Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik

Entity Linking for the Biomedical Domain

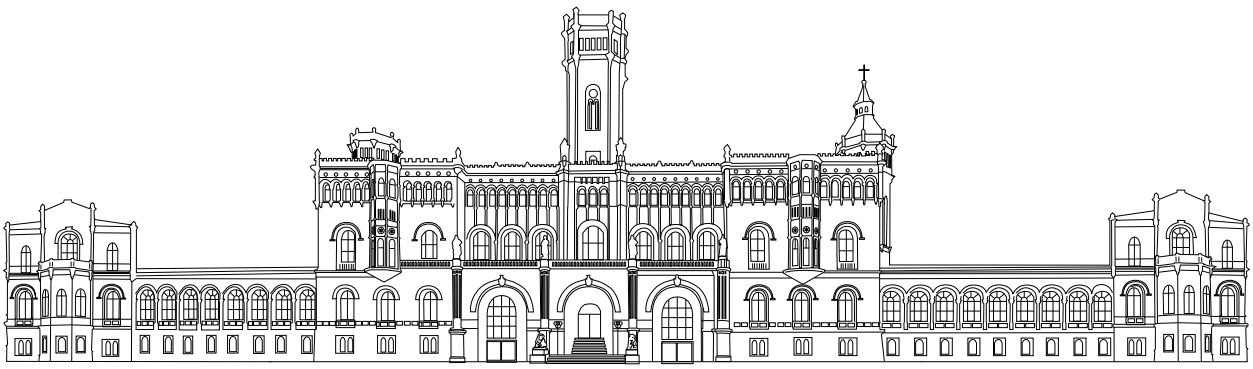
A thesis submitted in fulfillment of the requirements for the degree of
Master of Computer Science

BY

Sahar Nassimi
Matriculation number: 10027234
E-mail: s.faghihinezhad@stud.uni-hannover.de

First evaluator: Prof. Dr. Maria-Esther Vidal
Second evaluator: Prof. Dr. Sören Auer
Supervisor: M.Sc. Ahmad Sakor

February 10, 2023



Declaration of Authorship

I, Sahar Nassimi, declare that this thesis titled, 'Entity Linking for the Biomedical Domain' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Sahar Nassimi

Signature: _____

Date: _____

Acknowledgements

I would first like to thank Prof. Dr. Maria-Esther Vidal, who let me write my master thesis at the Scientific Data Management research group of the TIB-Leibniz Information Center of Technology and Natural Sciences at Leibniz University. The door to Prof. Vidal office was always open whenever I ran into a trouble spot or had a question about my research or writing. She consistently allowed this work to be my own work, but steered me in the right the direction whenever she thought I needed it.

I would also like to thank my supervisor Mr. Ahmad Sakor whose insight and knowledge into the subject matter steered me through this research. He has supported me throughout my thesis and provided guidance and feedback throughout this project. Without his passionate participation and input, the validation survey could not have been successfully conducted.

Finally, I must express my very profound gratitude to my family and to my beloved husband-I simply couldn't have done this without you, special thanks. Thank you for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without you.

Abstract

Entity linking is the process of detecting mentions of different concepts in text documents and linking them to canonical entities in a target lexicon. However, one of the biggest issues in entity linking is the ambiguity in entity names. The ambiguity is an issue that many text mining tools have yet to address since different names can represent the same thing and every mention could indicate a different thing. For instance, search engines that rely on heuristic string matches frequently return irrelevant results, because they are unable to satisfactorily resolve ambiguity. Thus, resolving named entity ambiguity is a crucial step in entity linking. To solve the problem of ambiguity, this work proposes a heuristic method for entity recognition and entity linking over the biomedical knowledge graph concerning the semantic similarity of entities in the knowledge graph. Named entity recognition (NER), relation extraction (RE), and relationship linking make up a conventional entity linking (EL) system pipeline (RL). We have used the accuracy metric in this thesis. Therefore, for each identified relation or entity, the solution comprises identifying the correct one and matching it to its corresponding unique CUI in the knowledge base. Because KBs contain a substantial number of relations and entities, each with only one natural language label, the second phase is directly dependent on the accuracy of the first. The framework developed in this thesis enables the extraction of relations and entities from the text and their mapping to the associated CUI in the UMLS knowledge base. This approach derives a new representation of the knowledge base that lends it to the easy comparison. Our idea to select the best candidates is to build a graph of relations and determine the shortest path distance using a ranking approach. We test our suggested approach on two well-known benchmarks in the biomedical field and show that our method exceeds the search engine's top result and provides us with around 4% more accuracy. In general, when it comes to fine-tuning, we notice that entity linking contains subjective characteristics and modifications may be required depending on the task at hand. The performance of the framework is evaluated based on a Python implementation.

Contents

List of Figures

Chapter 1

Introduction

Entity linking is the task of identifying mentions of named entities (or other terms) in a text document and disambiguating by mapping them to canonical entities (or concepts) listed in a reference knowledge graph[89]. The goal of entity linking, also known as entity disambiguation, is to link the mentions in text, like "Steve", to the correct entity in the knowledge graph, like "Steve Jobs" on Wikipedia. The mention is always ambiguous, for example, "Steve" can also be Steve Wozniak or many other entities. Thus, how to use the context to disambiguate the mention is the major problem in entity linking. Entity linking for text is quite important [72]. A wide range of web corpora is in the form of text, such as Question-Answer (QA) queries, search queries, and news titles. The applications on these texts such as question-answering systems and search engines cannot function normally if the mentions are not correctly linked. Therefore, the challenge of text entity linking lies in making full use of context words.[72]. In the biomedical domain, entity linking connects references to diseases, drugs, and treatments to normalized entities in standard vocabularies. It is a critical part of automation, in the fields of public health, research, and clinical practice. Since there are different names for the same entities in Medical Information Systems, using and integrating medical data is challenging. If a drug has multiple names, researchers are unable to evaluate its effects, and patients face the risk of accidentally taking the same medication more than once [71].
Since a word often only relates to one entity, ambiguity is not the main problem with biological entity linking. The complexity lies in the fact that the surface forms vary significantly as a result of abbreviations, morphological changes, synonymous words, and different word orders. For instance, "Hepatitis C virus" and "stomach cancer" are both abbreviated as "HCV" and "Malignant Gastric Neoplasm," respectively. There are so many different surface forms that it is impossible to predict all the

various representations of an object in advance. Because these systems assume that all possible forms of an entity are known, they cannot be used in our circumstances. As a result of this challenge, entity-linking techniques have been developed specifically for biomedical entity-linking. Generally, existing approaches in this field have limitations: In earlier biomedical research, string similarities of mentions and entity names were captured using rule-based systems that require manual rule definition, and the rules were tied to an application. Existing entity linking approaches are unable to efficiently find semantically related words. The state-of-the-art approaches heavily depend on training the data, and if the data is not available, they cannot operate well. The well-annotated linking data rely on lots of manpower and is time-consuming. In the related work section, we will discuss in detail the state-of-the-art approaches for entity linking in the biomedical domain.

1.1 Motivating Example

We explore a motivating example to provide a clear picture of the issue that this thesis attempts to resolve.
In this work, we focus on extracting and linking entities and relations for medical texts in Unified Medical Language System (UMLS) [69]. In this scenario, as the Figure 1.1 shows, the input text is typically a sentence like "The doctor checks his heart rate, blood pressure, and temperature because of his fever." and "The doctor told him to stay at home in extreme-temperatures such as Miliaria, heat exhaustion, etc." The temperature in the first sentence is inferred from words like "heart rate" "blood pressure" and "fever" to be about body temperature, so it should be linked to a cluster of concepts about the human body. While the temperature in the latter sentence is inferred from words like "extreme-temperatures", "Miliaria" and "heat exhaustion" to be related to weather temperature, and it should be linked to a cluster of concepts about the weather.
Regarding our motivating example, it can be problematic if the disambiguation and linking are handled incorrectly because different biomedical concepts may have mentions that are quite similar to one another. If this happens, the context as a whole might be incorrectly interpreted. In contrast to earlier methods, in this work, we build the whole graph of relation in UMLS in the pre-processing stage, where the nodes represent the context surrounding the target mention and the relevant knowledge base entries, and the directed edges stand in for the reference dependencies between the nodes. We then assign a higher rank to semantically similar entities based on the context in which the entity is presented.

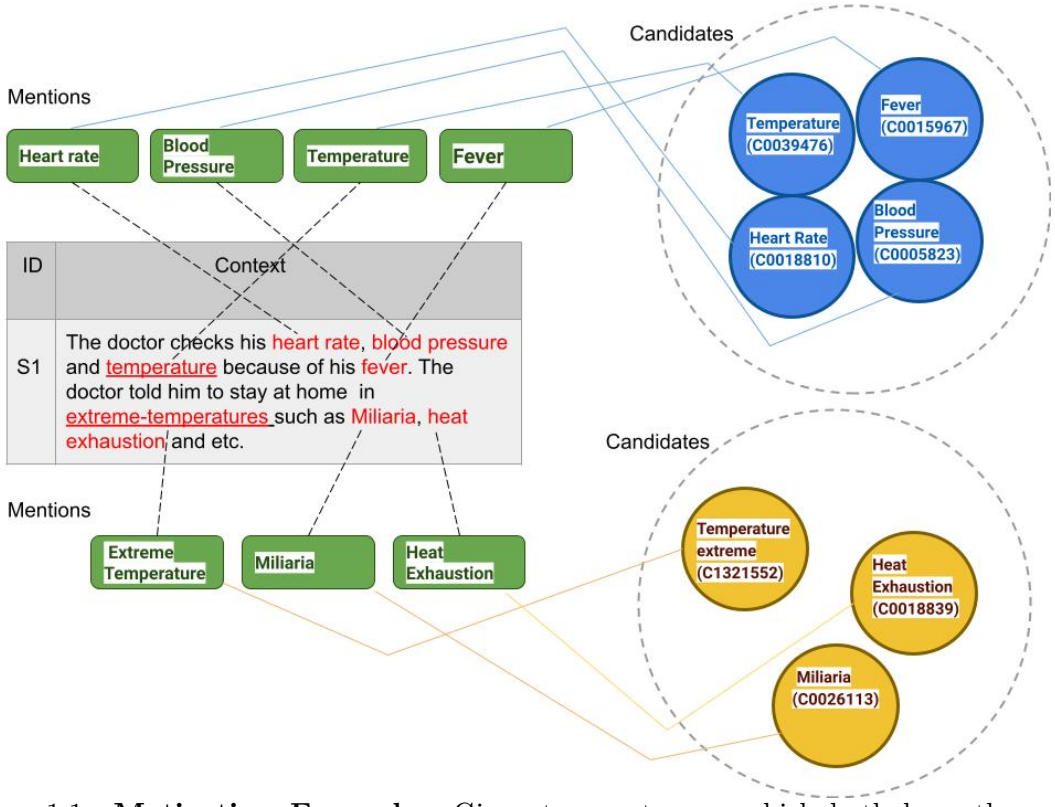


Figure 1.1: **Motivating Example.** Given two sentences, which both have the word "temperature", the challenge is to correctly link the "temperature", to its correct concept unique identifier (CUI) in the Unified Medical Language System [69] knowledge base. All other highlighted entities are used to find the semantically similar contexts in each sentence and link the mention "temperature" to its correct counterpart, with the help of other semantically similar entities

1.2 Contributions

This thesis contributes to the integration of the semantic web and natural language processing (NLP) as two distinct computer science fields. We propose a framework focusing on Named Entity Recognition (NER), and Entity Linking (EL) tasks. The main contributions of this work are: 1. Introducing an entity linking method within the graph-based framework. 2. Developing a new heuristic strategy to detect entities and relations in a text by using similarity measures like shortest path length between two words. 3. Creating a new representation of the knowledge base by using the labels of the data and placing them in an indexed-based database. 4. Using the heuristic method and "Elasticsearch", to do entity disambiguation and relation

linking. 5. Establishing criteria for candidates ranking, that can be applied to rank the Concept Unique Identifier (CUIs) of entities and relations linked to their KB.

1.3 Document Structure

This thesis consists of seven chapters. Chapter 1, provides an overview of the whole document. We give the reader a motivating example, a list of the major contributions, and a description of the problem that is being addressed in this work. Chapter 2 covers the background, and the fundamental principles required to understand the subsequent chapters. After acquiring the essential concepts, in Chapter 3 we discuss the related papers to this work, review the state-of-the-art while referring to relevant publications, and position the thesis concerning them. The approach, as well as the formal problem definition, and the proposed solution, are discussed in detail in Chapter 4. The motivating example is presented as a running example to illustrate the approach for a better understanding of the approach. In Chapter 5, the implementation of the approach highlights the different modules and development decisions. In Chapter 6, the implementation's performance is evaluated concerning its performance based on a popular benchmark in the biomedical domain. Eventually, in Chapter 7, we wrap up our thesis work and talk about the method's limitations as well as future works.

1.4 Summary of the Chapter

In conclusion, this chapter introduces the entity linking for texts in the biomedical domain. To better understand the problem that has to be addressed, the chapter also offers a motivating example. In addition, this chapter described the contribution of this thesis.

Chapter 2

Background

This chapter presents the terminology and concepts required to understand the problem tackled in this thesis. The principles of entity linking and its definition in the biomedical domain, numerous types of entity linking, deep learning techniques, and various technical domains where these concepts are applied are all attempted to be well understood in the section that follows.

2.1 Knowledge Base

A knowledge base is a set of interlinked representations of entities, such as real-world objects, incidents, situations, or abstract concepts, linked together to enable the storage, evaluation, and reuse of this knowledge in a machine-interpretable way. The Wikidata [122], Amazon, Netflix, Industry-scale Knowledge Graphs [112] (such as Microsoft, Google (FANGs) and Facebook) have long ago realized this and use knowledge bases.
Knowledge bases vary from a simple database to providing a structured data collection that is more similar to how the human brain arranges information. They rely on semantic models that consist of rules for interpreting the data as well as a formal classification with classes, subclasses, relationships, and instances (e.g., ontologies and dictionaries). Figure 2.1 shows the general concept of a knowledge base. The DBpedia [66] project derives a data corpus from the Wikipedia encyclopedia. DBpedia works on converting Wikipedia content into structured data, so Semantic Web techniques can be used against it, such as running complex queries against Wikipedia, connecting it to other online datasets, or developing new applications or interactive content. Figure 2.2 provides a summary of the information extraction process used by DBpedia and demonstrates how the data is presented on the

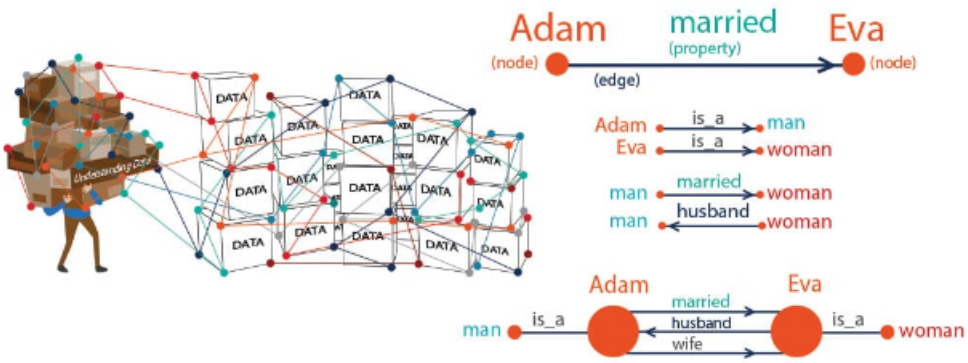


Figure 2.1: **Knowledge-Base Concept Definition** [1], Knowledge bases give the data a semantic model that consists of rules for interpreting the data as well as a formal classification with classes, subclasses, relationships, and instances (ontologies and dictionaries).

web. The structured information includes graphics, geographic locations, external links, and some other items. The host for the DBpedia RDF dataset is OpenLink Virtuoso [2]. The data can be accessed either by SPARQL queries or HTTP requests. DBpedia 2014, which is released on May 2014, consists of 2.46 billion RDF (Resource Description Framework) triples obtained from Wikipedia's other language editions and 3 billion RDF triples which were extracted from 580 million pages from the English version of Wikipedia[1]. One example of the knowledge base is UMLS [69], or Unified Medical Language System, which is a set of files and software that brings together many health and biomedical vocabularies and standards to enable interoperability between computer systems. The purpose of the National Library of UMLS is to facilitate the development of computer systems that behave as if they "understand" the meaning of the language of biomedicine and health. The UMLS provides data for system developers and search and report functions for less technical users. The graph representation of UMLS is shown Figure 2.3. There are three UMLS Knowledge Sources[3]:

- The Metathesaurus, which contains over one million biomedical concepts from over 100 source vocabularies
- The Semantic Network, which defines 133 broad categories and fifty-four relationships between categories for labeling the biomedical domain

[1] `http://wikidata.dbpedia.org/about`

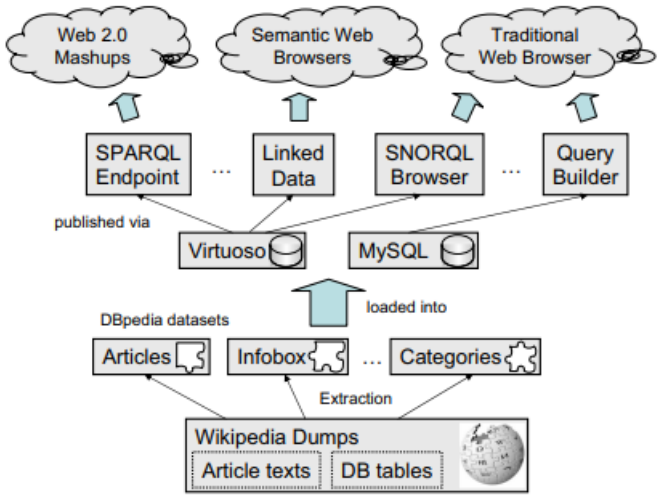


Figure 2.2: **Overview of the DBpedia components**. It gives a general overview of the information extraction process used by DBpedia and demonstrates how the data is presented on the internet. Virtuoso [79] and MySQL are now used as the storage backends for these primary DBpedia interfaces. [66]

- The SPECIALIST Lexicon & Lexical Tools, which provide lexical information and programs for language processing

They are distributed with flexible lexical tools and MetamorphoSys, the UMLS install and customization program. An entity name can be considered as anything referred to with a proper noun and is tied to a particular kind of lexical unit connected to specific domains that have a proper name, such as people, places, organizations, and more. It also contains numerical phrases and the names of specific categories such as pharmaceuticals, illnesses, works of art, etc. The phrase "entity name" has traditionally been used to refer to a person, place, or organization. Later, the word was expanded to cover date, time, and quantity.

2.2 Ontology

An ontology is a formal representation of attributes, groups, and relationships of concepts in a domain. To enable such a description, components need to be specified as individuals (instances of objects), classes, properties, and relations as well as constraints, rules, and axioms. As a result, ontologies introduce a transferable and reusable knowledge representation as well as the ability to incorporate new domain-

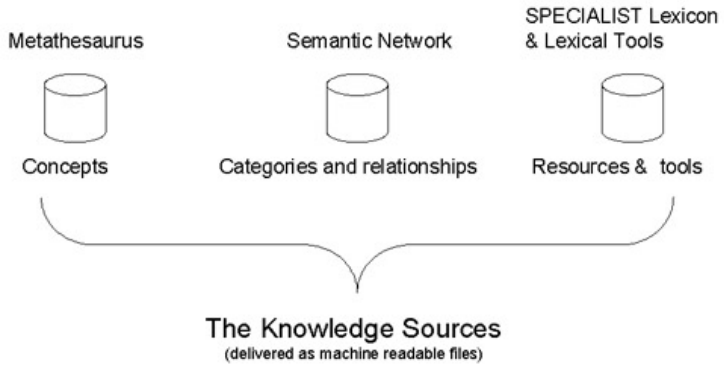


Figure 2.3: **UMLS Graph** Three UMLS Knowledge Sources are available: over one million biomedical concepts from more than 100 source vocabularies are included in the Metathesaurus.For labeling the biomedical domain, the Semantic Network defines 133 broad categories and 54 relationships between categories. The SPECIALIST Lexicon & Lexical Tools offer lexical information and language processing software.[3]

specific knowledge.
A knowledge graph, which is a collection of entities in which the types and relationships between them are represented by nodes and edges between these nodes, can be created by applying the ontology data model to a set of individual facts. The ontology creates the framework for the knowledge graph to capture the data in a domain by describing the structure of the knowledge in that domain.[2] Figure 2.4 generally depicts the meaning of an ontology.

2.3 Natural Language Processing (NLP)

NLP could be referred to as one of the branches of artificial intelligence (AI). With the help of statistical, machine learning, and deep learning, it mainly focuses on processing human language in the form of text or spoken words and trying to understand the full meaning of the speaker's or writer's sentence. Some tasks of NLP are:
Word sense disambiguation (WSD): With the help of semantic analysis, which chooses the word that is mostly correct in a given context, selects the correct meaning of a word among multiple meanings. For example, word sense disambiguation helps to differentiate the meaning of the verb 'make' in 'Just make sure our asses are protected' (to take special care about something) vs. 'make plans' (decide).
Named entity recognition (NER): Is the task of identifying words or phrases as

[2] `https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies/`

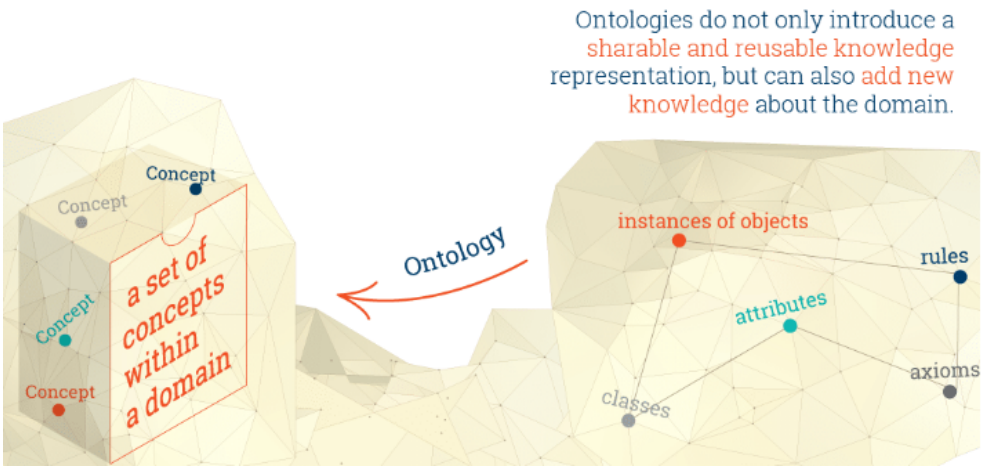


Figure 2.4: **What are Ontologies** A formal description of knowledge as a collection of concepts within a domain and the relations between them is known as an ontology. We need to formally identify elements like individuals (instances of objects), classes, attributes, and relations as well as constraints, rules, and axioms to enable such a description. As a result, ontologies introduce a reusable and transferable knowledge representation as well as the potential to incorporate additional domain-specific knowledge.[4]

valuable entities to help find the words that are the most informative in the sentence. NEM recognizes 'Laptop' as a device or 'Paris' as a location.
Sentiment analysis aims to make the most use of the meaning of a text, i.e., by taking out the subjective qualities, purposes, feelings, confusion, suspicion, and other elements from the text.

2.4 Named Entity Recognition

NER, also known as entity identification or entity extraction, is an NLP technique that automatically recognizes named entities in a text and organizes them into predefined categories. Entities include things like names of people, organizations, places, times and dates, quantities, monetary values, percentages, and more. As shown in Figure 2.5, specific mentions such as "WeWork" as an organization or "Adam Neumann" as a person are highlighted in the sentence.[**W**]ith named entity recognition, the essential information can be extracted to determine what a text is about, or simply used to gather crucial information to store in a database.

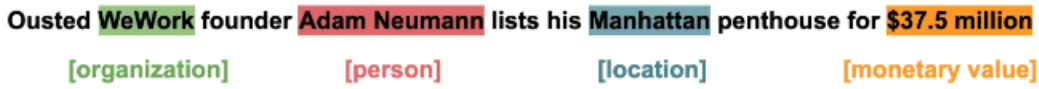


Figure 2.5: **Named entity recognition (NER)s**. With the use of NER, named entities in a text can be automatically recognized and categorized into predetermined groups. Entities include names of people, groups, places, dates, amounts, monetary values, percentages, and more. [5]

2.5 Entity Linking

Entity linking is the task of identifying mentions of named entities (or other terms) in a text document and disambiguating them by mapping them to canonical entities (or concepts) listed in a reference knowledge graph [116]. EL or named entity disambiguation is the process of linking or mapping entity names. To detect named entities from a text using the distinctive graph-based formal entity names in a knowledge base like Wikipedia, WordNet, and so on, EL employs parametric learning models. However, the parametric learning technique does not generalize the data well when there are more than a thousand formal entity names. Therefore, EL also employs deep learning structures to analyze entity names with Wikipedia references. In practice, many entity linking systems rely on the following resources or assumptions [102]:

- **Single entity set** This assumes that there is a single comprehensive set of entities E shared between training and test examples [102].

- **Alias table** An alias table contains entity candidates for a given mention string and limits the possibilities to a relatively small set. Such tables are often compiled from a labeled training set and domain-specific heuristics [102].

- **Frequency statistics** Many systems use frequency statistics obtained from a large labeled corpus to estimate entity popularity and the probability of a mention string linking to an entity. These statistics are very powerful when available [102].

- **Structured data** Some systems assume access to structured data such as relationship tuples (e.g., (Barack Obama, Spouse, Michelle Obama)) or a type hierarchy to aid disambiguation [102].

2.5.1 Two ways of Entity Linking

Solutions for entity linking can be divided into two categories [72, 84]:

- Local approaches [114]: Tries to disambiguate each mention in a document separately, utilizing clues such as the textual similarity between the mentioned context and the candidate entity information. local approaches handle mentions separately and employ local features for mention disambiguation, such as contextual words or named entities. Generally, candidates are ranked based on how similarly their feature vectors resemble those of the mentioned article. A drawback of local approaches is that they ignore the semantic correlation [107] between features and mentions, which can assist in resolving the feature sparsity issue. For instance, despite their semantic similarities, the spellings of Big Apple and New York City will make them less likely to be related.

- Global approaches [114]: Utilizes relations of entities in the context to estimate coherence. These methods[75, 87] perform entity linking on all mentions collectively while taking into account the semantic relatedness between mentions. Utilizing semantic similarity can give context information, for instance, connecting the USA to the United States of America will make it simpler to recognize the mention of "New York City". These methods help to determine an allocation for mentions that has the highest level of internal coherence among all potential assignments, as well as entities that are coherent with the mentions. Since it is an NP-Hard problem to find an assignment with the maximum possible coherence, all global approaches rely on approximation algorithms.

2.5.2 Biomedical Entity Linking

Biomedical entity linking aims to map biomedical mentions, such as diseases and drugs, in biomedical literature, to medical entities in a standard knowledge base such as UMLS [69]. The specific challenge in this context is that the same biomedical entity can have a wide range of names, including synonyms, morphological variations, and names with different word orderings[73]. In the healthcare field, accurate entity disambiguation is essential for understanding the biomedical context. Since many different biomedical concepts might be mentioned in very similar ways, failing to resolve these conflicts will result in the wrong perception of the whole context. This will greatly increase the risks involved in making medical-related decisions. Additionally, many other applications that require automatic text indexing can benefit from biomedical entity linking. For example, healthcare professionals can use it to automatically link a patient's medical record to various medical entities. These entities

can then be utilized for other activities including diagnosis and medical decision-making, health analytics and demographic trends, predictive analysis[68], medical information retrieval, feature extraction[88], and question answering[126]. In many cases, entity linking in biomedical literature is different from entity linking in other types of text. Consider the example in Figure 2.6, where Antihistamines are a mention of the entity Histamine Antagonists, and other entities are the top candidates found in the UMLS. The following are the common issues of entity linking in the biomedical domain: 1. The mentions can be misleading. In this example, almost every other candidate, in this case, has terms that properly reflect those in the mention. If we only employ surface-level features, it will be difficult to link the mention to the appropriate entity. As a result, the mention and its context must be fully understood by the model in terms of semantics. 2. The candidates may be similar to one another not just on the surface, but also in terms of semantic content. Additional information, such as fine-grained types, is therefore helpful in identifying the correct entity. 3. The mentions and context in medical entity linking are typically longer than those in the general domain, which presents another difficulty. As a result, linking medical content with traditional techniques is less effective. 4. Finally, there are several many domain-specific terms, abbreviations, and typos in the biomedical field. The efficiency of neural models is subsequently degraded as a significant proportion of concepts are outside the lexicon of popular pre-trained embeddings like GloVe[113].

2.6 Machine Learning

Machine learning (ML) is one of the subjects of artificial intelligence, which is widely considered the ability of a machine to mimic intelligent human behavior. Machine learning is used to accomplish the goal of AI, which is to carry out complicated tasks in a way that is similar to how humans solve issues. There are three main categories of machine learning:

- In **supervised** learning, labeled data are utilized to train algorithms to classify data or predict outcomes with greater accuracy. Supervised learning is used in many domains such as spam detection, face detection, signature recognition, and weather forecasting.

- In **unsupervised** machine learning, algorithms explore unlabeled data for patterns. In other words, unsupervised learning models can recognize patterns or trends without the need for human interventions. This strategy is perfect for

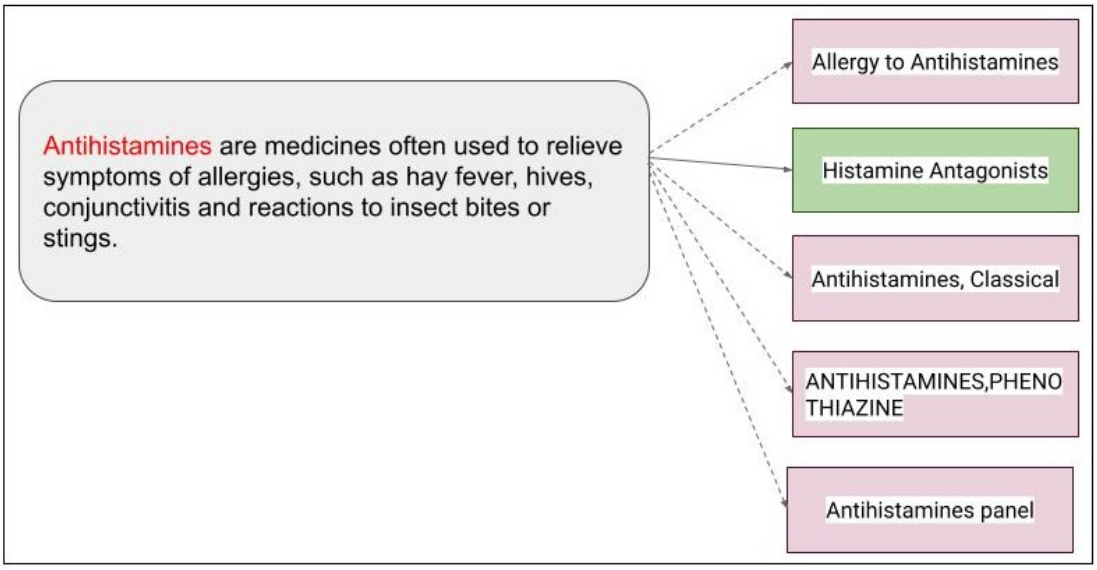


Figure 2.6: **An example of a biomedical entity, in this case, the highlighted word is the extracted mention, the pink boxes indicate the top candidate entities that were obtained from the biomedical as knowledge-base, and the green box represents the ground truth item for this mention.** The text in the given example is from [6]

exploratory data analysis, cross-selling strategies, market characteristics, and image and pattern recognition since it can find similarities and differences in data.

- **Semi-supervised** learning stands somewhere between the two. During training, it leverages a smaller labeled data set to enable classification and feature extraction from a larger, unlabeled data set. Semi-supervised learning can handle the problem of not having sufficient labeled data for a supervised learning algorithm. It is a particularly efficient strategy when labeling data is time-consuming or costly.

2.6.1 Deep Learning

Deep learning is a type of ML, which uses artificial neural networks and algorithms that are designed to mimic the functions of the human brain to learn from vast amounts of data. Figure 2.7 shows a general schematic of deep learning.

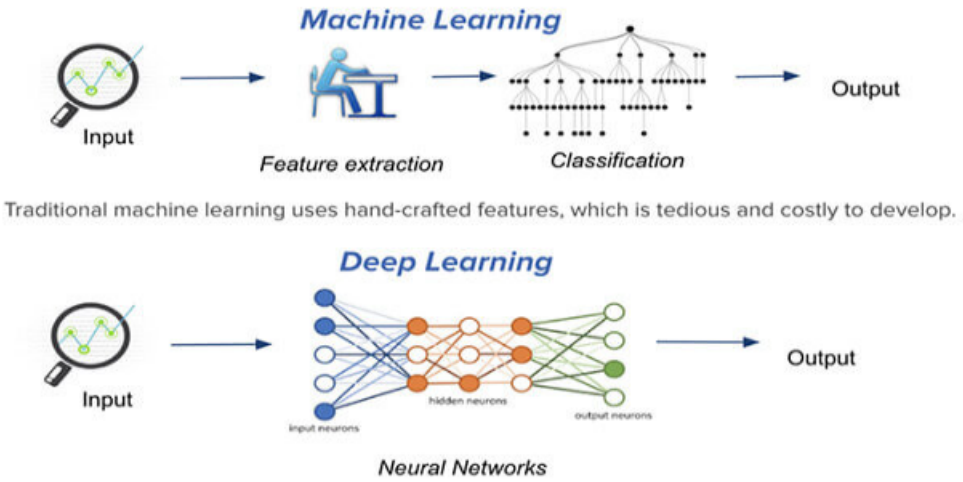


Figure 2.7: **Deep Learning learns hierarchical representation from the data itself, and scales with more data**. Before data can be fitted into a model that can make decisions based on these characteristics in a machine learning technique, an engineer or expert must design features that can distinguish between Target 1's and 0's. Handcrafting these features presents an inherent hurdle for engineers. A model's features would not need to be manually defined for deep learning because it does not necessarily require structured data. Here, each network structure chooses the particular traits or qualities that define the target. Now, a range of potential dangers, including potentially highly innovative ones, can be found using this model. [7]

How does deep learning work?

Deep learning is powered by neural network layers, which are algorithms based on human brain function. Training with large amounts of data is what creates the neurons in the neural network. The outcome is a deep learning model that, after being trained, can process new (unseen) data. Without the need for human intervention, deep learning models gather data from various sources and assess it in real time. Since graphics processing units (GPUs) can do several operations simultaneously, they are the best for training models. Most AI solutions that can enhance automation and analytical processes are powered by deep learning. People frequently come across deep learning, when they use their smartphones or the internet. Some general uses of deep learning are for example creating subtitles for YouTube videos, executing speech recognition on smartphones and smart speakers, performing facial identification for pictures, and enabling self-driving automobiles.

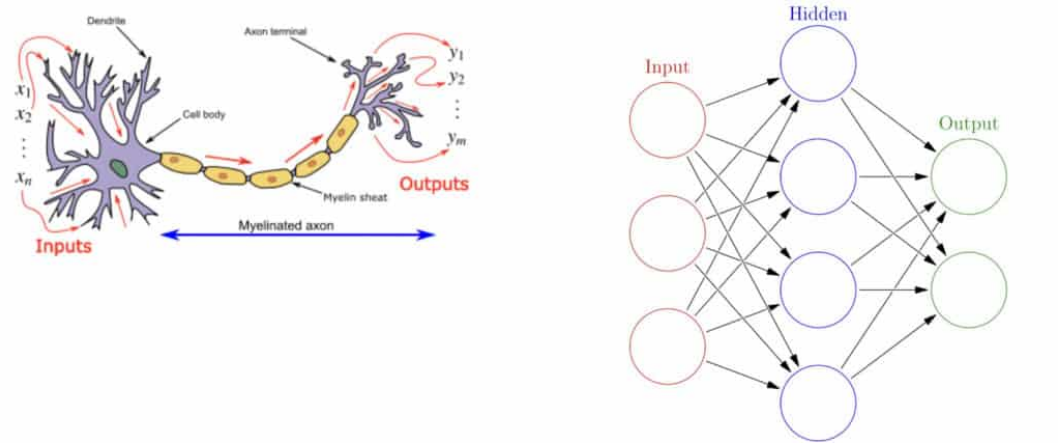


Figure 2.8: **Brain neuron, the left. Simple neural network on the right.**. Artificial neural networks are created to resemble how the human brain functions. The dendrites of the human brain serve as the artificial neural network's input, as shown in the image. The output of the neural network is then represented by the axion terminals. The network shown on the right only has one hidden layer. A deep neural network is a network that has numerous hidden layers.[8]

Introductory to (Deep) Neural Networks

Artificial neural networks are created to resemble how the human brain functions. As shown in the Figure 2.8, the dendrites of the human brain serve as the artificial neural network. The output of the neural network is then represented by the axiom terminals. The figure on the right shows a network that has only one hidden layer. A deep neural network can be defined as a network that has many hidden layers.

Convolutional Neural Network

Convolution Neural Networks (CNNs) are multi-layered artificial neural networks that are capable of extracting features from picture and text data. CNNs have mostly been utilized for computer vision tasks like object detection, image classification, and text extraction. Typically, a convolutional neural network has two main layers:

1. A convolution layer to extract features from the data.

2. A pooling layer to make the feature map smaller.

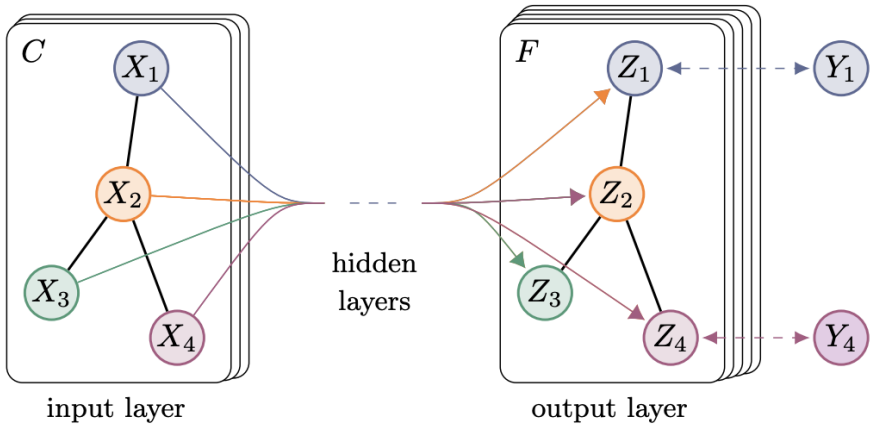


Figure 2.9: **Graph Convolutional Network's representation**, A multi-layer Graph Convolutional Network (GCN) with C input channels and $LARGEF$ feature mappings in the output layer is shown schematically. Layers share a common graph structure (edges are displayed as black lines, and labels are indicated by Y_i).[9]

Graph Convolutional Network

A Graph Convolutional Network (GCN), is a method for semi-supervised learning on graph-structured data. It is an effective version of convolutional neural networks that function directly on graphs. To design the convolutional architecture, a localized first-order approximation of spectral graph convolutions has been taken into account. The model learns hidden layer representations that encode both local network structure and node attributes and scales linearly as the number of graph edges increases.

2.7 Word Embedding

In general, word embeddings are a kind of word representation that links the human comprehension of language to that of a computer. Word embeddings have achieved text representations in an n-dimensional space, where words with the same meaning have similar representations. This means that two related words are represented by almost similar same vectors. As a result, when utilizing word embeddings, each word is represented by a real-valued vector in a specified vector space. Each word is mapped to a single vector, and the values of the vectors are learned in a way simulating a neural network. Word2Vec is one of the most popular methods for

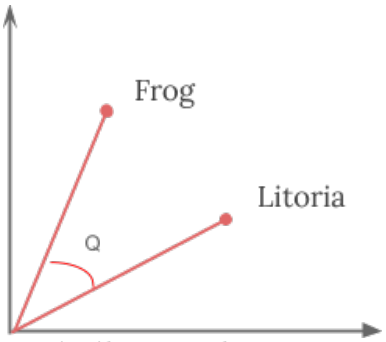


Figure 2.10: **In vector space, similar words are arranged closely together**. For Frog and Litoria, the angle q tends to zero [10]

learning word embeddings using basic neural networks.

2.7.1 Word2Vec

Word2vec uses a two-layer neural network to effectively create word embeddings. As input, word2vec gets a text corpus, which produces a set of feature vectors, or vectors that represent words in the corpus. Word2vec converts text into a numerical form that deep neural networks can understand, even if it is not a deep neural network. The goal of Word2Vec is to have similar word embeddings when the contexts are similar. These words are therefore quite close together in this vector space. According to mathematics, the angle (Q) between these vectors should have a cosine value that is close to 1, or close to 0. Figure 2.10 illustrates that for frog and Litoria, the angle q tends to be zero.

2.8 (BERT)

"BERT (Bidirectional Encoder Representations from Transformers) is a recent paper [78] published by researchers at Google AI Language. It has caused a stir in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks, including Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and others. BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modeling. This is in contrast to previous efforts, which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bi-directionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper,

the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible."[11]

2.9 Ranking in Information Retrieval

One of the essential issues in information retrieval (IR), the scientific and technical field that underlies search engines, is ranking a query. Given a query q and a collection D of documents that match the query, the goal is to order the documents in D according to some criterion so that the "better" results appear first in the list of results sent out to the user. Search engine queries and recommender systems are just two examples of how ranking in information retrieval is employed in many different contexts. To give users accurate and appropriate results, the majority of search engines use ranking algorithms.

2.9.1 Ranking Models

There are three types of ranking models: Boolean models (BIR), Vector Space Models, and Probabilistic Models.

Boolean Model

The Boolean model applies the AND, OR, and NOT conditions mentioned in the query to find all the related documents. For instance, given the sentence below, it returns only documents that include all occurrences of the terms alumni, graduates, and either company or college. By using this model, the search engine eliminates any documents that cannot possibly match the query.

```
alumni AND graduates AND (university OR college)
```

Vector Space Model

In a vector space, words, sentences, and even documents are represented numerically as a group of objects called vectors. Compared to a simple vector like map coordinates, which only have two dimensions, those used in natural language processing can have thousands of dimensions. A vector space model is an algebraic approach that treats entities (like text) as vectors. This makes it easy to evaluate word similarity or the relevancy of a search query and document. Cosine similarity is frequently used, to compare two vectors. To calculate the continuous degree of similarity between

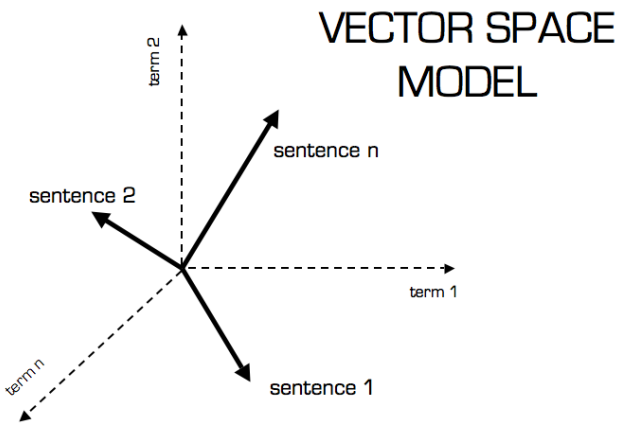


Figure 2.11: **Vector Space Model**. Documents are modeled as vectors using the Vector Space Model (with TF-IDF counts). As a result, we can calculate how similar several documents are in this area. [12]

two items, such as a query and documents, as well as partial matching, the vector space model uses linear algebra with non-binary term weights. Figure 2.11 shows a general understanding of this model.

Probabilistic Model

Probabilistic models simulate an event or phenomenon using random variables and probability distributions. A probability model provides a probability distribution as a solution, whereas a deterministic model provides a single possible outcome for an event. These models consider the reality that we can hardly ever know everything about a situation, and there is almost always some element of randomness to consider. For instance, health insurance is premised on the idea that, while it is known that a person will need his insurance (in case of a disease for example), the exact moment is unknown. These models can be entirely random, partially deterministic, or both.

2.10 Graph Theory

The area of mathematics known as graph theory studies networks of points connected through lines. The history of graph theory dates back to 1735 when the Swiss mathematician "Leonhard Euler" [13] found an answer to the Königsberg bridge [14]

dilemma. An old puzzle known as the Königsberg Bridge Problem attempted to find a path over each of the seven bridges that cross a branched river that flows by an island without having to traverse any of them more than once. Euler argued that such a road does not exist. His proof essentially proved the first theorem in graph theory, however, it only made passing reference to the physical arrangement of the bridges. The term "graph" referred to a set of vertices (also known as points or nodes) and edges (also known as lines) that connect the vertices. A graph is referred to as a multigraph when any two vertices are connected by more than one edge. A simple graph is one without loops and with no more than one edge between any two vertices. The graph is said to be complete when each vertex has an edge connected to every other vertex. A directed graph, also known as a digraph, is created by giving each edge a direction when it is appropriate to do so.

2.11 Similarity Measure

The similarity measure is a function that assesses how similar two phrases or things are to one another. Depending on the nature of the two objects, this function changes. The cosine similarity function, for instance, may be used if the objects are vectors. And the function could be Hamming distance[15], Levenshtein distance[16], or Jaro Winkler[17] distance function if the two items are strings.

2.11.1 String metric

A string metric (also known as a string similarity metric or string distance function) is a metric that estimates the distance between two text strings for estimating string matching or comparison. For example, the strings "Tom" and "Tommy" are similar. A string metric presents a number that indicates an algorithm-specific distance measurement.

2.11.2 Types of string similarity

String similarity algorithms can be categorized into several domains based on the properties of operations. Let us talk about some of them:

- Edit distance based: These algorithms aim to determine how many operations are required to change one string into another. The similarity between two strings decreases with the number of operations. One thing to keep in mind is that each index character in the string is given equal weight in this case.

```
>> textdistance.levenshtein('arrow', 'arow')
1
>> textdistance.levenshtein.normalized_similarity('arrow', 'arow')
0.8
```

Figure 2.12: **An example of the Levenshtein distance**. As is clear, if we add one 'r' in string 2 i.e. 'arow', it becomes identical to string 1. The edit distance is therefore 1. We can produce a bounded similarity score between 0 and 1, comparable to hamming distance. There is an 80% similarity rating. [18]

Some known edit distance-based algorithms are: the Hamming distance [15], the Levenshtein distance [16], and Jaro-Winkler distance [17]. Let us take a look at the Levenshtein distance as an example.

Levenshtein Distance

The number of modifications needed to change one string into another is used to calculate the Levenshtein distance. The permitted transformations are insertion, which adds a new character, deletion, which removes a character, and replacement, which swaps one character for another. The method attempts to change the first string to match the second one by carrying out these three operations. Finally, we are given an edit distance. As it is shown in Figure 2.12, string 2 becomes identical to string 1 if we add one "r" to make it "arow." The edit distance is therefore 1. We can produce a bounded similarity score between 0 and 1, comparable to hamming distance. The similarity rating is 80%, which is pretty a good score [18].

- Token-based: Instead of entire strings, a collection of tokens is requested as input in this category. The goal is to identify tokens that are similar in both sets. The similarity of sets increases with the number of shared tokens. By dividing a string with a delimiter, sets can be generated. In this method, a sentence can be broken down into tokens of words or n-grams. Note that tokens of all lengths are treated equally here. Some of the known token-based methods are the Jaccard index [19] and Sorensen-Dice [20]. In the following, the Jaccard index is explained with an example.

```
>> tokens_1 = "hello world".split()
>> tokens_2 = "world hello".split()
>> textdistance.jaccard(tokens_1 , tokens_2)
1.0
>> tokens_1 = "hello new world".split()
>> tokens_2 = "hello world".split()
>> textdistance.jaccard(tokens_1 , tokens_2)
0.666
```

Figure 2.13: **Jaccard index**. By default, we tokenize strings using spaces to turn the words into tokens. Next, we determine the similarity score. As both words are present in both strings in the first example, the score is 1. The score would be quite low if an edit-based algorithm were used in this situation. [19]

Jaccard Index

The formula, which belongs to the set similarity domain, is to calculate the proportion of common tokens to all unique tokens. [21]

$$Jaccard(U,V) = \frac{|U \cap V|}{|U \cup V|} = \frac{|U \cap V|}{|U| + |V| - |U \cap V|} \tag{1}$$

It is expressed mathematically, where the denominator is the union and the numerator is the intersection of the tokens (unique tokens). The second scenario is when there is considerable overlap, in which case we must eliminate the common terms because adding up all the tokens from both strings will result in their double addition. Tokens rather than whole strings are required as input, hence it is up to the user to tokenize their text effectively and wisely depending on the use case. To create tokens for the words in the strings in the example in Figure 2.13, we first tokenize the string using the default space delimiter. Next, the similarity score is calculated. As both words are present in both strings in the first example[19], the score is 1.

2.11.3 Shortest path algorithms

in graph theory, the shortest path algorithms [22] are employed to automatically determine the paths between two vertices (or nodes) such that the total of the weights of its edges is minimized. Dijkstra's algorithm, A* search algorithm, and Bellman-Ford algorithm are the most frequently used to solve this issue. In Figure 2.14, we discuss how Dijkstra's algorithm works.

Dijkstra's algorithm

Dijkstra is a graph search algorithm that builds a shortest path tree to solve the single-source shortest path problem for a graph with non-negative edge path costs. The algorithm determines the shortest path–or path with the lowest cost– between each vertex in the network and a specified source vertex (or node). It begins at the selected node (also known as the source node). The algorithm records the shortest path between each node and the source node that is currently known. If a shorter path is discovered, the path values are updated. A node is added to the path and labeled as "visited" when the algorithm determines the shortest path between it and the source node. Until every node has been added to the path, this process is repeated. The algorithm's output is a path that follows the shortest route between each node in the graph to connect the source node to every other node. The algorithm's pseudo-code is shown in Figure 2.14.

```
1  function Dijkstra(Graph, origin, destination):
2      for each vertex v in Graph:           // Initializations
3          dist[v] := infinity,    previous[v] := undefined
4      dist[origin] := 0                     // Distance from origin to origin is 0
5      Q := the set of all nodes in Graph // All nodes in the graph are unoptimized - thus are in Q

6      while Q is not empty:                 // The main loop
7          u := vertex in Q with smallest dist[]
8          if dist[u] = infinity:     break                    // There is no route from origin to destination
9          if u = destination:        break                    // reach the destination
10         remove u from Q
11         for each neighbor v of u:         // where v has not yet been removed from Q.
12             alt := dist[u] + cost_between(u, v)
13             if alt < dist[v]:             //If the distance is less than the previously recorded distance
14                 dist[v] := alt,       previous[v] := u

15     //read the shortest path
16     S := empty sequence
17     u := destination
18     while previous[u] is defined:
19         insert u at the beginning of S
20         u := previous[u]

21     return S
```

Figure 2.14: **Dijkstra's Algorithm Pseudo-code**[23]

Bellman-Ford algorithm

A graph algorithm called the Bellman-Ford algorithm [24] determines the shortest paths between a particular start node and all other nodes. Although this approach is similar to Dijkstra's, it can also control negative edge costs and spot negative cycles. The algorithm operates by initially overestimating costs from the start node to all other vertices and iteratively decreasing the cost as additional paths are discovered. It takes at most V-1 iterations to discover the shortest cost between the start and all other vertices since V-1 is the longest the shortest path may be. A negative cycle occurs in a graph when the sum of the edge costs is less than zero. If there are negative cycles, there can never be the shortest path because repeating the cycle could result in even lower numbers. Therefore, a negative cycle must exist if the cost drops after the previously completed V-1 iterations. The algorithm's pseudo-code is shown in 2.1:

Code Listing 2.1: **Bellman-Ford-pseudocode** [25]

```
1  function bellmanFord(G, S)
2    for each vertex V in G
3      distance[V] <- infinite
4        previous[V] <- NULL
5    distance[S] <- 0
6
7    for each vertex V in G
8      for each edge (U, V) in G
9        tempDistance <- distance[U] + edge_weight(U, V)
10       if tempDistance < distance[V]
11         distance[V] <- tempDistance
12         previous[V] <- U
13
14   for each edge (U, V) in G
15     If distance[U] + edge_weight(U, V) < distance[V}
16       Error: Negative Cycle Exists
17
18   return distance[], previous[]
```

2.11.4 Cosine Similarity

A measure called cosine similarity [26] determines the cosine of the angle formed by two vectors created in a multidimensional space. The two vectors are counted as similar if the angle between them is small. In the Figure 2.15, the angle between the two vectors A and B decreases as the cosine similarity value approaches 1, so A and

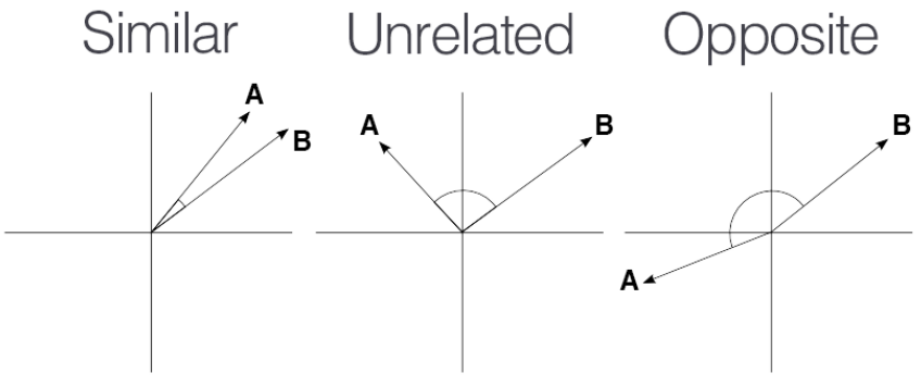


Figure 2.15: **Cosine Similarity**. The angle between the two vectors A and B decreases as the cosine similarity value approaches 1. A and B are more similar to one another in this situation. [26]

B are more similar to one another in this situation. The mathematical definition of cosine similarity is the division between the dot product of vectors and the product of the Euclidean values or magnitude of each vector. In Figure 2.15 A and B are vectors in a multidimensional space, respectively. Given that $\cos(\theta)$ has a value between [−1,1]:

- −1: this value denotes strongly opposing vectors or a lack of similarity.
- 0: denotes orthogonal (or independent) vectors.
- 1: implies that the vectors are highly similar.

$$\begin{aligned} \cos &= \frac{\overrightarrow{a} . \overrightarrow{b}}{\| \overrightarrow{a} \| \| \overrightarrow{b} \|} \\ \| \overrightarrow{a} \| &= \sqrt{a_1^2 + a_2^2 + a_3^2 + \ldots + a_n^2} \\ \| \overrightarrow{b} \| &= \sqrt{b_1^2 + b_2^2 + b_3^2 + \ldots + b_n^2} \end{aligned} \tag{2}$$

2.12 Search Engine

A typical search engine performs web searches to gather data relevant to the query. Some search engines can retrieve data from their data repository before even scanning the web. The indexed documents that are kept in a relevant database are searched by a search engine. The materials are indexed to improve the efficiency and effectiveness

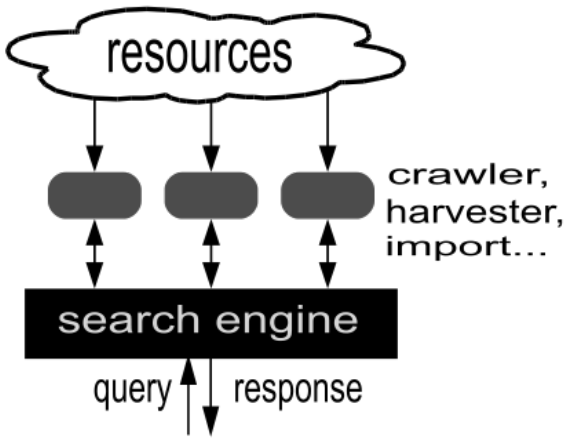


Figure 2.16: **Search Engine Diagram**. Utilizing its web crawlers, search engines scan hundreds of billions of pages. Search engine bots or spiders are frequent names for these web crawlers. By downloading online pages and using links on those pages to find newly available pages, search engines surf the web. The objective of the search engine algorithm is to provide a relevant group of excellent search results that will promptly answer the user's query or question. [27]

of finding the right documents. The Boolean model is used by search engines to match documents. A formula called the practical scoring function is used to determine relevance. This equation depends on the vector space model, Term Frequency (TF), and Inverse Document Frequency (IDF). In Figure 2.16 the concept of a search engine is illustrated.

2.12.1 The Elasticsearch engine

Elasticsearch [28] is a cutting-edge search and analytics engine that was first introduced in 2010 and is based on Apache Lucene. Elasticsearch is a Java-based, fully open-source NoSQL database. It cannot be queried with SQL since it maintains data in an unstructured way. It is a RESTful search and analytics engine that can be utilized to address an increasing number of use cases. Elasticsearch centrally stores data, making it possible to look for and analyze the data that has been indexed. It gives the results for each query together with the relevant relevance score. The value of the relevance score is influenced by several factors, including field length norm, inverse document frequency, and term frequency. We use the Elasticsearch python library to access the Elasticsearch engine, and we utilize the python library

```
from elasticsearch import Elasticsearch

es = Elasticsearch()
res = es.search(index="test", doc_type="articles", body={"query":
{"match": {"content": "fox"}}})
print("%d documents found" % res['hits']['total'])
for doc in res['hits']['hits']:
    print("%s) %s" % (doc['_id'], doc['_source']['content']))
```

Figure 2.17: **Code snippet of sending the query to the Elasticsearch engine**. This code snippet shows how can we interact with Elasticsearch using REST API, 'hits' means the number of returned results from Elasticsearch API call. [29]

Figure 2.17 to send the queries to the Elasticsearch server (it uses Restful API in the backend).

How does Elasticsearch work?

Elasticsearch [30] receives raw data from many different sources, such as logs, system metrics, and web applications. Before being indexed in Elasticsearch, this raw data is parsed, standardized, and enhanced through a process called data ingestion. Users can use aggregations to get complex summaries of their data once it has been indexed in Elasticsearch and can run complex queries against it.

What is an Elasticsearch index?

A group of related documents is referred to as an Elasticsearch index [30]. Elasticsearch uses JSON documents to store data. A set of keys (names of fields or characteristics) and their corresponding values are correlated for each document. An inverted index is a type of data structure that Elasticsearch uses to provide extremely quick full-text searches. Every unique term that appears in each document is recorded in an inverted index, along with the documents in which it appears. Elasticsearch stores documents and creates an inverted index during the indexing process to enable a fast search of the document data. The index API is used to start indexing, and it allows you to add or modify JSON documents in particular indexes.

What is Elasticsearch Score?

Elasticsearch uses the score to rate how relevant a match is to the query. Elasticsearch, which is based on Lucene, uses Lucene's built-in default scoring mechanism as its default scoring algorithm. This similarity model makes use of Term Frequency (TF) Subsection 2.12.2 and Inverse Document Frequency (IDF) Subsection 2.12.3 and the Vector Space Model (VSM) Section 2.9.1 for multi-term queries [31].

What is a fuzzy query?

Fuzzy query [32] returns information that is related to the search term in the documents, as determined by the Levenshtein edit distance. An edit distance is known as the number of one-character modifications required to change one term into another. These alterations could involve:

- Changing a character (man → fan)
- Removing a character (laptop → lapop)
- Inserting a character (tal → tall)
- Transposing two adjacent characters (flow → wolf)

The fuzzy query builds a set of all potential extensions, or variations, of the search phrase within a given edit distance to locate terms that are related. Then, exact matches for each expansion are returned by the query.

How to Use Fuzzy Searches in Elasticsearch

Fuzzy queries in Elasticsearch [33] do not require an exact match between the terms in the queries and the terms in the Inverted Index. Elasticsearch makes use of the Levenshtein Distance Algorithm to measure the distance between queries. The Fuzzy query feature in Elasticsearch [34] is an effective tool for many different situations. This unique query can frequently be used to fix username searches, misspellings, and other odd issues.

Executing a fuzzy query in Elasticsearch [33]

A common ElasticSearch search query can be executed similarly to a fuzzy query. Even though it is optional, it is crucial to provide the fuzziness parameter in the JSON request along with the maximum Levenshtein distance you wish to tolerate. Assume we have the typo word "Gppgle".

Code Listing 2.2: **Elasticsearch with exact match Query** [33]. As is shown in the code snippet, when we look for the exact match the Elasticsearch will return us just 1 successful case.

```
    ## Request
```sh
curl --request POST \
  --URL http://localhost:9200/fuzzy-query/_doc/_search \
  --header 'content-type: application/json' \
  --data '{
        "query": {
                "match": {
                        "text": {
                                "query": "gppgle"
                        }
                }
        }
}'
```
Response
```sh
{
  "took": 4,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 0,
      "relation": "eq"
    },
    "max_score": null,
    "hits": []
  }
}
```
```

When using a standard Match Query, Elasticsearch will first examine the query "gppgle" before searching for it. The only term that matches the term "gppgle" is "google," which is the single term in the inverted index. Elasticsearch will not generate any matches as a result. Let us attempt the fuzzy in Match Query in Elasticsearch now.

Code Listing 2.3: **Elasticsearch's fuzzy in Match Query** [33]. As is shown, when the fuzziness=AUTO, the number of returned result and the max_score increases, because we not only look for the exact match for the word, but also get all other words that are written similar with the word, for example, if the query="gppgle", we will have the 'hit'="google" as the related fuzzy result.

```
    ## Request

```sh
curl --request POST \
  --URL http://localhost:9200/fuzzy-query/_doc/_search \
  --header 'content-type: application/json' \
  --data '{
        "query": {
                "match": {
                        "text": {
                                "query": "gppgle",
                                "fuzziness": "AUTO"
                        }
                }
        }
}'
```
Response

```sh
{
  "took": 8,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 0.19178805,
    "hits": [
      {
        "_index": "fuzzy-query",
        "_type": "_doc",
        "_id": "w8YOCXUBHf9qB4Apc0Cz",
```
```

```
        "_score": 0.19178805,
        "_source": {
          "text": "google"
        }
      }
    ]
  }
}
```

As you can see, Elasticsearch responded to fuzzy by returning a result. Previously, we discovered that the distance between "gppgle" and "google" is 2. We used "fuzziness": "AUTO" instead of a number in the query. The reason it is working is that Elasticsearch will decide what fuzziness distance is acceptable if we use the "AUTO" value in the "fuzziness" field. Although "AUTO" fuzziness is preferred, if desired, we can tune it with an exact number.

2.12.2 Term Frequency (TF)

Term frequency [35] concept means the number of times a term occurred in a document. The simplest calculation is to count the occurrences of each word. However, there are ways to change that value based on the length of the document or the frequency of the term that appears most often.

$$TF = \frac{number\ of\ times\ the\ term\ appears\ in\ the\ document}{total\ number\ of\ terms\ in\ the\ document} \tag{3}$$

2.12.3 Inverse Document Frequency (IDF)

Inverse Document Frequency (IDF) [35] is a metric that shows how frequently a word is used. A low score indicates the word is frequently used across documents. The term is considered to be less important if it has a low score.

$$IDF = log(\frac{number\ of\ documents\ in\ the\ corpus}{number\ of\ documents\ in\ the\ corpus\ contain\ the\ term}) \tag{4}$$

2.13 Stop Words in NLP

Many words in the English language, such as "I," "the," and "you," are used frequently in texts yet do not offer any significant information for NLP operations and

modeling. These words are called stopwords[36] and they are almost always recommended to be eliminated as part of text preparation. When we exclude stopwords it results in a smaller text corpus and improves the performance and robustness of the NLP model. However, sometimes eliminating stopwords could have a negative impact if it modifies the sentence's meaning. For instance, we consider the example "This is not a good way to start" as an example of a negative statement. This sentence becomes a positive phrase when the stopwords are removed: "good way start".

2.14 Summary of the Chapter

This chapter covered essential concepts and technologies that help in understanding the motivation of this thesis, and the issue that we want to address. We also discussed some fundamental evaluation measures to understand the concept of similarity, how it can be adapted to our work, and the reasoning for our choice of measure.

Chapter 3

Related Work

We discuss earlier research that is related to our hypothesis in this chapter. This chapter is divided into four sections, and in each one, we discuss the most recent research about the highlights and drawbacks of each paper that is relevant to the elements of our framework.

3.1 Rule-Based Systems

There are previous works focused on rule-based methods for short text entity linking, in the following, we will discuss some of them: This paper [117] presents Falcon 2.0, an entity and relation linking tool that uses rule-based techniques to identify entities & relations in short texts and map them to the existing knowledge graph, such as DBpedia and Wikidata. Although there are several methods for linking entities and relations to DBpedia, Falcon 2.0 is one of the tools that specifically target Wikidata. Falcon 2.0 extracts several entities and relations from short input texts, and each extracted entity and relation in the output is linked to a particular Internationalized Resource Identifier (IRI) in Wikidata. It uses Background Knowledge (BK), a set of rules for executing entity and relation linking, which combines Wikidata labels with their related aliases. Falcon 2.0 analyzes a natural language text and uses three modules—POS tagging, tokenization & compounding, and N-Gram tiling—to identify the entities or relationships. The performance of Falcon 2.0 on sentences with little context is constrained, which is one restriction of the Falcon 2.0 method. Falcon is a rule-based approach and these methods have general limitations when the sentential context is sparse. Users-created nonstandard relations like "Which is CIQUAL 2017 ID for cheddar?" clearly fall outside the scope of rule-based approaches. In paper [81] a system named TAGME, which can efficiently and judiciously augment a

plain-text with pertinent hyperlinks to Wikipedia pages, is presented. The specialty of TAGME is that it may annotate texts which are short and poorly composed, such as snippets of search engine results, tweets, news, etc. This annotation is extremely informative, so any task that is currently addressed using the bag-of-words paradigm could benefit from using this annotation to draw upon (the millions of) Wikipedia pages and their interrelations. But its disambiguation strategy merely relies on a global voting mechanism of other mentions in the context, which incurs the same problem of global approaches, that they are not effective for short text entity linking. In [72] a comprehensive approach for short text entity recognition and linking is introduced. This approach is capable of detecting local topics in short texts and linking entities to a knowledge base with extremely little context. Authors introduce concepts of entities as explicit fine-grained topics to solve the sparsity and the noisy problem of short text. Although this study represents the state of the art in the field of entity linking for short texts, annotating datasets is labor-intensive, costly, and time-consuming because it depends on volunteers for some of the datasets. Authors of paper [77] have introduced a Sieve-Based approach for entity linking in the biomedical domain. They tried to advance the state of the art in normalizing disorder mentions in documents from two genres, clinical reports, and biomedical abstracts, so they proposed a sieve approach, which is composed of one or more heuristic rules. In the context of normalization, each rule normalizes (i.e., assigns a concept ID to) a disorder mention in a document. Sieves are ordered by their precision, with the most precise sieve appearing first. To normalize a set of disorder mentions in a document, the normalizer makes multiple passes over them: in the "i-th" pass, it uses only the rules in the "i-th" sieve to normalize a mention. If the "i-th" sieve cannot normalize a mention unambiguously (i.e., the sieve normalizes it to more than one concept in the ontology), the sieve will leave it unnormalized. When a mention is normalized, it is added to the list of terms associated with the ontology concept to which it is normalized. This way, later sieves can exploit the normalization decisions made in earlier sieves. Although this paper has the advantage of simplicity and modularity, it fails to resolve ambiguous normalization, where a disorder mention is mapped to more than one concept. The [74] paper is a novel approach to Wikification by incorporating, along with statistical methods, a richer relational analysis of the text. The authors have provided an extensible, efficient, and modular Integer Linear Programming (ILP) formulation of Wikification that incorporates the entity-relation inference problem, and demonstrates how finding relationships in the text is quite helpful for both candidate creation and ranking Wikipedia titles. However, since this paper is a labor-intensive approach, it requires a set of hand-crafted rules and a KB containing relations between entities. So on the one hand it is hard to generalize to

other languages which do not have such KBs, and on the other hand, it mainly relies on experts to design the rules.
Rule-based methods capture string similarity of mentions and entity names by using hand-crafted rules with manually assigned weights, string similarity algorithms, and domain-specific knowledge. While efficient, this method has some clear drawbacks. At first, semantic relatedness is not addressed when utilizing string similarity algorithms with no consideration for context. Second, defining a new set of rules and weights for each pair in the dataset may be required, which is usually a tedious and time-consuming task that hurts scalability. Our approach gets over these issues by employing similarity measures to identify semantic relatedness without defining rules over the dataset.

3.2 Machine Learning Approaches

Entity linking algorithms can be considered as two main categories: non-collective and collective inference techniques. Non-collective approaches often rely on prior popularity and context similarity with supervised models [105, 106]. Each concept mention's ranking score is calculated separately. On the other hand, collective approaches utilize the global coherence between concept mentions through supervised or graph-based re-ranking algorithms [96, 101, 83]. Collective inference methods handle the linking problem by maximizing the agreement between the text of the mention document and the context of the entities of the knowledge base. Graph-based re-ranking models typically obtain linking agreement information from training data and propagate the agreement information to other nodes. Both existing non-collective and collective algorithms require large amounts of a manually-labeled entity mention to reach high linking accuracy. The [124] paper introduces a conceptually simple, scalable, and highly effective BERT-based entity linking model, along with an extensive evaluation of its accuracy-speed trade-off. The authors present a two-stage zero-shot linking algorithm, where each entity is defined only by a short textual description. The first stage does retrieval in a dense space defined by a bi-encoder that independently embeds the mention context and the entity descriptions. Each candidate is then re-ranked with a cross-encoder, that concatenates the mention and entity text. This paper however is limited to working well when the cross-encoder possibly makes mistakes because of misleading context cues. In [82] a novel framework called Multi-turn Multiple-choice Machine reading comprehension (M3) to solve the short text EL from a new perspective: a query is generated for each ambiguous mention exploiting its surrounding context, and an option selection module is employed to identify the golden entity from candidates using the query. In

this way, the M3 framework sufficiently interacts with limited context with candidate entities during the encoding process, as well as implicitly considering the dissimilarities inside the candidate bunch in the selection stage, but its efficiency decreases when it wants to extract relations between entities or explicit type information. In [109] Babelfy is presented, which is a unified graph-based approach to EL and WSD based on a loose identification of candidate meanings coupled with the densest subgraph heuristic which selects high-coherence semantic interpretations. To lower the degree of ambiguity in the initial semantic interpretation of a directed graph G, a novel densest sub-graph heuristic was developed in this study. The fundamental argument is that the densest region of the graph will contain the meanings of each text fragment that are most suitable. Finding the densest sub-graph of size at least k is an NP-hard issue [80]. A 2-approximation greedy approach for arbitrary graphs [93] was the foundation for the author's definition of a heuristic for k-partite graphs. Their modified approach relies on the iterative removal of low-coherence vertices, or fragment interpretations, to choose a dense sub-graph of the G. The joint solution is based on three key steps: i) the automatic creation of semantic signatures, i.e., related concepts and named entities, for each node in the reference semantic network; ii) the unconstrained identification of candidate meanings for all possible textual fragments; iii) linking based on a high-coherence densest sub-graph algorithm. One limitation of this paper is that the semantic analysis of small contexts can be improved by leveraging the coherence between concepts and named entities. MedCAT is the open-source Medical Concept Annotation Toolkit (MedCAT) which is introduced in [95] provides a) a comprehensive self-supervised machine learning algorithm for identifying concepts by employing any concept vocabulary such as UMLS/SNOMED-CT.; b) an annotation interface with several features for modifying and training information extraction models.; and c) integrations to the wider CogStack ecosystem for vendor-agnostic organizations and health systems deployment.

In conclusion, machine-learning approaches to automatically learn the correct similarity measures between mentions and entity names from training sets are being presented to replace the necessity for manual rules. However, these methods are unable to identify keywords that are semantically related. In our approach, we address this issue by building a graph of relationships and employing similarity metrics to identify commonalities between mentions.

3.3 Hybrid Models

These models make use of rule-based models as well as machine-learning models. In [127] explore Knowledge-RIch Self-Supervision (KRISS) for entity linking, by leveraging readily available domain knowledge to compensate for the lack of labeled information. In training, it generates self-supervised mention examples on unlabeled text using a domain ontology and trains a contextual encoder using contrastive learning. For inference, it samples self-supervised mentions as prototypes for each entity and conducts linking by mapping the test mention to the most similar prototype. This approach subsumes zero-shot and few-shot methods, and can easily incorporate entity descriptions and gold-mention labels if available. One drawback of this paper is that its accuracy falls for some datasets in their experiment. In [115] ChemSpot, a named entity recognition (NER) tool for identifying mentions of chemicals in natural language texts is presented, which includes trivial names, drugs, abbreviations, molecular formulas and International Union of Pure and Applied Chemistry entities. Since the different classes of relevant entities have rather different naming characteristics, ChemSpot uses a hybrid approach, combining a Conditional Random Field with a dictionary. However, ChempSpot missed some entity mentions where either they were absent in the dictionary or the fact that they were not recognized by the CRF (a probabilistic undirected graphical model). A supervised Deep Learning ontology alignment method called VeeAlign was developed in [91] paper. It aligns classes and attributes based on context-driven structural similarity as well as semantic similarity. Thus, based on the type of neighborhood, the method integrates an innovative manner of modeling context by dividing it into different components. Some of these facets just take into account nearby one-hop nodes, while others additionally take into account the paths that lead from the root to these nodes, depending on their relative importance. The authors use a new dual attention technique that combines path-level attention and node-level attention to overcome this difficulty. The former aids in identifying the most crucial path out of all those that are available, whilst the latter identifies the node with the most influence on the alignment of the central concept. This paper has low recall due to the lack of incorporation of a translator in some datasets. As well as its performance decreases in property matching on the conference tracks.
Hybrid models combine rule-based and machine-learning models, therefore the effectiveness of both techniques affects the hybrid model's performance. Due to rule-based methods, hybrid models may be laborious, time-consuming, and domain-specific, but they also run the risk of performing poorly due to a lack of data needed to accomplish machine learning tasks. To address common problems of hybrid methods, we

use some heuristics such as string similarity in our benchmark, and also build the graph of relations in the pre-processing step.

3.4 Deep Learning Based Systems

Deep Learning (DL) has three key benefits in solving the EL task. First off, DL can use the given texts and KBs to automatically find several levels of distributed representations to improve the disambiguation without human interaction, whereas standard ML-based methods require substantial feature engineering and analysis. The second advantage of DL is that it is more transferable, allowing deep neural networks to learn representations that are more transferable and that separate the exploratory factors of variations underlying the data samples and organize features into hierarchical groups based on how they relate to invariant factors. A complex and advanced EL method may be motivated by the fact that DL may learn feature representations and accomplish classification or regression in an end-to-end manner. Graph representation learning is one of the most promising ways in various fields like classification, link prediction, and matching. Generally, graph learning methods extract relevant features of graphs by taking advantage of machine learning algorithms and achieving promising results in various domains over graph-structured data [85, 94, 121, 99, 125]. In [85] they presented GraphSAGE, a general inductive framework that leverages node feature information (e.g., text attributes) to efficiently generate node embeddings for previously unseen data. This paper lacks efficiency for directed or multi-modal graphs and does not perform for non-uniform neighborhood graphs. In [94] a scalable approach for semi-supervised learning on graph-structured data is presented, which is based on an efficient variant of convolutional neural networks which operate directly on graphs. This paper has some limitations like (1) a Memory Problem for large graphs, (2) did not support edge features, and is limited to undirected graphs. Authors of [121] proposed GAT, a novel convolution-style neural network that operates on graph-structured data, leveraging masked self-attentional layers. The graph attentional layer utilized throughout these networks is computationally efficient (does not require costly matrix operations, and is parallelizable across all nodes in the graph). Although this paper revealed promising results, it suffers to efficiently work on larger batch sizes. This paper [99] studied feature learning techniques for graph-structured inputs. The result is a flexible and broadly useful class of neural network models that has favorable inductive biases relative to purely sequence-based models (e.g., LSTMs) when the problem is graph-structured. The limitations of this work are that (1) the presented task translation does not incorporate temporal order of inputs or ternary and higher order relations, and (2) have

difficulties handling less structured input representations. In [125] authors study the graph-to-sequence problem, introducing a new general and flexible Graph2Seq model that follows the encoder-decoder architecture. They showed that using our proposed bidirectional node embedding aggregation strategy, the graph encoder could successfully learn representations for three representative classes of directed graphs, i.e., directed acyclic graphs, directed cyclic graphs, and sequence-styled graphs. However, the proposed approach does not perform well, a pooling-based graph embedding method like Graph2Seq-PGE performs better than Graph2Seq-NGE (node-based graph embedding). One potential reason is that the node-based graph embedding method artificially added a super node in the graph, which changes the original graph topology and brings unnecessary noise into the graph. This work [97] uses convolutional neural networks to structure biomedical entity linking as a ranking problem (CNNs). Two limitations of this paper are, candidate generation depends on handcrafted rules, which determine the upper threshold of the CNN-based ranking system, that is the percentage of accurate entities in all candidate sets. Also, the CNN-based ranking method completely disregards vague entity mentions. In paper [67] authors presented a novel method, SNERL, to simultaneously predict entity linking and entity relation decisions. SNERL can be trained without any mention-level supervision for entities or relations, and instead relies solely on weak and distant supervision at the document level, readily available in many biomedical knowledge bases. This method has difficulty when for example the candidate set itself is not accurate and since this method rely on the candidate set to filter the annotations for the documents, it might end up with significant annotations that are not present in the title and abstract. In [71] authors proposed a novel methodology, which models the latent type of mentions and entities, to improve the biomedical entity linking task. They incorporate this methodology in LATTE, a novel neural architecture that jointly performs fine-grained type learning and entity disambiguation. To the best of our knowledge, this is the first work to propose the idea of latent type modeling and apply it to biomedical entity linking. Since many KBs also fit under the broad definition of heterogeneous graphs, heterogeneous graph embedding has also recently attracted attention from the scientific community. In paper [119] authors introduce Relational Graph Convolutional Networks (R-GCNs) and apply them to two standard knowledge base completion tasks: Link prediction (recovery of missing facts, i.e. subject-predicate-object triples) and entity classification (recovery of missing entity attributes), however, their work is not well suited for modeling asymmetric relations. This paper [128], tries to solve the problem of heterogeneous graph representation learning and introduce a heterogeneous graph neural network model (HetGNN), which encodes the content of each node into a vector and then adopts

a node type-aware aggregation function to collect information from the neighbors. HetGNN also uses attention over the node types of the neighborhood node to get the final embedding. However, its performance can deteriorate when the neighbor size exceeds a certain number. The recommended neighbor size in the paper is 20 to 30.

Although deep learning algorithms are very robust in the field of natural language processing, they are quite expensive to train due to complicated data models and require very large amounts of data to outperform other approaches. Deep learning also needs hundreds of machines and expensive GPUs. The cost to users goes up as a result. In our approach, we do not use a large amount of data. In contrast to DL approaches, our method is therefore simpler to understand and does not require expensive machines to function.

3.5 Heuristic Approaches

A heuristic approach[37] is an approach that employs practical strategies based on prior knowledge. A heuristic method does not, however, ensure that it will produce the best result. It is applied When classic methods take too long to address an issue. Additionally, it is utilized to find approximations of solutions when the traditional approaches fail to yield an accurate result. Two examples of heuristic approach are search optimization [38] and traveling salesman [39] issue. Heuristic approaches find solutions more quickly than traditional methods since they are based on practical information that produce accurate answers, as opposed to traditional extensive ways that produces results but take a long time to complete.
According to this paper[90], entity linking can be adapted for new domains by using contextual knowledge related to the textual content to assess and the assignment that will employ the extracted information. The authors look into how contextual adaptation, or the combining of semantic knowledge from a domain-specific knowledge base with evidence from extra information sources (the text to analyze and the job to address), can improve the cross-domain performance of entity linking. By utilizing a set of contextual adaption heuristics, the suggested method seeks to improve the cross-domain performance of a hybrid language and graph-based entity linking module. These heuristics take into account the text's processing order, entity mentions' co-references, the topical domain's significance, and semantic type. One limitation of this paper is that the results are made worse by the semantic type heuristic. This is disappointing because authors believe that semantic typing gives the entity linking procedure a unique new perspective. However, in actual use, the semantic typing heuristic is susceptible to two significant elements that have a direct impact on how

well it performs: First, the hybrid module's automatic semantic typing of the entity about its textual context; second, the accuracy of the entity types in DBpedia[40] (if any). Paper [76] tries to solve the problem of entity linking in QUOTEBANK, which is a large collection of speaker-attributed quotes from the news, by utilizing heuristics that depend on simple signals in the context of mentions and the reference KB. The proposed heuristics could be considered as strong baselines for unsupervised entity linking in large datasets as they perform very well on QUOTEBANK, have minimal computational complexity, and achieve competitive performance on the AIDA-CoNLL benchmark. One limitation of this paper is that a term is not seen as important because they give it no weight to indicate its significance. So authors intend to utilize additional signals from the KB to remedy this issue.
We can classify our approach as a heuristic approach since we build the entire graph of particular relations in the knowledge base, traverse the graph in the following steps to determine the minimum distance between the nodes, and then prioritize the nodes with the smallest average distance as being the most relevant to one another. What makes our approach better are: 1. our approach performs efficiently in cases where mentions are ambiguous and in most cases can link the mentions to their correct counterpart in the knowledge base. 2. It does not need any training data in advance, which means in cases where there is no data at hand, it can perform well.

3.6 Summary of the Chapter

This chapter reviewed some work related to this thesis. Additionally, they were compared to the work of this project. The process of connecting entities mentioned in the text to their knowledge base equivalents is known as entity linking. Entity linking is a popular issue in real-world applications including information retrieval, content analysis, and question-answering, as well as in the research field. What we can conclude from this chapter is that there are different ways to entity linking and there is no ideal way to use entity linking. In some situations, rule-based models may perform better, whereas, deep-learning techniques may perform best in other scenarios. Because the context, entity types, knowledge base, datasets, and other metrics all play a role in entity linking.

Chapter 4

Approach

In this section, we first formalize the problem of entity linking for texts in the biomedical domain. Then we present, Lumos, the framework of our solution. In this thesis, we examine entity linking for the English language, rather than cross-lingual entity linking. The following Figure 4.1 and Figure 4.2 represent the architecture of our framework.

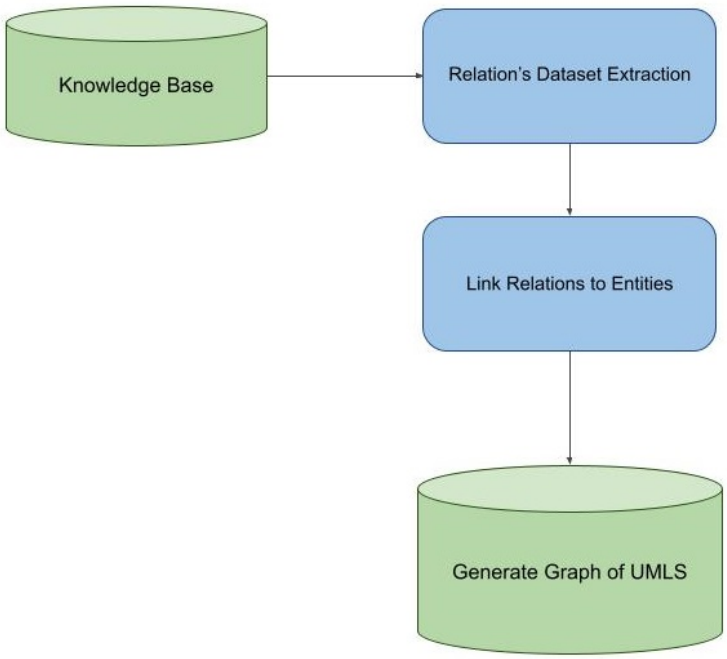


Figure 4.1: **Approach's Pre-Processing Step**. Extracting relations table from UMLS and building up the graph of relations in UMLS using Neo4J

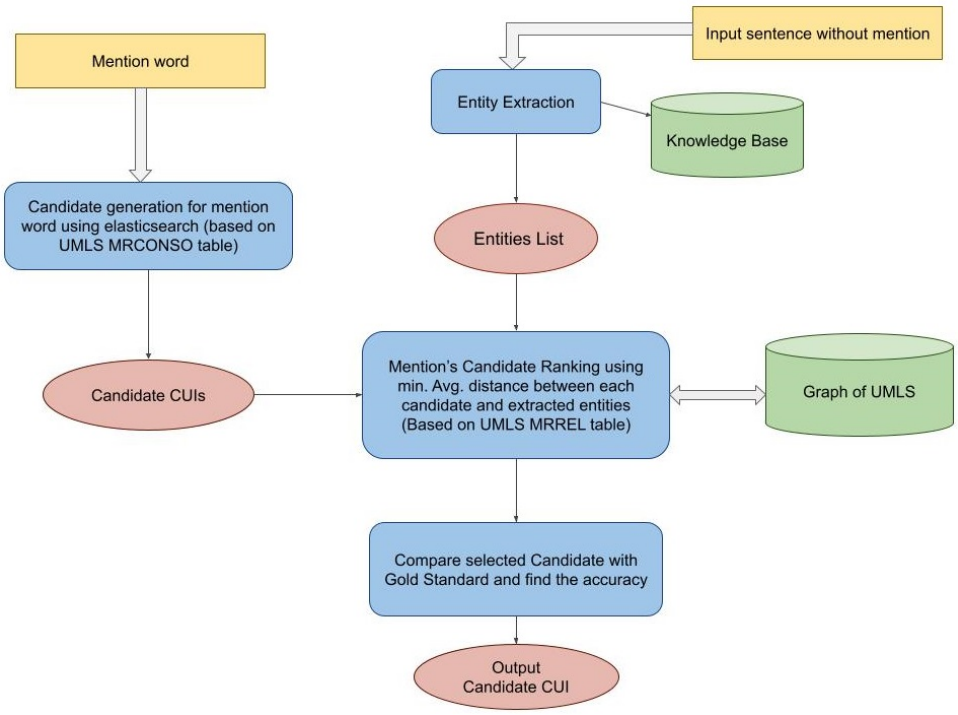


Figure 4.2: **Approach's Processing Step**. It consist of Lumos stages. It gets the input sentence (mention, input sentence without mention), Lumos approach then process the mention and entities throughout the stages. Finally, it gives a candidate CUI as the output for the mention.

4.1 Problem Definition

The problem in this work is twofold. The first part is to find entities in the text. Afterward, link each mention and entity to their counterparts in the knowledge graph. Then find the minimum average distance between the mention and other nodes, which have relations with each other. The problem is formally defined as follows: each document $D \in \mathbb{D}$, has a set of tokens $T^{(D)} = \{t_1^d, t_2^d, ..., t_N^d\}$, and the knowledge graph has a set of resources R. The task of entity linking is to map the set of tokens in the text t_k to the set of resources r_k in the knowledge graph. The mapping function is: $f : \mathcal{F}(t_k) \rightarrow \mathcal{F}(r_k)$. Let $acc(.,.)$ be a utility function that gives the linking determined by f a value of accuracy. let p be a function that represents the list of correctly associated sets of tokens (mentions) with resources/entities. The following optimization criteria should be resolved by the function :

$$\arg \max_{token \in \mathcal{F}(\mathcal{T})} acc(p(token), f(token)) \tag{5}$$

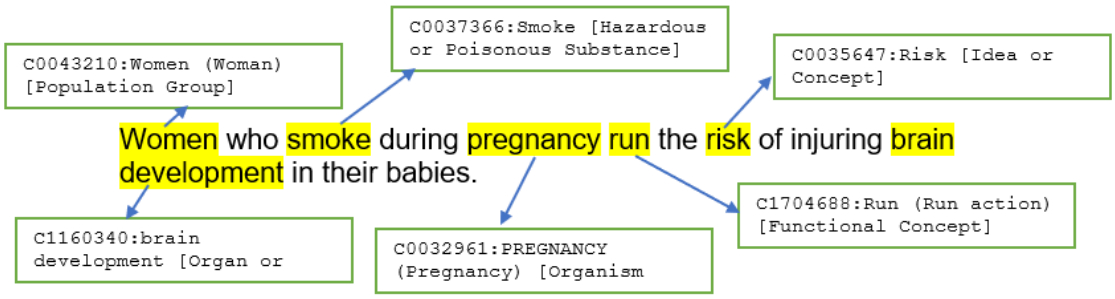


Figure 4.3: **Example of entity recognition approach.** Text from [41]. As is shown in the figure, with the help of an entity recognition tool, we can find the CUIs for each word, that exists in the entity recognition tool's database.

In the following 4.1 and 4.1, we explain in detail the problem of entity recognition as well as entity linking.

Named Entity Recognition

Named entity recognition is known as the task of finding significant nouns in the text. People, organizations, places, and all other entities with names are considered to be important nouns since they are essential for readers to comprehend the text's meaning. In a contrast to entity linking, named entity recognition simply is used for the identification of named entities in the text and the determination of their categories. It is not required for it to comprehend the meanings of these terms or to use a knowledge base to differentiate between them. While named entity disambiguation is regarded as a crucial step that can influence the outcome of entity linking when performing the task of entity linking [104]. In practical applications, the entity recognition step always comes before the entity linking task. It is our task to identify entity mentions in a text input and link them to the correct entities in the knowledge base. An example of entity recognition is given in Figure 4.3. the problem of entity recognition as well as entity linking.

Entity Linking

Most of the information we obtain is in the form of texts. These texts contain a large number of named entities, which compromise the fundamental components of texts. These entities, however, are vague, therefore we must link them to an existing knowledge base to help readers understand what the entities refer to and interpret the text more clearly. The process of disambiguating these extracted entities and connecting them to their counterparts in the knowledge base is called named entity

linking or entity linking. Entity linking is the task of linking named entities in texts to their corresponding entities in a knowledge base (Wikipedia [42], WordNet [43] and Unified Medical Language System(UMLS) [44]). For instance, given the text "Sarah eats an apple a day and enjoys the taste of it." entity linking will link the query mention apple to her corresponding entity apple (the fruit) in the knowledge base rather than apple (Apple Inc. technology company) using a variety of algorithms. In this thesis, we work on the biomedical named entity linking, which is extracting mentions from biomedical texts and linking them to their counterparts in biomedical knowledge bases. Biomedical Entity Liking is difficult due to the high ambiguity of entity mentions, which are [110]:

- Ambiguity: abbreviations are the biggest source of ambiguity. Depending on the context, a single abbreviation can be seen as two distinct things. For instance, "BCa"[45] stands for either breast cancer or bladder cancer.

- Polysomy: a term that can be used to describe several things. For example, "wing" might mean "one of the bird's wings is broken = parts of a bird for flying" or "the hospital is building a new wing = a new part of a building" [46].

- Synonyms: an entity can be denoted by multiple names or aliases. For example, Charles Dickens or Boz both refer to Charles John Huffam Dickens[47], who is a popular writer.

- Multi-word in the biomedical domain: most of the words in the biomedical domain are multiple words like follicular lesion of undetermined significance.

- Nested words: a word in the biomedical domain may be a part of a longer word. For example, "CBC (Complete blood count)" may occur in "controlling CBC" which is a laboratory check.

With the help of entity linking [123], we can retrieve information and disambiguate them in a way that the correct results are chosen for specific entities. In Figure 4.4 we illustrate the general schematic for entity linking. As shown each entity linking system consist mainly of three parts: generating a candidate set, disambiguating candidate entities, and linking results. In the following, we will describe these three parts. *Definition 1.* Entity linking's formal description: Given a text sequence s, the goal is to detect all the entity mentions in s as entity mentions $M = \{m_1, m_2, ..., m_N\}$ in document D and link each mention m_i to a single entity $\tilde{e}_i$ in KG or represent it as *NIL*, which indicates the mention m_i cannot be successfully linked to any corresponding entity in ε. We use $\varepsilon(m_i)$ to refer to ground truth entity of mention m_i and

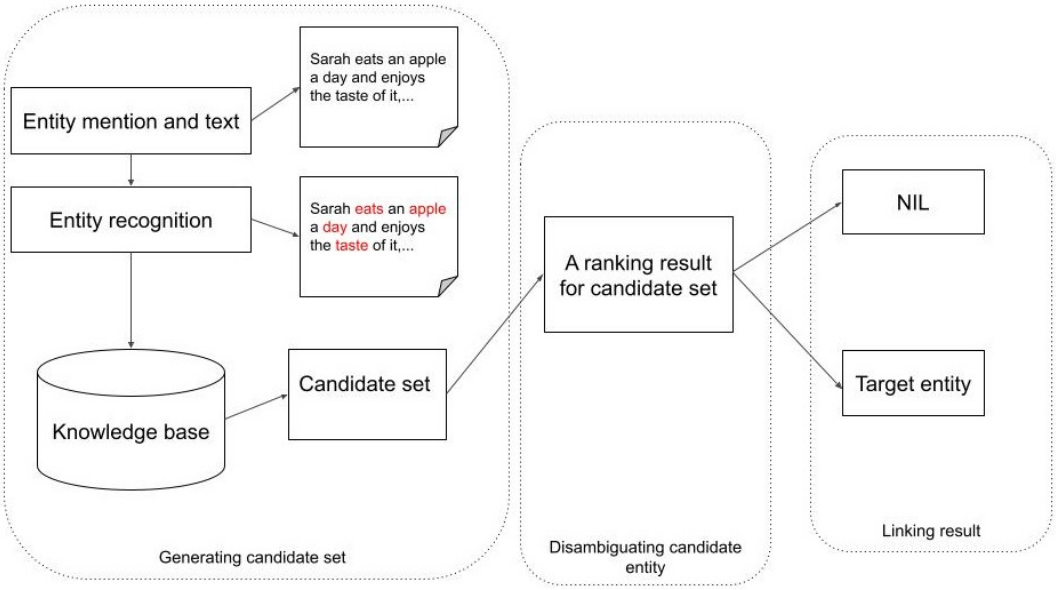


Figure 4.4: **A basic entity linking model** [123]. The basic model of an entity linking system consists of candidate entity generation, candidate entity disambiguation, and result selection.

$\hat{e}_i$ to refer to the predicted entity. In general, EL approaches often have two steps: entity recognition stage and candidate disambiguation stage.
Definition 2. An entity mention is a fragment of text that refers to an entity in the knowledge base.
Definition 3. A segmentation p of a text s is a sequence of text fragments

$$p = \{ti|i = 1, ..., l\} \tag{6}$$

such that:

- text fragments cannot overlap with each other.
- concatenation of all text fragments in p equals s.

The candidate generation module: Here, candidate entities for each query mention in the text will be identified. Once the named entity mentions are recognized using named entity recognition tools, discussed in Subsection 5.5.2, the candidate entities are then located in the knowledge base using the entity names in conjunction with additional features.
The candidate disambiguation module: This module is a crucial part of the entity

linking process. It uses several techniques that combine distinct entity properties to rank the candidate entities. Two features are used in the candidate disambiguation phase: the similarity between a query mention and the candidate entity's name and the context of the candidate entity.
The linking result module: In this module, the target entity is selected with the help of the ranking result of the candidate entity disambiguation phase. The system will return NIL when all candidate entities' scores fall below the threshold. However, the threshold is typically set manually.

4.2 Proposed Approach: Lumos

Lumos covers the problem mentioned in the problem definition. Our basic idea to tackle the problem of extracting and linking entities in the sentence is based on building a graph where the nodes are the mentions and entities, and the edges indicate the affinities between the nodes. We utilize an entity recognition tool to find entities in the text. After having a list of all entities with their corresponding CUIs from the KB, Lumos calculates the average distance for each mention and extracted entities and assigns a weight to each mention based on the distance to other nodes. Finally, Lumos chooses and links the best candidate (the one with the minimum score) to the mention and selects this candidate as the preferred output. To extract entities and relations from the text, Lumos employs a set of rules. The guidelines were taken from [118], and are based on English morphological concepts. For instance, the rule "Verbs are not entities" causes Lumos to exclude all verbs from the list of candidates for entities. Another rule that Lumos employs is the removal of stopwords from the candidate list of entities, as stopwords may negatively impact the linking task. Let us describe the Lumos method formally. We define the parameters 4.1 in Lumos as follows: As the initial step, we get the input text from document D and make a list of detected entities e with their CUIs $S = (S(D) \longrightarrow S(detected\,entities), S(CUI))$. Then Lumos eliminates the stopwords sw and verbs $\acute{e} = (e - sw + e - verbs)$. Next, the candidate generation function uses a search engine, to map each indexed-based mention to its corresponding CUI, $Q = q(mention) \rightarrow q(CUI)$. Then, Lumos find the distance between two nodes in the graph with W: $W = Avg.dist.(\acute{e}, Q, max\ score = 10)$. After that, we calculate the highest score difference score with a function N and assign the value to parameter B: $N = (ER - AR) \rightarrow B$. Afterward, Lumos ranks the candidates, in the way that, it adds up two values avg. dist. value from function W, and N, together: $K = (W + N)$. Next, in the candidate selection step, it chooses the min value from the previous step, $L = min(K)$. Finally, it compares the selected candidate with the gold_standard

Table 4.1: **Parameters used in Lumos**

Parameters	
Entity	e
Mention	m
Stopwords	sw
Document	d
Q(.)	candidate generation function
S(.)	entity recognition function
W(.)	calculte distance function
L(.)	candidate selection function
K(.)	candidate ranking function
ER	Elasticsearch highest score value
AR	highest score difference value
B	candidate value score for candidate from Elasticsearch
V(.)	function for adding up two values, avg. dist. value and B, together
Z(.)	compare the selected candidate with gold standard and find accuracy
Gold_Standard	the correct CUI

(the correct CUI) and find the accuracy: $Z = ((V(CUI), Gold\_Standard(CUI))$, it set the accuracy to 1, if $V(CUI) = Gold\_Standard(CUI)$, and accuracy = 0, if $V(CUI) \neq Gold\_Standard(CUI)$.

4.3 The Lumos Architecture

To achieve our goals, we divide our work into two parts. The pre-processing and processing components make up the two main parts of our framework. In addition to that, the sub-tasks of our approach start with the pre-processing step which is building the whole graph of relations in the UMLS knowledge base and with the help of the search engine, creating an indexed-based representation of the knowledge base. Here, it is important to highlight that our method is independent of any specific search engine. We could have used any search engine that can handle fuzzy query processing, which is discussed in Section 2.12.1, for this work, we chose Elasticsearch. After this step, an entity recognition tool is used to extract the entities and find, for each entity, a unique identifier in the knowledge base. Afterward, each word's relation is extracted and the candidates for the recognized entities are obtained by searching the index-based knowledge base. Then, the results are collected, and the task of ranking consists of the combination of two values. Finally, the last step is candidate disambiguation, in which we choose the best candidate based on the minimum score and select it as our preferred output. Also, the following Figure 4.5 represents the architecture of our framework.

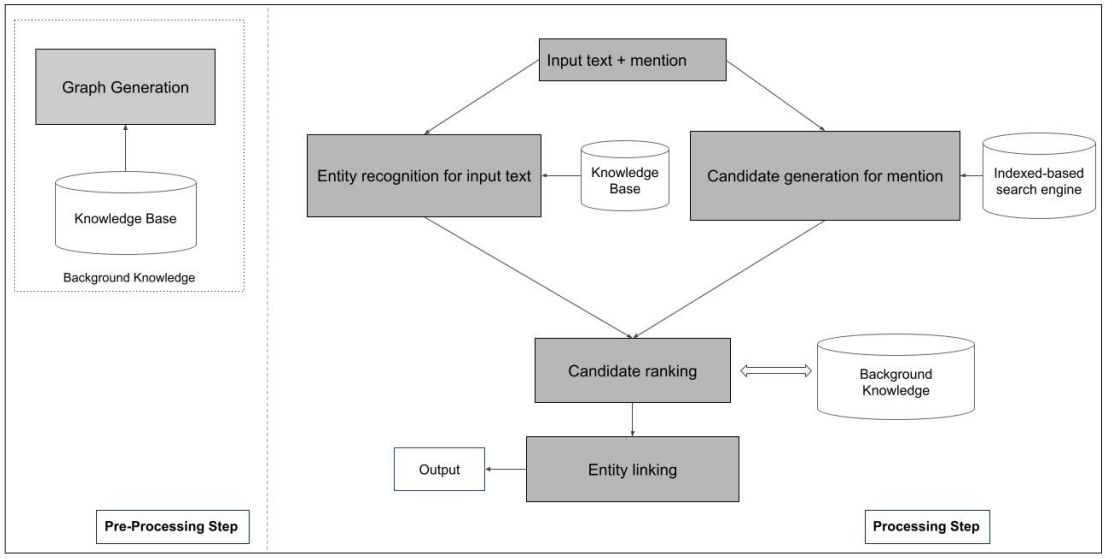


Figure 4.5: **Framework architecture**. This figure represents the whole architecture of the Lumos approach, as shown, the work is divided into two major steps: pre-processing and processing step.

4.3.1 Pre-processing

In this step, Lumos generates the whole graph of relations of the UMLS dataset and use an index-based search engine. Most of the relations and entity linking tools require a knowledge base that will be used to link the detected relations and entities. By generating a graph representation of the related concepts of the knowledge base for the linking task, we can search for relations between different nodes of the graph, i.e., look for a specific sub-graph in the whole graph, each time an entity is searched.

4.3.2 Candidate Recognition and Generation Stage

As the first step in EL tasks, since looking through the entire entity collection ε would incur an unacceptably high computational cost, a limited list of potential entities $C_i = \{e_{i_1}, e_{i_2}, ..., e_{i_m}\} \subset \varepsilon$ for the mention m_i is often created. In this step, we pass the text to the entity recognition tool and as a result of this step, the entity recognition tool gives us the keyword with its specific CUI. A candidate generation phase is used to limit the number of entities that are taken into consideration for a given mention, m, to a candidate set *(m)*. Here, we pass every recognized entity to our database, which we have configured in 4.3.1.

Removing Stop Words and Verbs

As discussed in Section 2.13, stopwords and verbs are not included in our methodology since they can deviate from the outcome of the entity linking and hurt the calculation of the average distance. In other words, by removing these words and not including them in the calculation of the average distance, we can link entities more effectively because these phrases no longer have an impact on the distances between the mention node and other candidate entities. The stopwords and verbs are identified as a result of the previous step, and in this phase, they are removed from the entity recognition tool's returned results to avoid influencing the further steps.

