

Reads the text given in the document with which the Preprocessor is initialised and processes this text such that it is in a good format for further processing.

Parameters
None –

Return type
string

pysbd_sentence_boundaries()

Creates a SpaCy pipeline component to segment a text into sentences using pysbd.

Parameters
doc (*Doc*) – SpaCy Doc object

Returns
doc – SpaCy Doc object

Return type
Doc

Module contents

medextractor.rdf package

Submodules

medextractor.rdf.RDFSerialiser module

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

Bases: object

knowledgebase_to_graph()

Transfers the content of the knowledgebase into an rdflib-graph.

Parameters
None –

Return type
None

serialise_knowledgebase(*output_path*)

Serialises knowledge base into an RDF file.

Parameters
output_path (*string*) – path to which the xml-document resulting from the rdflib-Graph will be saved

Return type
None

medextractor.rdf.graphmanager module

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

Bases: object

add_symptom(*disease*, *symptom*)

Adds the given symptom together with the given disease to the rdflib-graph and saves the given disease in the set of diseases.

Parameters

- **disease** (*string*) –
- **symptom** (*string*) –

Return type
None

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

Serializes the graph according to the given serialization_format and saves the resulting document to the given output_path.

Parameters

- **output_path** (*string*) –
- **serialization_format** (*string*) –

Return type
None

Module contents

8.1.2 Submodules

8.1.3 medextractor.config_manager module

class medextractor.config_manager.**ConfigManager**

Bases: object

8.1.4 medextractor.create_manual_rdf_graph module

Program for "manual" creation of RDF-Graphs: Disease and corresponding symptoms are given, an RDF-Graph is built and saved as xml-file.

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

8.1.5 medextractor.medextractor module

8.1.6 medextractor.ruler_creator module

class medextractor.ruler_creator.**RulerCreator**

Bases: object

load()

save() → None

8.1.7 Module contents

Main Medextractor

This is the form of a docstring.

It can be spread over several lines.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX