



Universiteit
Leiden

Master Computer Science

Predicting care plan goals using a knowledge
graph-based recommender system

Name: Diego Barreiro Clemente
Student ID: s3254933
Date: 19/07/2023
Specialisation: Data Science
1st supervisor: Prof. Dr. Marco Spruit
2nd supervisor: Dr. Suzan Verberne

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Natural Language Processing has been rapidly maturing over the past recent years, enabling models to better understand textual data. However there is still a lot of potential to be explored and leveraged.

In this thesis we try to close the bridge between English related models and work and non-English models by providing detailed steps of a pipeline to extract valuable information from a Dutch written healthcare data reports. Since the vast majority of companies and researchers work with English data, the models available for other languages like Dutch in this case are very limited, specially for downstream tasks.

The main goal of the thesis is to build a recommendation system that is capable of providing meaningful insights about a given patient. The ideal scenario would be to provide the target patient's ID and obtain based on similar patients, a list of keywords about the patient's specific care needs.

We lay out each step involved in a comprehensive manner, motivating the choice of resources along the way. By evaluating the approaches and experiments providing a descriptive study on its performance.

It can be seen from the results obtained that there is a lot of potential in finetuning a model for a downstream task, where the structure of the data is unique. While the performance can still be improved, we showcased that with limited labeled data a model capable of achieving competitive results is possible.

Other approaches revolving around the use of word embeddings for classification purposes have been discussed, and while not used in the final implementation of the pipeline the potential for other tasks outside the scope of this project is shown.

Our contribution consists on an end-to-end pipeline that is able to reflect the advantages of leveraging NLP techniques to process large healthcare records to extract meaningful information from unstructured data.

Keywords: Named Entity Recognition (NER), Knowledge Graph (KG), recommendation system, Dutch health records, information extraction.

Contents

1	Introduction	1
2	Background and Literature Review	3
2.1	Models	3
2.1.1	Neural Networks	3
2.1.2	Transformer models	4
2.2	NLP in Healthcare	4
2.2.1	NER	5
2.2.2	Part-of-Speech	6
2.3	NLP in the Dutch language	6
2.4	Pretraining and Fine-tuning	6
2.5	Knowledge Graphs	7
2.5.1	Recommendation system	7
3	Methodology	9
3.1	Dataset and Preprocessing steps	10
3.1.1	Exploratory analysis	10
3.1.2	Training data	11
3.2	Named Entity Recognition (NER)	12
3.2.1	Tools and resources	12
3.3	Relation Extraction	13
3.4	Recommendation system	14
3.4.1	Knowledge graph	14
3.4.2	Node embeddings	14
4	Experimental Setup and Results	15
4.1	Data processing	15
4.2	Named Entity Recognition	20

4.2.1	Word Embeddings	21
4.2.2	Fine-tuning transformer model	27
4.3	Triplet extraction	31
4.3.1	Optimization	31
4.3.2	Extracting Named Entities from the large collection	32
4.3.3	Relation Extraction	33
4.4	Recommendation system	35
4.4.1	Knowledge Graph	35
4.4.2	Similar patients	38
4.4.3	Similar careplans	41
5	Discussion	42
6	Conclusion	44
A	Appendix	49

List of Figures

3.1	Diagram of the pipeline.	9
4.1	Model output for an example sentences.	20
4.2	Visual representation of the word embeddings.	24
4.3	Zoomed-in visual representation of the word embeddings.	24
4.4	Cumulative explained variance plot.	25
4.5	Example sentence in Label Studio.	27
4.6	Example sentences using data augmentation.	28
4.7	Performance comparison using varying cores.	33
4.8	Entity reconstruction process.	34
4.9	Small sample of the Neo4j graph visualization.	36
4.10	Example of the final Recommendation output using embeddings.	39
4.11	Example of the final Recommendation output using exact matches.	39
4.12	Visualization plots with most repeated neighbours across similar patients (same target patient).	40
4.13	Example of the final careplan recommendation output.	41

List of Tables

4.1	Example of how data is presented in the original csv file	16
4.2	Resulting records from approach 1.	17
4.3	Resulting records from approach 2.	18
4.4	Resulting records from approach 3.	19
4.5	Top results based on Word2Vec embeddings (Examples for Body parts and Medications).	22
4.6	Top results based on Word2Vec embeddings (Examples for Diseases and Locations, healthcare facilities).	22
4.7	Number of components per common cumulative variance values.	23
4.8	Similarity metric between three samples words and the category words.	26
4.9	Precision and Recall scores for each fold evaluated on the test set.	31
4.10	F1 scores for each fold evaluated on the test set.	31
4.11	Mean and Standard Deviation for the performance metrics: Precision, Recall and F1 scores.	31
4.12	Predefined relation labels.	35

Chapter 1

Introduction

The ever increasing amount of data in today's world leads to the improvement and refinement of data processing techniques, to make sense and extract value from large volumes of information.

In the healthcare domain specifically there is a lot of information regarding a patient's health condition. This just makes it a very painstaking and laborious task to manually evaluate a patient based on all of its data and provide the optimal careplan that fits the patients needs. While doing this exhaustive study manually may yield a good result it is not suitable and sustainable for large number of patients, that is where Natural Language Processing (NLP) techniques play an important role.

The field of Artificial Intelligence (AI) has evolved rapidly in recent years, and with it the NLP subfield. In this Thesis the advancements made in NLP are leveraged to extract meaningful information from patients health records, to help save time and improve the patients care outcome.

The high volume of healthcare records poses a great challenge in extracting relevant information and organizing it, information such as diseases, medications, locations and the relations between them specially given the lack of a predefined structure within these records. To address this issue we aim to develop a pipeline that integrates a Knowledge-graph based recommender system using State-of-the-art techniques such as Named Entity Recognition (NER) and Relation Extraction (RE) combined to simplify a long string of text into a few key words that capture the gist of it. The system will be trained to automatically recognise and extract keywords such as different types of diseases, medical conditions, medications and locations and provide patients with similar data. The aim is to provide the careplan needs of any given patient based on this similar patients' medical data, because patients with similar medical conditions will require in most cases the same type of treatment, saving medical professional the time to go through the entire medical history of the patient.

These insights are extracted from a large dataset of Dutch healthcare records provided by a private healthcare facilitator based in Delft, records that contain detailed information about the patient's medical history such as a summary of each medical appointment, GP (general practitioner) annotations after each visit and caregiver remarks about the current state of the patient.

The aim of the project is to facilitate the work of professionals whose job is to choose the opti-

mal care plan for a given patient (mostly elderly) according to their short-term and long-term medical history, by comparing similar patient's conditions and previously assigned careplans. These professionals would have to go through hundreds of records about a patient to get an idea about its current situation, because some of them date back to more than 10 years ago and may still be relevant to the case. This project tries to emphasize the benefits of leveraging NLP techniques to extract the key information from text data, with the goal of establishing the proposed solution as the primary source of information in future patient studies.

In this thesis project, we propose and develop a pipeline to address the following research questions:

1. "How can a knowledge graph-based recommender system provide context for predicting the care plan goals?"
2. "How does the performance of a rule based approach compare to that of a transformer based approach for Named Entity Recognition in Dutch clinical notes."

The entirety of the project is developed for Prime Vision, a private company located in Delft, with direct supervision from the NLP department within the company. This project serves as a potential feature to be added to a larger project, carried out at the company for the past few years which tries to implement NLP features into a real-world application in the medical domain. Additionally the trained NER model and preprocessing scripts used in the project extend its benefits to another ongoing NLP project at the company.

Chapter 2

Background and Literature Review

For this Thesis project the use of NLP techniques is extremely helpful as it is going to help extract value from the health records, by understanding the contexts in which the relevant Named Entities appear.

This chapter presents an overview of the the key topics, concepts and technical implementations crucial for the development of the project.

2.1 Models

In this section, we discuss the influential role of Neural Networks and Transformer-based models that have served as inspiration and references for the project. While they are not all directly utilized, these models have significantly contributed to shaping the methodologies and approaches of the project.

2.1.1 Neural Networks

Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) had been the main option to work with when treating text before the introduction of transformers, as they are efficient at dealing with sequential data, as seen in Sutskever, Vinyals, and Le (2014) with the introduction of the seq2seq model. The authors present a model based on RNNs that shows how prominent sequential learning can be when treating sentences of variable length. RNNs and CNNs networks can be adapted to estimate the output of a given input by taking into consideration the previous stack of inputs from the same sentence making them suitable for text classification tasks. CNNs, originally used for image processing can also be adapted to NLP, Kim (2014), with very little effort, attaining state-of-the-art performances in several main tasks.

While RNNs do capture the context of a word, they lack in performance when dealing with long sentences, as they do not capture well relationships between words that appear far away within the same sentence.

To tackle this limitation, the attention mechanism, Vaswani et al. (2017), was later introduced that would enable the creation of an improved neural network architecture, the transformer

model.

2.1.2 Transformer models

A lot of research has been done recently that enhanced the field of NLP, one major breakthrough being the development of Large Language Models (LLMs) which was made possible by the creation of transformer-based models like BERT or GPT-2/GPT-3 thanks to the introduction of the attention mechanism, this posed an inflexion point for all NLP related studies. This new attention mechanism allowed the models to account for the full spectrum of words in a given sentence, thus taking into account the entire context of a given sentence.

BERT

In 2018 Google presented BERT (Bidirectional Encoder Representations from Transformers), Devlin et al. (2018), model which achieved state-of-art performance across many tasks in NLP, and the fine-tuned versions of it still do on some downstream domain specific tasks.

The BERT model uses two unsupervised tasks to pretrain a model:

1. MLM, short for Masked Language Model, in this task the model would need to predict random masked words. This step ensures the model learns contextual representations of words.
2. NSP, short for Next Sentence Prediction, it is the task of predicting if a given sentence is the succeeding sentence or not for another selected sentence, it shows good performance in learning sentence relationships and proven to be very beneficial for the model in understanding the structure of a text (group of several related sentences).

RoBERTa

The RoBERTa model, Liu et al. (2019), is based on the same principles and architecture of the BERT model (Robustly optimized BERT approach), but it was created to bypass some of the limitations of this model and to improve its performance, as it was trained on a much larger corpus for a longer period of time.

As seen before the BERT model uses two main tasks, but in this case the RoBERTa model removes one of them, the NSP task, as it was proven not to yield a direct enhancement in performance. The RoBERTa architecture instead focuses on the MLM task, to predict the context of words based on their context. It uses a different strategy than that posed in the BERT paper, a dynamic approach as opposed to a fixed approach, which has seen an improvement in the generalization of word representations.

2.2 NLP in Healthcare

The healthcare domain has also seen plenty of opportunity for growth making use of these transformer models. In the past years researchers in the medical and health domains have used

pretrained transformer models to extract valuable information from patients healthcare records. The fine-tuned version of some of these models have been proven to achieve state-of-the-art performance across multiple downstream domain specific tasks of NLP, Akbik et al. (2019).

Health records usually contain important information about a patients current and past conditions, including very detailed information about the mental and physical well-being of the person at each time. This information is shown over a sequence of long text sentences ordered chronologically by patient and visit, this adds up to a lot of text for a medical professional to go through, this is where the advances in NLP can help in extracting important keywords such as names of diseases, medical conditions, locations and medications, that can be later used to create a unique profile for every patient. This can save a lot of time in the decision making of the careplan facilitator when selecting the optimal plan.

The authors of the BioBERT, Lee et al. (2019), clearly provide proof that a finetuned BERT model on domain specific corpora and with very little modifications to the structure of the model will achieve state-of-the-art performances across many NLP tasks. In the paper the authors mention a large limitation that arises when comparing the performance of a general model against a domain specific model, which is the difference in word distributions between a general domain corpora and specific domain corpora, the introduction of their BioBERT model tries to overcome this limitation.

2.2.1 NER

Named Entity Recognition is one of the main sequence labelling tasks in NLP, it is very domain dependent, as the named keywords vary according to the needs of the application. It is a task that has undergone extensive research and many methods have been explored so far, so there is no perfect approach. In Gorinski et al. (2019) the authors experiment using various methods for domain specific NER tasks, and comment on the trade-off of using a rule-based approach in comparison to using machine learning methods.

In the case of the Health domain, and more specifically in the healthcare subdomain, there are several keywords to be extracted. However, the most relevant and subjective to an individual's situation are the diseases, medications, body parts affected and locations where the individual received treatment or care of any kind.

The way the named entities are extracted is by looking at patterns in which previously seen entities have appeared, considering similar contexts as similar words tend to appear in similar syntactical structures and among analogous words. When training a model on a downstream NER task it is important to keep a wide variety in sentences, using different vocabulary and structures so that it is able to capture as many contexts as possible for a given type of entity. The authors of Kocaman and Talby (2020) highlight the need to extract valuable insights from unstructured corpora, which is often seen populating the majority of data in the healthcare domain, and present a biomedical NER model capable of outperforming previous state-of-the-art models in several benchmarks.

Additionally, the authors in Nath, Lee, and Lee (2022) have added the concept of polarity into the NER task, providing more context to entity extraction by considering negation and other attributes related to the entity.

2.2.2 Part-of-Speech

The main reason why Part-Of-Speech (POS) tagging also plays an important role on identifying these keywords is because it is the process of analyzing a sentence from the grammatical point of view. Words with similar roles within a sentence are likely to carry out the same purpose, it is also extremely useful to find words that modify an entity, it is not the same "Dementia" as "Possible dementia" or "Not dementia", precisely for that we need POS tagging in conjunction with NER to achieve better results.

In Nguyen and Verspoor (2018) the authors evaluate the performance of different POS tagging and dependency parsing models specifically on biomedical texts, as well as proposing approaches to enhance the performance of such models.

Additional work was done by the authors of Zhang et al. (2021) where they propose a method using a dependency parser model to extract relations between different groups of words instead of solely words within the same sentence.

2.3 NLP in the Dutch language

While English has a very large availability of training data and language models, that is not so much the case for the Dutch language. This is slowly changing as recent years have seen a lot of interest and improvement in building good models to work with Dutch data as seen in Vries et al. (2019) (BERTje) and Delobelle, Winters, and Berendt (2020) (RobBERT).

In more recent times new research done, Delobelle, Winters, and Berendt (2022) (RobBERT2022), takes into account the evolution of the Dutch language in recent years to enhance the model's ability to capture modern unseen words.

2.4 Pretraining and Fine-tuning

Once we have a problem at hand that we need to address, we have to consider a two important things before we can start training a model on a task. These two factors will contribute to our decision on whether we should choose a pretrained model and fine-tune it on a downstream task or if we should train a model from scratch with our data.

1. Accessibility of existing models for the particular task at hand.
2. Availability of training data.

There is evidence that either approach can be beneficial attending to our unique circumstances; in Zhou et al. (2022) a good performance was achieved by fine-tuning a pretrained model on a small set of records data, in Verkijk and Vossen (2021) it was shown that training a model from scratch yielded better results than fine tuning an existing model.

The reason why fine-tuning is usually the preferred choice is because training a model from scratch is computationally expensive, time-consuming and takes a large corpus of training data to carry out, so for most researchers that is not an option, and since the availability of domain specific models has risen significantly in the past years this is less of a problem. However,

fine-tuning a model still has some limitations, if the corpus used in the pretrained model is different from that of the target domain the model may not be as effective, as it may not have enough knowledge of the target domain. In cases where the pretrained domain and the target domain are not too dissimilar it is shown to be beneficial to use a pretrained model that has already learned various language representations across multiple samples, that can help in the fine-tuning phase, Gururangan et al. (2020).

2.5 Knowledge Graphs

Knowledge Graphs are becoming more and more popular as a means of data storage and representation because of their ability to capture node attributes and the relations of each attribute to the node they are linked to, making it a very appealing approach for certain NLP tasks, as the extensive studies made by the authors in Schneider et al. (2022) and Turki et al. (2023) indicate. The data is usually in the form of triplets: an entity that acts as a subject, the object and a relation that links both together, giving a deeper understanding and context about the data.

In contrast to conventional databases that make use of joins to gather data from several different tables, in knowledge graphs we make use of graph traversal algorithms that have been optimized to query data faster in this new format. Knowledge Graphs are also convenient to incorporate new data samples and very intuitive when it comes to showing a small amount of nodes for exploratory analysis.

Many recent papers have leveraged the use of Knowledge Graphs in NLP tasks. In Barroca et al. (2022) the authors go over the steps of extracting information from text to enrich a Knowledge Graph and how to overcome some of the challenges of it. Sarhan and Spruit (2021) introduced a new model for NER in a relatively unexplored domain (Cybersecurity) and built on top of a Knowledge Graph using KG canonicalization to enhance the graph's accuracy.

2.5.1 Recommendation system

Additionally, a Knowledge Graph can be used for more than storing and querying information, but it can also be employed as a recommendation system. By exploiting the embeddings of the nodes in the graph we can extract similar nodes using similarity metrics to compare such embeddings. This can be extremely useful in a variety of applications, specially in the healthcare industry, where it can save time and provide better results than manually checking the data, resulting in a much better experience for the patient and care provider.

Graph embedding algorithms saw an Inflection point when the authors of Perozzi, Al-Rfou, and Skiena (2014) first presented DeepWalks. The idea behind it was to use random walks to learn low dimensional vector representations of nodes by treating the random paths as sentences. In Grover and Leskovec (2016) the authors introduce `node2vec`, an algorithm for learning features in a network. Making use of random walks the algorithm returns the feature representation of nodes based on their neighbors. Node2vec tries to overcome some of the limitations that the earlier DeepWalk algorithm had, by adding a weighted component, which was missing, so that frequency and importance of a node can also impact the result.

In Hamilton, Ying, and Leskovec (2017) authors introduce GraphSAGE, another algorithm

for obtaining node embeddings from a graph. GraphSAGE works for unseen data, avoiding the overhead of training individual embeddings for new samples. Leveraged by the authors of Szubert and Steedman ([2019](#)) where they present a way of knowledge fusion to merge identical nodes together and avoid having multiple nodes referring to the same object.

Chapter 3

Methodology

In this chapter the methods and experiments carried out will be explained in a detailed manner. The selection of methods and techniques is motivated below alongside the issues faced and how to overcome them.

An overview of the entire pipeline process is shown below.

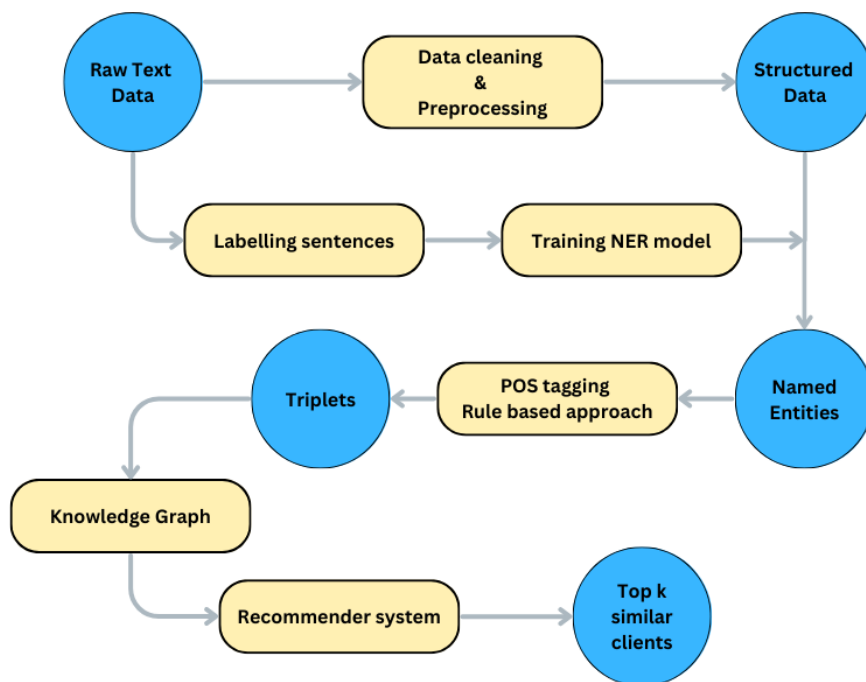


Figure 3.1: Diagram of the pipeline.

In the diagram each process is encapsulated in a yellow background element. Every step of the process leads to the transformation of the input into a more compact form that captures the information we truly need. The information regarding the methods is laid out in a detailed manner below, focusing on the set up, while the experimentation has its own chapter following this one.

3.1 Dataset and Preprocessing steps

In this first section we introduce the data that has been used in our study, its structure and the techniques used to clean and preprocess to later feed into a model. Due to the sensitive origin of the data it is not possible to share samples from the actual data, as it contains personal information from patients that could lead back to them (Names of patients, diseases, locations...), but a precise description of it will give a very accurate idea about how the data looks.

The adequate measures have been taken to ensure the data stays safe, by encrypting the folder containing all the work using VeraCrypt and encrypting the folder in case of extraction using BitLocker. Only my supervisors and myself had direct access to the data.

The data comes from a private healthcare facilitator based in Delft (they have requested to remain anonymous) with facilities in many other cities in The Netherlands. The dataset contains health records of patients that have been or still are currently under the company's supervision, these records contain in their majority notes from the health professionals after each check up visit with the patient. The health professional summarizes the information they extract from the patient regarding their physical and mental well-being and add personal comments for the care professional responsible for the patient on a day-to-day basis to see, these comments or notes may include the medication the patient should be taking, special treatments for any known diseases or conditions, dietary needs and many other relevant remarks.

3.1.1 Exploratory analysis

The data is contained within a excel file in csv format. The file contains information which has no consistent structure across rows of data. To help us navigate the raw text data we are provided a header row with the names of the columns of data, totaling 32, we pay special attention to the 'clientId', 'employeeObjectId' and 'comment' columns, this last column containing all the information (annotations) regarding the medical visit of the health professional with the patient, the rest of fields include integer values that add no value to the project and are in most cases empty. Although each healthcare record supposedly has 32 values, this is not always the case, there are inconsistencies in the length of each record making it very hard to extract value at first.