4.3.3 Collecting Results

We have so far looked for relations and entities in Elasticsearch. Now, we organize the results into a list of candidates, eliminate redundant candidates to speed up the execution of subsequent steps, and check to see if the ontology and property index has received any hits from Elasticsearch.

4.3.4 Candidate Selection

This stage consists of two sub-tasks, namely entity linking and candidate ranking.

Candidate Ranking

To increase the final accuracy, we decided to have ranked as the sum of two values. These two values are average distance and Elasticsearch the highest score difference. They will be combined and make up the final weight for each mention and entity.
Compute average distance: Here we compute the average distance from the heuristic approach proposed in this thesis. For each mention m_i, the retrieved set of candidate entities

$$Ck = \{c_1^k, c_2^k, ..., c_l^k\} \tag{7}$$

receives a score based on the shortest path length between the entities and the target mentions. The ranking function combines i) the shortest path length between the source and the target entity in the knowledge base, and ii) the maximum distance between the entities. We assign nodes with no affiliations to one another the maximum distance score, which is 10.
The code snippet for computing the avg Code Listing 4.1.

Code Listing 4.1: **Function to compute the average distance between two nodes**. Given two nodes this function calculates the shortest path between two nodes using the Dijkstra algorithm. We set the distance to 10 for the nodes having no relations with each other.

```
def find_dist(cui1,cui2,max_dist=10):
    try:
        dist=shortest_path_length(G,
        source=cui1, target=cui2,
        weight=None, method='dijkstra')
        return(dist)
    except:
        return(max_dist)
```

The search engine the highest score difference: In this step, after getting the highest score for the mention word from Elasticsearch output, we calculate the difference between the highest score and other candidates' CUI for the mention. Why do we need the highest score difference for each candidate and not the highest score? In Elasticsearch output, the more related results have higher scores, but in the other ranking parameter that we use (average distance), the lowest value is considered. So, we must convert the Elasticsearch score in a way that, the lower the better. Therefore, we will set the highest score as the zero weight, subtract other CUI scores from the highest score, and consider it as the weight for that CUI. For example, consider the mention is "temperature" and as the tables show: the top candidate is "C0005903" with the score "13.3073015". As we see in the 4.2, the highest score for the mention is considered to have the weight zero and, the result with a higher score have a lower score difference and can bring lower weight to candidate ranking.

Candidate Disambiguation

In this stage, after having the ranking scores for each mention, we sum up all the scores and make an average for each of them. Then, we group the mentions, which are close to each other (have the shortest average path in the graph) based on their relations. Finally, in our proposed approach, Lumos, we choose the lowest candidate score as the correct candidate or NIL in some specific circumstances. The output is the candidate for each mention.

Label	CUI	score	score difference
temperature	C0005903	13.3073015	0
temperature	C0039476	13.3073015	0
swinging temperature	C0277800	12.002837	1.3044645
temperature regulation	C0412806	12.002837	1.3044645
joint temperature	C0427304	12.002837	1.3044645

Table 4.2: **Example of Elasticsearch_score**. In this table, the score = "13.3073015", which is the highest score, is returned for the word "temperature", so we consider the CUI= "C0005903" as the correct result for the specified word. We then subtract other results from the highest score, to have a value that the Lumos approach will use in the ranking stage (the other parameter used in the Lumos approach, is the minimum average distance, i.e., the lower the distance, the more related it is.) So we need to convert the value from the highest score, to have for both: the lower the value, the more related it is.

4.4 Summary of the Chapter

In this chapter, we gave a formal definition of entity linking and discussed how we would address the issue in the biomedical field. We described each process' function and demonstrated our approach with a running example to help for better understanding.

Chapter 5

Implementation

This chapter goes into extensive detail about how we put our approach into practice, the development process we used to develop our framework, and its design. We present a running example of our methodology.

5.1 Software Methodology

The software methodology we used to build our framework is Spiral Software Model. The Spiral framework concentrates heavily on management and risks awareness. Following this structure, development teams work in spirals that are repeated until the final project is ready to be released. The project is continuously improved over time in small chunks, and teams keep updating it according to their new learnings. Each spiral model [48] has four main distinctive quadrants:

1. Planning and identifying requirements: This entails analyzing system needs and determining resource availability (time, effort, and cost).

2. Risk analysis: Analyze all the potential risks associated with the solutions, such as cost and schedule delays. The development of a strategy for reducing risk follows.

3. Development and testing: in this step the implementation, test, and deployment tasks take place.

4. Review and plan: At this point, users are taking part in the testing and giving feedback on the solution. Teams subsequently use that feedback to plan the future spiral.

Following Figure 5.1 visualize the model and the corresponding steps.

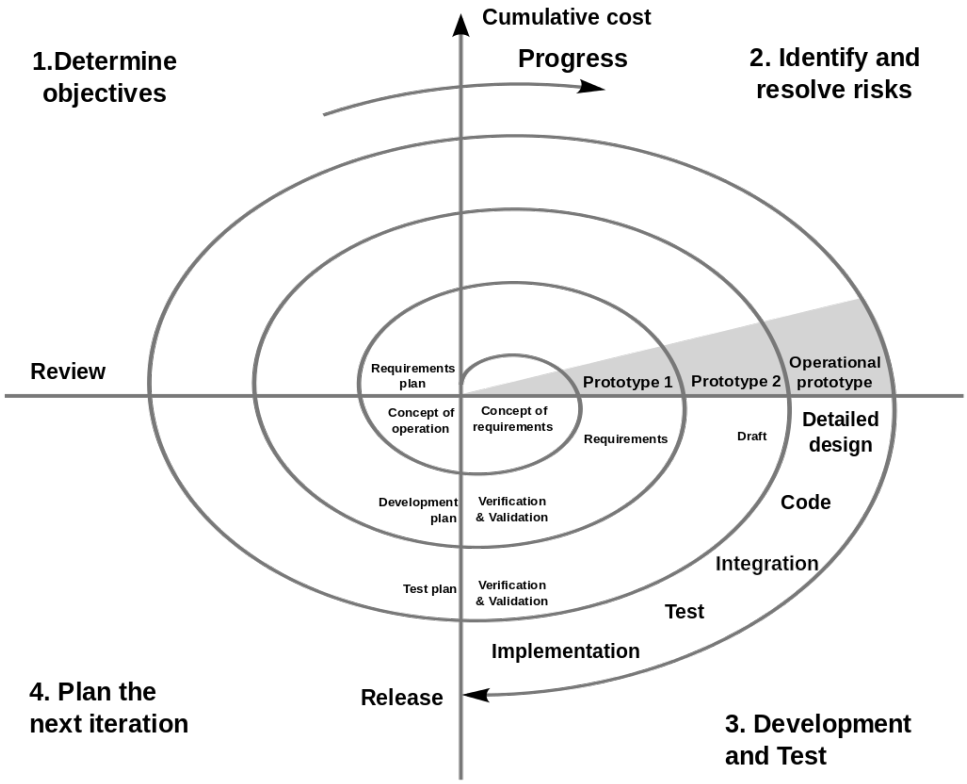


Figure 5.1: **The Spiral Model**[49]. It captures the key elements of the spiral model, including cyclic concurrent engineering, risk-driven process and product selection, risk-driven system growth, and cost-saving early exclusion of unworkable alternatives and rework avoidance. [70]

5.2 Design, Structure, and Dependencies

A detailed description of the implementation is presented in this section. The most important dependencies should be listed first. We used Python because it is simple, highly interpreted, and compatible and has Built-in Data Structures, as it is the dominant language for machine learning—the majority of machine learning frameworks are built in Python. Following Figure 5.2 depicts the approach with an example we are following in this thesis. We used NumPy [50] and pandas [51], as they both support high-performance matrix calculation and are capable of executing a vectorized code[86], have a python-compatible syntax, and can be utilized for high-level data analysis. We used two tools to create the graph and visualize it. The graph is built using Networkx, as discussed in Section 5.4, which also calculates the distances be-

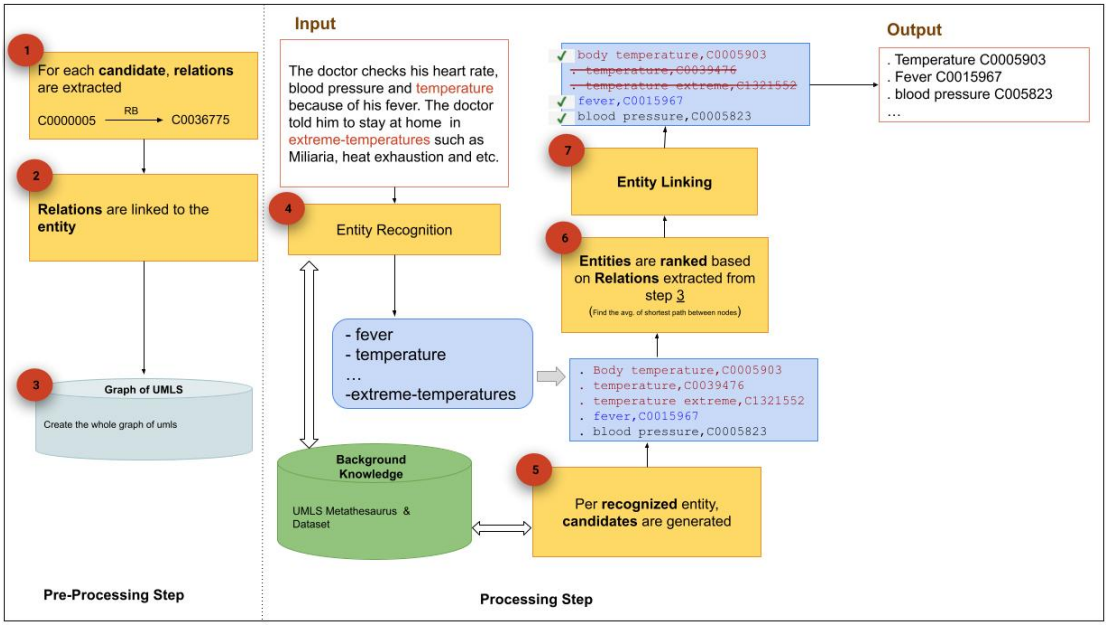


Figure 5.2: **Proposed Approach with Example**. Given our motivating example to the Lumos approach, we aim to find correct CUIs for the mentions "temperature" and "extreme-temperatures". As the result of the Lumos approach, we get as the output a list consisting of mentions and entities with their corresponding CUIs.

tween the nodes. Although Networkx has significantly faster query times, we utilized Neo4j [52], which is discussed in Section 5.5 to display the graph because Networkx is not primarily a drawing tool and does not provide a GUI for doing so.

5.3 Approach's Workflow

5.3.1 Graph Generation

To build the graph of relationships, we extracted the MRREL table from the UMLS knowledge base and created indexes with Elasticsearch, the output is a JSON file that contains the labels for each UMLS entity. For the task of relation extraction, we are only interested in Parent (or RB) and Child (or RN) relations. So we created a CSV file of relations that only contained the RB and RN relation to minimizing the size of the graph. After that, we built the graph in NetworkX Section 5.4 using this CSV file and utilized Neo4j Section 5.5 for graph visualization. In the following, these two tools are introduced:

5.4 NetworkX

NetworkX [53] is a Python module for building, modifying, and researching the structure, dynamics, and objectives of complicated networks. It provides common graph algorithms as well as data structures for graphs, digraphs, multigraphs, and multidigraphs. It allows you to create and evaluate network structures, load, and store networks in a variety of data formats, generate different kinds of random or conventional networks, and much more. We can use pathfinding algorithms with NetworkX. For instance, calculating the shortest path, which is discussed in Subsection 2.11.3.

5.5 Neo4j

Neo4j is a graph database management system developed by Neo4j, Inc.[54]. A graph database has nodes edges and properties, instead of rows and columns. It is more effective in many use cases, including some big data and analytics applications. A graph database could be used to represent relationships. The most common instances are relationships on social media. Figure 5.3 is an example of calculating the shortest path length between the word "Heat Stroke", as the source node with the CUI='C0018843' and the word "Anemia", as the destination node with the CUI='C0002871'. The nodes that are in the middle, show us that the two entities are connected through these nodes. RB relations are displayed in blue lines and RN relations in pink lines. According to 5.2, we see that the shortest path length between the two nodes in the given example is 4. We explain, shortly, why we used the shortest path algorithm for finding relations between graph nodes.

Code Listing 5.1: **Example of relations**

```
from neo4j import GraphDatabase

driver = GraphDatabase.driver("neo4j://localhost:7687"
,auth("usr","pass"))
session = driver.session()
result = session.run
("match s=shortestPath((src{CUI:'C0018843'})
-[*]-(dst{CUI:'C0002871'}))return s")
record = result.single()
print(record["s"])
session.close()
driver.close()
```

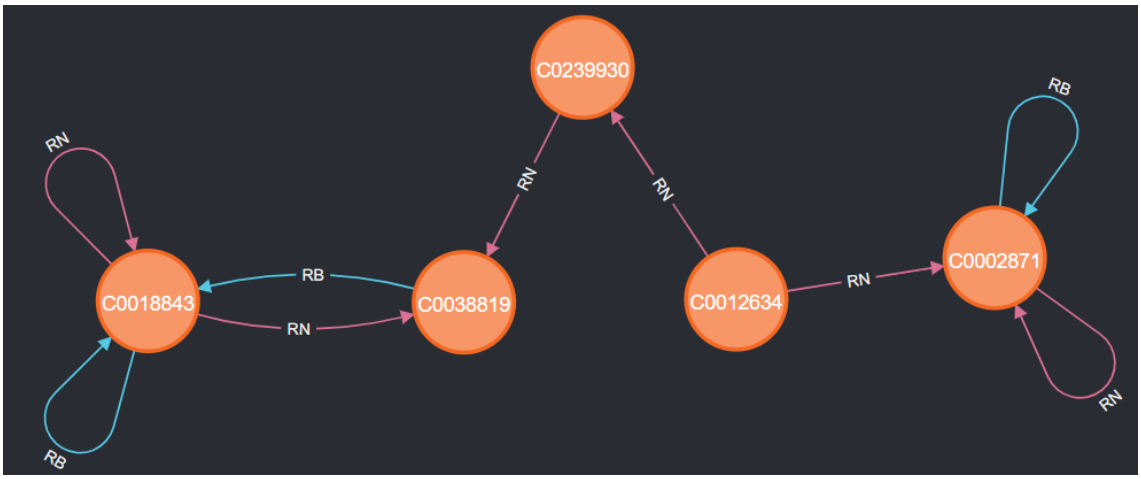


Figure 5.3: **Example of the graph of relations between two entities**. As shown in the figure the entity with CUI = "C0018843" and entity with CUI = "C0002871", are related to each other via other nodes and edges in between. And the distance between these two nodes = 4.

Code Listing 5.2: **JSON format of result for the given example Figure 5.3**

```
{
"start": {
"identity": 27496,
"labels": [
    "Concept"],
"properties": {
"CUI": "C0018843"}},
  "end": {
"identity": 2327,
"labels": [
      "Concept"],
"properties": {
"CUI": "C0002871"}},
  "segments": [
    {
      "start": {
"identity": 27496,
"labels": [
          "Concept"],
"properties": {
"CUI": "C0018843"}},
      "relationship": {
"identity": 120246,
"start": 62652,
```

```
"end": 27496,
"type": "RB",
"properties": {
"RELA": ""}},
      "end": {
"identity": 62652,
"labels": [
          "Concept"],
"properties": {
"CUI": "C0038819"}}},
    {
      "start": {
"identity": 62652,
"labels": [
          "Concept"],
"properties": {
"CUI": "C0038819"}},
      "relationship": {
"identity": 414432,
"start": 411580,
"end": 62652,
"type": "RN",
"properties": {
"RELA": ""}},
      "end": {
"identity": 411580,
"labels": [
          "Concept"],
"properties": {
"CUI": "C0239930"}}},
    {
      "start": {
"identity": 411580,
"labels": [
          "Concept"],
"properties": {
"CUI": "C0239930"}},
      "relationship": {
"identity": 37384,
"start": 14431,
"end": 411580,
"type": "RN",
"properties": {
"RELA": ""}},
      "end": {
"identity": 14431,
```

```
"labels": [
          "Concept"],
"properties": {
"CUI": "C0012634"}}},
    {
      "start": {
"identity": 14431,
"labels": [
          "Concept"],
"properties": {
"CUI": "C0012634"}},
      "relationship": {
"identity": 37362,
"start": 14431,
"end": 2327,
"type": "RN",
"properties": {
"RELA": ""}},
      "end": {
"identity": 2327,
"labels": [
          "Concept"],
"properties": {
"CUI": "C0002871"}}}
  ],
  "length": 4.0}
```


5.5.1 Read the Input Text

Our framework takes the dataset in JSON format as input. The input text will be passed to our first service, which is entity recognition. Since our benchmark is case-sensitive and this case-sensitivity can have an impact on accuracy results, we additionally lowercase our input text.

Removing Noisy Characters

We need to eliminate all the irrelevant symbols from the text to prepare it for processing. Noisy symbols are recognized as characters that are connected to the words or that come at the end of the text. To save the text from having to undergo further processing in the subsequent steps, it is beneficial to eliminate these characters. For instance, if we leave the apostrophe s in the sentence "Is sugar bad for human's health? " because it was not in the original entity label, it will cause an issue when

linking the entity to a human and could reduce the accuracy of the correct matching. We use Python string replacement to identify and remove the noisy characters. The following characters are considered noisy characters and should be excluded from the text: the question mark "?", the stop symbol ".", the exclamation mark "!", apostrophe s "'s", the single quote "", space, etc.

5.5.2 Entity Recognition

For named-entity recognition, which is explained in Section 4.1, we have used three tools, to evaluate which tool performs better on our dataset. In the following, each tool is explained:

MedCATTrainer

The Medical Concept Annotation Tool (MedCAT), is a Named Entity Recognition + Linking (NER+L) tool for identifying and linking clinical text concepts to existing biomedical ontologies like UMLS or SNOMED-CT — often a first step in extracting knowledge from the vast amounts of unstructured plain text available in clinical EHRs.
MedCATTrainer as a web-based interface (illustrated in Figure 5.4) can be used for biomedical NER+L models through active learning, to:

- Inspect concepts that have been identified and linked by MedCAT.
- Improve a MedCAT model by providing human annotators with some supervised training examples.
- Customize and collect further use-case-specific annotations training or specific models.

MetaMAP

MetaMap [56] is a tool for recognizing UMLS concepts in a text, i.e., It maps biomedical text to the UMLS Metathesaurus or it helps to find Metathesaurus concepts that are mentioned in the text. MetaMap employs a knowledge-intensive methodology based on symbolic, natural-language processing (NLP) and computational-linguistic approaches.
MetaMap [56] is frequently used for Information extraction, Classification/categorization, Text summarization, Question answering, Data mining, Literature-based

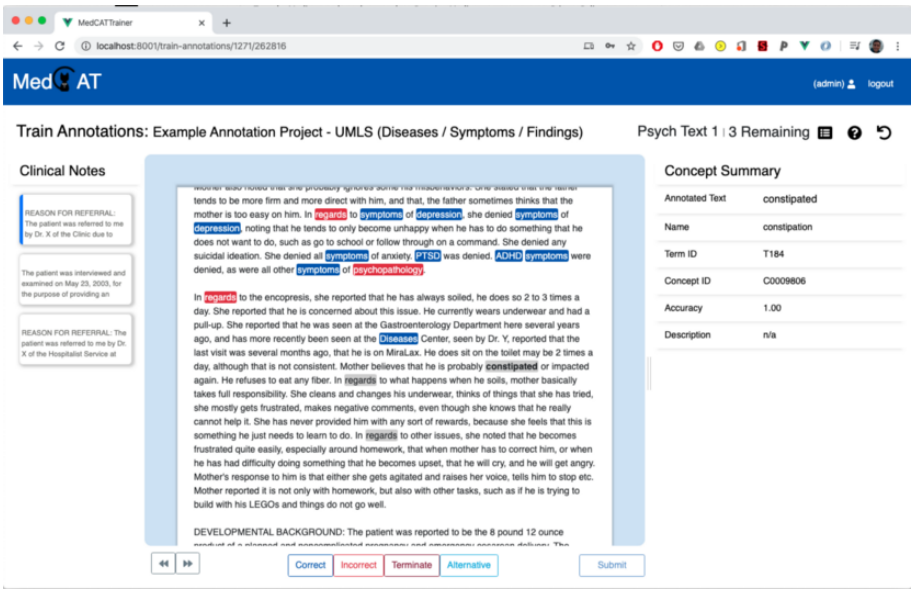


Figure 5.4: **The Annotations Interface of MedCATTrainer**[55]. The interface displays the clinical text currently being reviewed, the currently selected concept details taken from MedCAT, as well as a document summary and its annotation status.

discovery, Text understanding, UMLS concept-based indexing, and retrieval, Natural-language analysis of biomedical literature and clinical text.

BioFalcon

BioFalcon [118] is a relation and entity linking framework. It uses an API in two ways:

- Relations and entities linking in a sentence
- Entities linking in a short text or term

The output is the top K results that are retrieved from the API request.

Ranking Method for Candidates of Entities After testing all three tools on our test dataset, we got the best results from MetaMap Section 5.5.2. Since MetaMap is a fully customizable tool created at the National Library of Medicine (NLM) to map biomedical texts to the UMLS Metathesaurus or to find Metathesaurus concepts mentioned in the text, so, we decided to use MetaMap as our primary entity recognition tool and tested the MedMentions on MetaMap. The output from the entity recognition step is all detected entities, that have CUI in a list in this format:

[[entity1, CUI1], [entity2, CUI2], [entity3, CUI3]]. In this step, we check the entities, and if there are stop words like "to, from, which, never, etc." in entities, we will eliminate them by using the stopwords list as a reference from Spacy [57] python library. We also eliminate the verbs, because they can have a negative impact to find the correct candidates. We use the Spacy [57] library to exclude the verbs.

5.5.3 Generating candidates and calculation of scores

After having the mentions identified by one of the entity recognition tools, Lumos uses Elasticsearch to get all the CUI that is referred to a candidate, as discussed in 2.12.1. It searches the mention in Elasticsearch[3] and finds the first 10 records (CUIs) for it. Elasticsearch assigns each record a score, we discussed Elasticsearch score in Section 2.12.1. The search engine returns the CUIs for each mention. We store these candidates to be ranked later. We decided to use the Elasticsearch engine (as discussed earlier, we could have used other indexed-based search engines) to find the mention and its CUI. Two advantages of using Elasticsearch are, on the one hand, the searching process in big datasets can be done very fast. And on the other hand, we are able not only to get the exact candidate for the searched entry but also to get similar candidates as well. The total performance of entity linking models depends on the recall of

[3]We load in the pre-processing step, the MRCONSO tables from UMLS in Elasticsearch. MRCONSO is the table that contains all Atoms and CUIs' names and codes.

Code Listing 5.3: **ElasticSearch Function**. This figure shows, how the query is sent to Elasticsearch via REST API, and the 'hit' is the results, that are returned from the API call.

this step.

```
def search_ELASTICSEARCH(string):
    string=string.strip().lower()
    lst=[]
    client = Elasticsearch(["http://localhost:9200"])
    response = client.search(
        index="cui",
        body={
            "query": {
                "bool": {
                "should": [{"match": {"label": string}},
                {"match": {"lang": "ENG"}}]

            }
            }
            ,"size":10
        })
    for hit in response['hits']['hits']:
        lst.append(hit['_source']['cui'])
    return (lst)
```

5.5.4 Entity Linking

In the final step, the entity linking is a combination of two values: the first one is the calculation of the Elasticsearch highest score difference, which is discussed in 4.3.4, for all returned records from Elasticsearch (mention's CUIs). The second one is the average distance between the source node and other entities. The final weight for each candidate is the sum of these two values. Lumos chooses the lowest distance average record (CUI) as the candidate for the mention word.
The heuristic method used in this thesis is that, for each entity, we choose the correct candidate by finding the lowest weight for the candidate and all other candidates. In the end, we select and rank the candidate which has the lowest sum of average distance and Elasticsearch score difference among all others. We calculate the distance between the nodes from the relations table (mrrel). This method could help us find better results for the candidate node. The less the distance between nodes, the better it is, because that means the two entities are more related to one another and have shorter edges connecting them. We try to explain this with the help of an example and compare it with another example, where distance is not calculated:

Shortest Path with the Dijkstra's Algorithm

This algorithm [58] could be used to find the shortest path between nodes in a graph. You can specifically determine the shortest path between a node (referred to as the "source node") and every other node in the graph. The algorithm keeps a record of the shortest paths that are currently known between each node to the source node, and it updates these values whenever a shorter path is found. Once the algorithm has identified the shortest path between the source node and the target node, that target node is added to the path and tagged as "visited". Until every node in the graph has been added to the path, the process is repeated. In this manner, a path that connects the source node to every other node and takes the shortest path to each node is generated.

5.6 Summary of the Chapter

In this chapter, the approach described in Chapter 4 is implemented in a Python-based library. Within this chapter, we discussed the software methodology used in this work and provided a thorough explanation of how each phase of Lumos functions. The visualization part is implemented based on the packages provided by the Neo4j library. What we have learned in this section: we learned three recognition tools and implemented the basis of Lumos in all of them. We decided to choose for the heuristic approach the MetaMap entity recognition tool, regarding the number of correct outputs returned by this tool. We realized how stopwords and verbs can deteriorate our preferred result and how to handle them. We learned how to draw the graph and make the database in Neo4J and import the data from UMLS raw files in it. We get to know different forms of software development. We got to understand different techniques for running queries in Elasticsearch, how to configure Elasticsearch, and connect to the Elasticsearch server through HTTP protocol. We learned different methods of calculating the shortest path distance in the graph, like Dijkstra [58], A* search algorithm [59] and Bellman–Ford algorithm [60] and decided to use Dijkstra in this work. Finally, through various challenges we faced during implementation, we learned how to think progressively and, in a solution-based manner.

Chapter 6

Experimental Evaluation

To evaluate our work, we are going to test our framework in terms of the system's accuracy. In this chapter, we start by explaining our experimental setups, and after that, we explain the dataset utilized for this evaluation. We conclude by presenting the returned results for the accuracy and execution time for the given dataset. We attempt to answer the following research questions: **Q1)** How encoding the context of a text can contribute to entity linking tasks? **Q2)** How might knowledge encoded in domain-specific knowledge bases be applied to enhance the task of entity linking? **Q3)** what is the effect of representing relations among concepts in a knowledge base as a graph? Could it be used to measure entity similarity?

6.1 Experimental Setup

We used a virtual machine on Microsoft Azure [61] for the evaluation. The virtual machine is a Linux server with 4 CPU cores and 16GB RAM. It hosts the Elasticsearch engine and our framework in MetaMap. It connects to the Elasticsearch server locally.

6.1.1 Metrics

In our evaluation, we employed an accuracy metric to get the number of correct entities to the overall number of entities or relations in the dataset. Accuracy is known as the degree to which a calculated or observed value resembles the real value. It estimates the statistical error by comparing the measured value to the actual value. The range of such values reveals the measurement's accuracy. Accuracy is a key factor for many professionals, including analysts, mathematicians, and scientists, who use

it to assess the reliability of their data. For instance, by comparing experimental results to the accepted measurement standard, a scientist might assess the validity of their findings. Accuracy has the following definition[62]:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{8}$$

6.1.2 Benchmarks

MedMentions [108] is a publicly available dataset consisting of the titles and abstracts of 4, 392 PubMed[63] articles. The dataset is manually labeled by annotators and has labeled mention spans and entities linked to the 2017AA full version of UMLS. The dataset contains 352, 496 mentions, and each mention is specified with a single Concept Unique Identifier (CUI) and one or more semantic types indicated by a Type Unique Identifier (TUI). Additionally, MedMentions offers a random 60% - 20% - 20% split of the corpora into training, validation, and test sets. The test dataset contains 12% of concepts that are excluded from the training and validation sets. The concepts related to at least one mention in the MedMentions dataset make up our target KB for this dataset. We selected MedMentions as our main dataset, primarily because we wanted to understand the role of domain knowledge (UMLS is the underlying knowledge base) in the Lumos approach. Both entities from the training set and zero-shot entities are present in the validation and test sets (never seen at training time). We employ the author-recommended ST21pv subset, as suggested by [108], which includes more than 42% of entities in test time. Please take note that we ran our model on the test set to give the final findings because our model does not require any training with sentences and associated entities. To compare fairly with baselines, we employ both seen and unseen settings.

BC5CDR (BioCreative V CDR corpus) The BC5CDR, discussed in [98], is another biomedical entity linking benchmark, which has a corpus made up of 1, 500 PubMed articles in English containing 4, 409 annotated chemicals and 5, 818 diseases, that have been evenly divided into training, validation, and test sets. Each entity annotation uses Medical Subject Headings (MeSH) as the target vocabulary and contains mention text spans as well as normalized concept identifiers. This dataset additionally has 3, 116 chemical-disease relations in addition to entity linking annotations.

6.1.3 Baseline

Scispacy: Based on the reliable spaCy library [57], authors in [111] introduced scispaCy, which is a specialized NLP library for processing medical texts. In this paper, authors train spaCy models for POS tagging, dependency parsing, and NER using datasets relevant to the biomedical text. They also improve the tokenization module with new rules. "en_core_sci_sm" and "en_core_sci_md" are the two core released packages found in scispaCy. Models in the "en_core_sci_md" package have a bigger vocabulary and include word vectors, while those in "en_ core_sci_sm" have a limited vocabulary and do not include word vectors.
TF-idf: which is primarily utilized for candidate retrieval models [92, 65].
BIO-SYN: Introduced in [120], which is a novel approach for learning biomedical entity representations based on entity synonyms. The likelihood of all synonym representations in the top candidates is maximized by this method, which learns biomedical entity representations using the synonym marginalization technique and iterative candidate retrieval. To capture morphological and semantic information, the authors use both sparse and dense representations to represent each biomedical entity.
SAPBERT: Introduced in [100], which is A self-alignment pre-training technique for learning biomedical entity representations. The proposed method seeks to resolve the model entity relations issue (especially synonymy). The authors develop this framework for metric learning that learns to self-align synonymous biomedical entities. Both UMLS and task-specific datasets can be used with the approach for pre-training and fine-tuning.
INDEPENDENT: In this study [103], a zero-shot entity linking task is proposed, where mentions must be connected to hidden entities without the use of labeled data from the domain. No metadata or alias tables are assumed because the aim is to enable robust transfer to extremely specialized domains. In this scenario, entities are exclusively recognized by text descriptions, and models are forced to resolve new entities only based on language understanding. The authors demonstrate how strong reading comprehension models may be applied to generalize entities that have not yet been observed. The authors then go on to suggest a quick and efficient adaptive pre-training technique they call domain adaptive pre-training (DAP) to deal with the domain shift issue associated with linking unseen entities in a new domain.
CLUSTERING-BASED: This paper [65] tries to find relationships between the entity mentions. It suggests a model in which linking decisions can be made by clustering numerous mentions together and jointly producing linking predictions in addition to simply linking to a knowledge base entity.

6.1.4 Experiment 1

In the first experiment, we used MedMentions. We use MetaMap as our entity recognition tool and We achieve an accuracy of 53.71%. We performed a series of ablation studies to comprehend the effects of model choices and parameters. Also, the result of ablation studies shows us, first: There is no big difference between different algorithms for calculating the shortest path distance. The accuracy for both Dijkstra and Bellman-Ford algorithms is the same. Bellman-ford performs faster than Dijkstra using MedMentions. Second: when we compare the result of Metamap and Biofalcon, as two different entity recognition tools, we see that the accuracy for both is similar. Lumos reaches a little bit higher accuracy when it uses Metamap.

Ablation Study 1.1: using just the search engine

use Elasticsearch: to see how our heuristic method functions and evaluate its efficiency in finding the correct entity for the mention, we repeat the experiment using just the Elasticsearch function. The accuracy result for this experiment is 50% .
The reason why we got not have very high accuracy for Elasticsearch is that, as we explained earlier in this section, our proposed approach, Lumos, looks for the top 10 results of Elasticsearch and uses them as candidates for the mention. In some cases, the gold_standard value is not in the top 10 Elasticsearch results. This causes our approach not to be able to detect the correct CUI for the mention word. The next limitation is that we have used MRCONSO raw files from UMLS to load data in Elasticsearch. MRCONSO (the file, which contains all CUIs and labels) is not complete when it is compared to the UMLS web interface, and this causes Elasticsearch not to find all CUIs.

Code Listing 6.1: **Code snippet of using Elasticsearch with the top one result**. This is a function when Elasticsearch returns the size = 1, i.e. Elasticsearch returns the top result.

```
def search_ELASTICSEARCH(string):
    string=string.strip().lower()
    lst=[]
    client = Elasticsearch(["http://localhost:9200"])
    response = client.search(
        index="cui",
        body={
            "query": {
                "bool": {
                "should": [{"match": {"label": string}},
                {"match": {"lang": "ENG"}}]

            }
            }
            ,"size":1
        }
    )
    for hit in response['hits']['hits']:
        lst.append(hit['_source']['cui'])
    return (lst)
```

Ablation Study 1.2: Use another shortest_path distance algorithm

To test Lumos with another algorithm for calculating the shortest path distance, we used the Bellman-Ford algorithm, which is introduced in 2.11.3. We calculated accuracy and the run_time. We obtain an accuracy of 53.71%.

Ablation Study 1.3: Use another entity recognition tool

To find out, if our method is dependent on an entity recognition tool, we have utilized BioFalcon, which is introduced in Section 5.5.2. We achieved an accuracy of 53.53%.

6.1.5 Experiment 2

In this experiment, we used the BC5CDR dataset, which is a smaller dataset. We evaluated the accuracy of Lumos in three different ablation studies. We got an accuracy of 59.46%. We performed a series of ablation studies for the second experiment to understand the effects of model choices and parameters, as well. Also, the result

of ablation studies shows us, first: In this experiment also, there is no big difference between different algorithms for calculating the shortest path distance. The accuracy for both the Dijkstra and Bellman-Ford algorithms is the same. The Dijkstra algorithm performs three minutes faster than the Bellman-Ford using MedMentions (this difference in execution time is Negligible). Second: when we compare the result of Metamap and Biofalcon, as two different entity recognition tools, we see that the accuracy for both is similar in this experiment as well. Lumos reaches a little bit higher accuracy when it uses Biofalcon.

Ablation Study 2.1: using just the search engine

Elasticsearch: To see how our heuristic method functions and evaluate its efficiency in finding the correct entity for the mention, we repeat the experiment using just the Elasticsearch function. The accuracy result for this experiment is 53.3%.

Code Listing 6.2: **Code snippet of using Elasticsearch with the top one result**. This is a function when Elasticsearch returns the size = 1, i.e. Elasticsearch returns the top result.

```
def search_ELASTICSEARCH(string):
    string=string.strip().lower()
    lst=[]
    client = Elasticsearch(["http://localhost:9200"])
    response = client.search(
        index="cui",
        body={
            "query": {
                "bool": {
                "should": [{"match": {"label": string}},
                {"match": {"lang": "ENG"}}]

            }
            }
            ,"size":1
        }
    )
    for hit in response['hits']['hits']:
        lst.append(hit['_source']['cui'])
    return (lst)
```

Ablation Study 2.2: Use another shortest_path distance algorithm

We run the Bellman-Ford algorithm as the algorithm for the calculation of the shortest path distance on the BC5CDR dataset and got an accuracy of 59.46%.

Ablation Study 2.3: Use another entity recognition tool

In the second experiment, we wanted to find out whether our method is dependent on the entity recognition tool. So, we chose BioFalcon, as another entity recognition tool. We achieved an accuracy of 59.86%.

6.2 Accuracy and Execution_time Result of two Experiments

In the following table 6.1, we see the accuracy results for each baseline as well as for Lumos. As is shown in Table 6.1, for the MedMentions dataset, Lumos got the highest accuracy of 53.71% when we used Metamap as the entity recognition tool. We notice that Lumos is independent of the shortest_path distance algorithm, as we have achieved, for both Dijkstra and Bellman-Ford algorithms, the same accuracy result. We experimented with the performance of Lumos on the BC5CDR dataset as well. Here Lumos achieves the highest accuracy result when we used Biofalcon as the entity recognition tool and Dijkstra as the shortest_path distance algorithm. If we used the Ballman-Ford algorithm, when we have Biofalcon as our entity recognition tool, we would have achieved the same accuracy result, since in the experiments we understood that Lumos is independent of the shortest_path distance algorithm. Why Lumos is not influenced by the shortest_path algorithm? Because shortest_path algorithms track the shortest known distance between each node and the source node and update these values just when the shortest path is found, Lumos is not impacted by shortest_path algorithms. In our methodology, we have taken the MRREL table from the UMLS knowledge base and built a graph of relations based on the MRREL table, where the dataset's elements are static (at the time of our evaluation, if the dataset has updates in the future, that may be affected), i.e., the distance between two vertices is not altering. Therefore, it is not crucial whether the shortest path technique is used since the algorithm ultimately determines a static distance between the vertices in the MRREL table. After conducting two experiments and ablation studies, we understood that each of the two entity recognition tools used in Lumos has better results in different datasets. That means, MetaMap performed better, in case of higher accuracy 6.1, in the MedMentions dataset. It gave us more detected

CUIs for entities, that were correct in the MedMentions dataset. On the other hand, Biofalcon performed better in BC5CDR dataset. It gave us higher accuracy than MetaMap, i.e., Biofalcon found more correct CUIs for entities in the BC5CDR dataset 6.1. We have used MetaMap in our test dataset, the reason is explained in Section 5.5.2. In the evaluation, both entity recognition tools, Biofalcon and MetaMap are used. The reason we have considered the execution_time was that we wanted to see with which entity recognition tool and shortrdt_path algorithm, perform Lumos faster. In general, we have used accuracy and execution_time as parameters, to evaluate the efficiency and performance of our proposed approach.

Table 6.1: **The accuracy result on two Biomedical datasets (MedMentions and BC5CDR)**. Baseline values are from [64, 65], and the execution_time is calculated in Lumos. Other baselines did not consider the execution_time, and for them, we used "-", which means: there is no execution_time for the given baseline. As is shown in the table, the results for the Lumos approach are all the cases of two experiments and ablation studies.

	Accuracy	Execution time	Accuracy	Execution time
		MedMentions		BC5CDR
SciSpacy	38.8	-	53.9	-
N-GRAM TF-idf	50.9	-	86.9	-
BIO-SYN	72.5	-	87.8	-
SAPBERT	69.8	-	85.2	-
INDEPENDENT	72.8	-	90.5	-
CLUSTERING-BASED	74.1	-	91.3	-
DUAL	75.7	-	NA	-
Lumos (ours)(Metamap/Dijkstra)	53.71	14 hours and 30 minutes	59.46	3 hours and 17 minutes
Lumos(only Elasticsearch)	50.59	-	53.3	-
Lumos(Metamap/Bellman Ford)	53.71	13 hours and 35 minutes	59.46	3 hours and 21 minutes
Lumos(Biofalcon/Dijkstra)	53.53	22 hours and 17 minutes	59.86	5 hours and 11 minutes

6.3 Summary of the Chapter

In this section, we evaluated our suggested approach against one of the most known benchmarks in the biomedical domain. We went into detail about how our strategy operates. We calculated accuracy twice, once using our approach and once without, to determine whether our suggested method performs better and achieves higher accuracy or not.

Chapter 7

Conclusions and Future Work

This chapter provides a summary of the outcomes of this thesis, explains how the thesis problem was resolved, and our objectives were met, and identify major limitations in our work. We will also explore potential enhancements for future works based on the aforementioned restrictions.

7.1 Conclusion

In this thesis, we proposed Lumos, an independent and functional framework that can be used to recognize and link relations and entities in a sentence to the corresponding CUIs in the UMLS knowledge base. We evaluated the effectiveness of our approach on the two biomedical datasets. We proved through analysis that, compared to the baseline, our approach is more effective in linking mentions with ambiguous surface forms. Our method includes a heuristic approach to deal with the problem of recognizing and linking relations and entities in a sentence. Therefore, when dealing with phrases that contain multiple relations and complicated entities, our framework emphasizes the importance of heuristic approaches to handle difficulties like this in real-time.
Additionally, having an indexed-based representation of a knowledge base demonstrates how search queries can be applied to the knowledge base efficiently. Also, combining the score variable from the search engine helped us better rank and link the mentions to their corresponding CUIs in the knowledge graph. However, choosing the right variables and search engine configuration has a big impact on the quality of the results. We have achieved higher accuracy in both datasets, compared to the accuracy results of the baseline. The achievement of higher accuracy means that our proposed solution can handle ambiguous mentions efficiently and, in most cases, link

the mentions to their correct counterpart in the knowledge base.

7.2 Limitations

Like any other system or framework, our method has some limitations as well. In the following, we will talk about the limitations of our work:

- entity recognition tools Sometimes entity recognition tools are unable to recognize all the entities in the text and return an incorrect CUI based on the content. (In this case, "entity" refers to all the text's words, excluding the mention). As a result, we use the incorrect CUI to calculate the distance, which causes us to select the incorrect candidate for mention.

- One of Metamap's limitations is that it can be applied just to English contexts. Also, it is relatively slow and requires hours of computation. MetaMap's reduced accuracy in the presence of ambiguity is possibly its worst weakness. Although MetaMap uses a WSD algorithm to minimize ambiguity, it is obvious that further disambiguation work will be required to effectively fix the issue, especially given how ambiguous the Metathesaurus is getting.

- Limitation of knowledge graph: Since UMLS is a medical knowledge base, in our approach we always get lower average rates for the entities with medical backgrounds. For example: consider two words "common cold (C0009443)" and "cold temperature (C0009264)". For "common cold", there are 79 relations (Broader (RB) and Narrower (RN) Concepts), but "cold temperature" has only 11 relations (Broader(RB) and Narrower(RN) Concepts). As we traverse the graph of relations, the probability to have a lower distance for a node with many relations is extremely higher than the one with lower relations. So, for the entity ranking task, we get sometimes higher ranks for the words with a medical background, although this word might differ from the mention we are looking for. As the result, the total accuracy could deteriorate.

7.3 Future Work

Improvement of Relation and Entity Extraction: The heuristic approach can be strengthened by including more rules for relations and entity extraction. These additional rules can be introduced by looking at other biomedical datasets and attempting to identify patterns, which are rendered into rules in the approach.

Improvement of Entity Recognition: In the scope of this work, we have studied three entity recognition tools. In future work, we could mention that other entity recognition tools could also be applied, and the performance of the method could be evaluated with other tools.
Improvement of representation of the KB: For entities, there is not much room for improvement. However, to increase the possibility that any synonym for the relation would match, alternative dictionaries can be employed. Additionally, other forms of the relation's term may also be added to the KB. For instance, the word's singular or plural form, or all the tenses of the verbs in the relations.
Generalizing to other languages: The Lumos approach could be generalized to other languages as well. Other language-specific components could be added to the proposed heuristic method and could be validated with other datasets from other languages.

Bibliography

[1] URL: https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-base/.

[2] URL: https://virtuoso.openlinksw.com/.

[3] URL: https://www.nlm.nih.gov/research/umls/new_users/online_learning/OVR_001.html.

[4] URL: https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies/.

[5] URL: https://monkeylearn.com/blog/named-entity-recognition/.

[6] URL: https://www.nhs.uk/conditions/antihistamines/#:~:text=Antihistamines%5C%20are%5C%20medicines%5C%20often%5C%20used,short%5C%2Dterm%5C%20treatment%5C%20for%5C%20insomnia..

[7] URL: https://www.merkleinc.com/blog/dispelling-myths-deep-learning-vs-machine-learning.

[8] URL: https://cnvrg.io/cnn-sentence-classification/).

[9] URL: https://paperswithcode.com/method/gcn).

[10] URL: https://www.mygreatlearning.com/blog/word-embedding/).

[11] URL: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[12] URL: https://blog.christianperone.com/tag/vector-space-model/).

[13] URL: https://www.britannica.com/biography/Leonhard-Euler.

[14] URL: https://www.britannica.com/science/Konigsberg-bridge-problem.

[15] URL: https://en.wikipedia.org/wiki/Hamming_distance.

[16] URL: https://en.wikipedia.org/wiki/Levenshtein_distance.

[17] URL: https://en.wikipedia.org/wiki/Jaro%5C%E2%5C%80%5C%93Winkler_distance.

[18] URL: https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227.

[19] URL: https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227.

[20] URL: https://en.wikipedia.org/wiki/S%5C%C3%5C%B8rensen%5C%E2%5C%80%5C%93Dice_coefficient.

[21] URL: https://en.wikipedia.org/wiki/Jaccard_index.

[22] URL: http://wiki.gis.com/wiki/index.php/Shortest_path_problem.

[23] URL: https://www.researchgate.net/figure/The-pseudo-code-of-Dijkstras-algorithm-adapted-from-Wikipedia-2011_fig2_233025836.

[24] URL: https://www.educative.io/answers/what-is-the-bellman-ford-algorithm.

[25] URL: https://www.programiz.com/dsa/bellman-ford-algorithm.

[26] URL: https://medium.com/geekculture/cosine-similarity-and-cosine-distance-48eed889a5c4.

[27] URL: https://en.wikipedia.org/wiki/Search_engine_(computing).

[28] URL: https://en.wikipedia.org/wiki/Elasticsearch.

[29] URL: https://marcobonzanini.com/2015/02/02/how-to-query-elasticsearch-with-python/.

[30] URL: https://www.elastic.co/what-is/elasticsearch.

[31] URL: https://www.compose.com/articles/how-scoring-works-in-elasticsearch/.

[32] URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-fuzzy-query.html#fuzzy-query-ex-request.

[33] URL: https://hackernoon.com/how-to-use-fuzzy-query-matches-in-elasticsearch-dh1h3167.

[34] URL: https://www.elastic.co/blog/found-fuzzy-search.

[35] URL: https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/.

[36] URL: https://machinelearningknowledge.ai/tutorial-for-stopwords-in-spacy/.

[37] URL: https://en.wikipedia.org/wiki/Heuristic_(computer_science).

[38] URL: https://en.wikipedia.org/wiki/A*_search_algorithm.

[39] URL: https://en.wikipedia.org/wiki/Travelling_salesman_problem.

[40] URL: https://en.wikipedia.org/wiki/DBpedia.

[41] URL: https://www.news-medical.net/news/20230124/Smoking-during-pregnancy-reduces-childhood-brain-growth-in-very-preterm-babies.aspx.

[42] URL: https://www.wikipedia.org/.

[43] URL: https://wordnet.princeton.edu/.

[44] URL: https://www.nlm.nih.gov/research/umls/index.html.

[45] URL: https://www.ismp.org/resources/medical-abbreviations-have-contradictory-or-ambiguous-meanings.

[46] URL: https://www.studysmarter.us/explanations/english/lexis-and-semantics/polysemy/.

[47] URL: https://en.wikipedia.org/wiki/Charles_Dickens.

[48] URL: https://roadmunk.com/guides/types-of-software-development-methodologies/.

[49] URL: https://en.wikipedia.org/wiki/Spiral_model#/media/File:Spiral_model_(Boehm,_1988).svg.

[50] URL: https://numpy.org/.

[51] URL: https://pandas.pydata.org/.

[52] URL: https://neo4j.com/.

[53] URL: https://networkx.guide/algorithms/shortest-path/.

[54] URL: https://en.wikipedia.org/wiki/Neo4j.

[55] URL: https://towardsdatascience.com/medcattrainer-a-tool-for-inspecting-improving-and-customising-medcat-880a11297ebe.

[56] URL: https://www.nlm.nih.gov/research/umls/implementation_resources/metamap.html.

[57] URL: https://en.wikipedia.org/wiki/SpaCy.

[58] URL: https://en.wikipedia.org/w/index.php?title=Dijkstra%5C%27s_algorithm&oldid=1117533081.

[59] URL: https://en.wikipedia.org/wiki/A*_search_algorithm.

[60] URL: https://en.wikipedia.org/wiki/Bellman%5C%E2%5C%80%5C%93Ford_algorithm.

[61] URL: https://en.wikipedia.org/wiki/Microsoft_Azure.

[62] URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy#:~:text=Formally%5C%2C%5C%20accuracy%5C%20has%5C%20the%5C%20following,N%5C%20%5C%2B%5C%20F%5C%20P%5C%20%5C%2B%5C%20F%5C%20N.

[63] URL: https://pubmed.ncbi.nlm.nih.gov/.

[64] Dhruv Agarwal et al. "Entity Linking via Explicit Mention-Mention Coreference Modeling". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Seattle, United States: Association for Computational Linguistics, July 2022, pp. 4644–4658. DOI: 10.18653/v1/2022.naacl-main.343. URL: https://aclanthology.org/2022.naacl-main.343.

[65] Rico Angell et al. "Clustering-based Inference for Biomedical Entity Linking". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 2598–2608. DOI: 10.18653/v1/2021.naacl-main.205. URL: https://aclanthology.org/2021.naacl-main.205.

[66] S. Auer et al. "DBpedia: A Nucleus for a Web of Open Data". In: *ISWC/ASWC*. 2007.

[67] Trapit Bansal et al. "Simultaneously Linking Entities and Extracting Relations from Biomedical Text Without Mention-level Supervision". In: *AAAI*. 2020.

[68] Dennis Becker et al. "Predictive modeling in e-mental health: A common language framework". In: *Internet Interventions* (). DOI: https://doi.org/10.1016/j.invent.2018.03.002. URL: https://www.sciencedirect.com/science/article/pii/S2214782917301124.

[69] Olivier Bodenreider. "The Unified Medical Language System (UMLS): Integrating Biomedical Terminology". In: *Nucleic acids research* 32 (Feb. 2004), pp. D267–70. DOI: 10.1093/nar/gkh061.

[70] Barry Boehm and Wilfred Hansen. "The Spiral Model as a Tool for Evolutionary Acquisition". In: *CrossTalk* 14 (Jan. 2001).

[71] Busra Celikkaya et al. "LATTE: Latent type modeling for biomedical entity linking". In: *AAAI 2020*. 2020. URL: https://www.amazon.science/publications/latte-latent-type-modeling-for-biomedical-entity-linking.

[72] Lihan Chen et al. "Short Text Entity Linking with Fine-grained Topics". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (2018).

[73] Lihu Chen, Gaël Varoquaux, and Fabian M. Suchanek. "A Lightweight Neural Model for Biomedical Entity Linking". In: *AAAI*. 2021.

[74] Xiao Cheng and Dan Roth. "Relational Inference for Wikification". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1787–1796. URL: https://aclanthology.org/D13-1184.

[75] Silviu Cucerzan. "Large-Scale Named Entity Disambiguation Based on Wikipedia Data". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 708–716. URL: https://aclanthology.org/D07-1074.

[76] Marko Čuljak et al. "Strong Heuristics for Named Entity Linking". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop*. July 2022, pp. 235–246. DOI: 10.18653/v1/2022.naacl-srw.30.

[77] Jennifer D'Souza and Vincent Ng. "Sieve-Based Entity Linking for the Biomedical Domain". In: *ACL*. 2015.

[78] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. URL: http://arxiv.org/abs/1810.04805.

[79] Orri Erling and Ivan Mikhailov. "RDF Support in the Virtuoso DBMS". In: *Networked Knowledge - Networked Media: Integrating Knowledge Management, New Media Technologies and Semantic Systems*. Ed. by Tassilo Pellegrini et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 7–24. ISBN: 978-3-642-02184-8. DOI: 10.1007/978-3-642-02184-8_2. URL: https://doi.org/10.1007/978-3-642-02184-8_2.

[80] U. Feige, David Peleg, and Guy Kortsarz. "The Dense k -Subgraph Problem". In: *Algorithmica* 29 (Mar. 2001), pp. 410–421. DOI: 10.1007/s004530010050.

[81] Paolo Ferragina and Ugo Scaiella. "TAGME: on-the-fly annotation of short text fragments (by wikipedia entities)". In: *Proceedings of the 19th ACM international conference on Information and knowledge management* (2010).

[82] Yingjie Gu et al. "Read, Retrospect, Select: An MRC Framework to Short Text Entity Linking". In: *ArXiv* abs/2101.02394 (2021).

[83] Yuhang Guo et al. "A Graph-based Method for Entity Linking". In: *Proceedings of 5th International Joint Conference on Natural Language Processing*. Asian Federation of Natural Language Processing. URL: https://aclanthology.org/I11-1113.

[84] Zhaochen Guo and Denilson Barbosa. "Entity Linking with a Unified Semantic Representation". In: *Proceedings of the 23rd International Conference on World Wide Web*. New York, NY, USA: Association for Computing Machinery, 2014. ISBN: 9781450327459. DOI: 10.1145/2567948.2579705. URL: https://doi.org/10.1145/2567948.2579705.

[85] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: 2017. ISBN: 9781510860964.

[86] Charles R. Harris et al. "Array Programming with NumPy". In: (2020). URL: https://arxiv.org/abs/2006.10256.

[87] Johannes Hoffart et al. "Robust Disambiguation of Named Entities in Text". In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, July 2011, pp. 782–792. URL: https://aclanthology.org/D11-1072.

[88] Raphael Hoffmann et al. "Knowledge-Based Weak Supervision for Information Extraction of Overlapping Relations". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 541–550. URL: https://aclanthology.org/P11-1055.

[89] Aidan Hogan et al. "Knowledge Graphs". In: *ACM Comput. Surv.* 54.4 (July 2021). ISSN: 0360-0300. DOI: 10.1145/3447772. URL: https://doi.org/10.1145/3447772.

[90] Filip Ilievski et al. "Context-enhanced Adaptive Entity Linking". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia: European Language Resources Association (ELRA), May 2016, pp. 541–548. URL: https://aclanthology.org/L16-1086.

[91] Vivek Iyer, Arvind Agarwal, and Harshit Kumar. "VeeAlign: a supervised deep learning approach to ontology alignment". In: *OM@ISWC*. 2020.

[92] Manoj Prabhakar Kannan Ravi et al. "CHOLAN: A Modular Approach for Neural Entity Linking on Wikipedia and Wikidata". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. Online: Association for Computational Linguistics, Apr. 2021, pp. 504–514. DOI: 10.18653/v1/2021.eacl-main.40. URL: https://aclanthology.org/2021.eacl-main.40.

[93] Samir Khuller and Barna Saha. "On finding dense subgraphs". English (US). In: *Automata, Languages and Programming - 36th International Colloquium, ICALP 2009, Proceedings*. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) PART 1. Funding Information: Research supported by NSF CCF 0728839 and a Google Research Award.; 36th International Colloquium on Automata, Languages and Programming, ICALP 2009 ; Conference date: 05-07-2009 Through 12-07-2009. 2009, pp. 597–608. ISBN: 3642029264. DOI: 10.1007/978-3-642-02927-1_50.

[94] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In.

[95] Zeljko Kraljevic et al. *Multi-domain Clinical Natural Language Processing with MedCAT: the Medical Concept Annotation Toolkit*. 2020. DOI: 10.48550/ARXIV.2010.01165. URL: https://arxiv.org/abs/2010.01165.

[96] Sayali Kulkarni et al. "Collective Annotation of Wikipedia Entities in Web Text". In: ISBN: 9781605584959. DOI: 10.1145/1557019.1557073. URL: https://doi.org/10.1145/1557019.1557073.

[97] Haodi Li et al. "CNN-based ranking for biomedical entity normalization". In: *BMC Bioinformatics* 18 (2017).

[98] Jiao Li et al. "BioCreative V CDR task corpus: a resource for chemical disease relation extraction". In: *Database* 2016 (May 2016). baw068. ISSN: 1758-0463. DOI: 10.1093/database/baw068. eprint: https://academic.oup.com/database/article-pdf/doi/10.1093/database/baw068/8224483/baw068.pdf. URL: https://doi.org/10.1093/database/baw068.

[99] Yujia Li et al. *GATED GRAPH SEQUENCE NEURAL NETWORKS*. 2016.

[100] Fangyu Liu et al. "Self-alignment Pre-training for Biomedical Entity Representations". In: *CoRR* abs/2010.11784 (2020). arXiv: 2010.11784. URL: https://arxiv.org/abs/2010.11784.

[101] Xiaohua Liu et al. "Entity Linking for Tweets". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. URL: https://aclanthology.org/P13-1128.

[102] Lajanugen Logeswaran et al. "Zero-Shot Entity Linking by Reading Entity Descriptions". In: *ACL*. 2019.

[103] Lajanugen Logeswaran et al. "Zero-Shot Entity Linking by Reading Entity Descriptions". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3449–3460. DOI: 10.18653/v1/P19-1335. URL: https://aclanthology.org/P19-1335.

[104] Gang Luo et al. "Joint Entity Recognition and Disambiguation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 879–888. DOI: 10.18653/v1/D15-1104. URL: https://aclanthology.org/D15-1104.

[105] Rada Mihalcea and Andras Csomai. "Wikify! Linking Documents to Encyclopedic Knowledge". In: *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*. 2007. ISBN: 9781595938039. DOI: 10.1145/1321440.1321475. URL: https://doi.org/10.1145/1321440.1321475.

[106] David Milne and Ian H. Witten. "Learning to Link with Wikipedia". In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. New York, NY, USA, 2008. ISBN: 9781595939913. DOI: 10.1145/1458082.1458150. URL: https://doi.org/10.1145/1458082.1458150.

[107] David N. Milne and Ian H. Witten. "An effective, low-cost measure of semantic relatedness obtained from Wikipedia links". In: 2008.

[108] Sunil Mohan and Donghui Li. "MedMentions: A Large Biomedical Corpus Annotated with UMLS Concepts". In: *CoRR* abs/1902.09476 (2019). arXiv: 1902.09476. URL: http://arxiv.org/abs/1902.09476.

[109] Andrea Moro, Alessandro Raganato, and Roberto Navigli. "Entity Linking meets Word Sense Disambiguation: a Unified Approach". In: (), pp. 231–244. DOI: 10.1162/tacl_a_00179. URL: https://aclanthology.org/Q14-1019.

[110] Hamada A. Nayel et al. "Improving Multi-Word Entity Recognition for Biomedical Texts". In: *CoRR* abs/1908.05691 (2019). arXiv: 1908.05691. URL: http://arxiv.org/abs/1908.05691.

[111] Mark Neumann et al. "ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing". In: *Proceedings of the 18th BioNLP Workshop and Shared Task*. Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 319–327. DOI: 10.18653/v1/W19-5034. URL: https://aclanthology.org/W19-5034.

[112] Natasha Noy et al. "Industry-Scale Knowledge Graphs: Lessons and Challenges: Five Diverse Technology Companies Show How It's Done". In: *Queue* 17.2 (Apr. 2019), pp. 48–75. ISSN: 1542-7730. DOI: 10.1145/3329781.3332266. URL: https://doi.org/10.1145/3329781.3332266.

[113] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[114] Lev Ratinov et al. "Local and Global Algorithms for Disambiguation to Wikipedia". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 1375–1384. URL: https://aclanthology.org/P11-1138.

[115] Tim Rocktäschel, Michael Weidlich, and Ulf Leser. "ChemSpot: a hybrid system for chemical named entity recognition". In: (). ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bts183. eprint: https://academic.oup.com/bioinformatics/article-pdf/28/12/1633/16904367/bts183.pdf. URL: https://doi.org/10.1093/bioinformatics/bts183.

[116] Henry Rosales-Méndez, Aidan Hogan, and Barbara Poblete. "Fine-Grained Entity Linking". In: *Journal of Web Semantics* (2020). ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2020.100600. URL: https://www.sciencedirect.com/science/article/pii/S1570826820300378.

[117] Ahmad Sakor, Kuldeep Singh, and Maria-Esther Vidal. "FALCON: An Entity and Relation Linking Framework over DBpedia". In: *International Workshop on the Semantic Web*. 2019.

[118] Ahmad Sakor et al. "Old is Gold: Linguistic Driven Approach for Entity and Relation Linking of Short Text". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. DOI: 10.18653/v1/N19-1243. URL: https://aclanthology.org/N19-1243.

[119] Michael Schlichtkrull et al. "Modeling relational data with graph convolutional networks". In.

[120] Mujeen Sung et al. "Biomedical Entity Representations with Synonym Marginalization". In: *CoRR* abs/2005.00239 (2020). arXiv: 2005.00239. URL: https://arxiv.org/abs/2005.00239.

[121] Petar Veličković et al. DOI: 10.48550/ARXIV.1710.10903. URL: https://arxiv.org/abs/1710.10903.

[122] Denny Vrandečić and Markus Krötzsch. "Wikidata: A Free Collaborative Knowledgebase". In: *Commun. ACM* 57.10 (Sept. 2014), pp. 78–85. ISSN: 0001-0782. DOI: 10.1145/2629489. URL: https://doi.org/10.1145/2629489.

[123] Gongqing Wu, Ying He, and Xuegang Hu. "Entity Linking: An Issue to Extract Corresponding Entity With Knowledge Base". In: *IEEE Access* 6 (2018), pp. 6220–6231. DOI: 10.1109/ACCESS.2017.2787787.

[124] Ledell Yu Wu et al. "Scalable Zero-shot Entity Linking with Dense Entity Retrieval". In: *EMNLP*. 2020.

[125] Kun Xu et al. "Graph2Seq: Graph to Sequence Learning with Attention-based Neural Networks". In: ().

[126] Wen-tau Yih et al. "Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 1321–1331. DOI: 10.3115/v1/P15-1128. URL: https://aclanthology.org/P15-1128.

[127] Sheng Zhang et al. "Knowledge-Rich Self-Supervised Entity Linking". In: *ArXiv* abs/2112.07887 (2021).

[128] Chuxu Zhang* et al. "Heterogeneous Graph Neural Network". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.