In the vast majority of cases the rows of data do not have 32 comma separated values, but instead have much less, so the first assumption made was that the remaining data values were missing from the health record, and that each new record was delimited by a newline. Following this assumption, each record would have different lengths, making it harder to know in which column the actual annotations from the visit were, but seeing as the rest of the entries were numerical, looking at the only entry that had string sentences inside yielded the expected result. One major drawback of this interpretation was that most records would not have an 'objectId' or 'clientId' entry to identify the record, which would make things harder down the line.

After extensive examination of the dataset, a conclusion was drawn about the actual structure of the data. We concluded that each record had indeed 32 items, and rows with less items meant they were a fraction of a record that continues in another row. So the start of a record is delimited by the 'objectId' numerical entry value which is the first column of data, until

another 'objectId' is found.

Each row of data in csv file is a fraction of healthcare record of a patient. In some rare cases the record fits in just one row, but as explained before, every record has a fixed length of 32 elements so we merge rows until forming lists of 32 items. Each list of 32 items corresponds to a single visit of patient to the doctor, the same patient come have multiple records within the dataset, one for every visit or medical appointment. Table 4.1 shows a couple of rows of data to showcase the actual structure of it.

Following this last implementation, from the csv file we obtain a total of 12865246 records of data and 1930 records that were not able to be loaded because they had a different formatting than the rest. Overall we managed to successfully retrieve 99.985% of records.

Word embeddings

Initially after having processed the healthcare data, we considered the use of word embeddings to cluster words into different entity categories based on their similarity. Training a word2vec model on the entire dataset surprisingly works better than anticipated, as can be seen from the results in the following chapter. Having obtained the embeddings we apply dimensionality reduction techniques, such as PCA or t-SNE, Lebet and Lebet (2013), to filter out the dimensions based on cumulative variance and preserve the most informative components from the dense vector embeddings. By utilizing the two or three most significant components, we can effectively visualize the embeddings in a 2D or 3D plot to examine them for any discernible clusters.

3.1.2 Training data

To obtain a model capable of extracting relations, keywords and patterns from the health text data it is strongly recommended the use of labeled data for the model to learn. These keywords (Named Entities) are different to every domain, in our case what makes a patient unique are the medical conditions relevant to their well-being as well as the body parts affected by such condition, the medication they take and dosage and the health centers where they reside or were treated. After laying out the main personal identifiers of a patients health history we extract four main entity categories, which are: Diseases, Body, Medication and Locations.

We obtained labeled data from three different sources:

1. Samples obtained at random from the entire set of healthcare records until reaching a total amount of 1000 sentences. An additional 1000 sentences were obtained, from which every 250 subgroup of sentences had at least one main entity in them to balance out the set of sentences. The end result would still have overlapping tags meaning sentences used for one tag could also have words pertaining to other tag classes but the overall set was class balanced. The way we sampled these sentences was using a set of keywords scraped from sites that had relevant information related to each of the four main entity classes. In python there are two main libraries available for web scraping, BeautifulSoup and Selenium, the latter used in cases where a dynamic approach is more suitable. We extracted as many terms as we found available, ranging from a couple hundred to a few

thousand in some entity classes. These keywords were then used to filter out sentences from the dataset.

To manually label these selected sentences we used Label Studio, a multipurpose labelling software that is very convenient for NER task as it has a built-in template for this, where you import your data in csv format and you can label in each sentence the words with their corresponding entity label, save the results and export them in many formats, including .conll, a format comprised of the tokenised sentences with the corresponding IOB tags for each word, this format is seen as the standard in NER tasks.

2. Using Data Augmentation techniques, a similar approach to Synonym Replacement (SR) where we swap an entity with another entity of its kind, and making use of the scraped terms mentioned before, we enlarge the second set of 1000 sentences by a x8 ratio. To stick to a class balanced labeled set we set a maximum of 2000 extra sentences per main entity tag. The way we generate new sentences is by selecting one sentence from each subset of 250 sentences that belongs to one of the four main entities, and replace the entity or entities present in it that match the current main entity with another word within the scraped data. By generating sentences that exhibit a similar structure but vary in terms of keywords, we acquire diverse contextual knowledge for new unseen entities. This approach facilitates the ability of our model to learn and comprehend language in a more robust manner.
3. With the help of the openAi API and using one of the models they have at their disposal we generated an extra 1500 sentences from scratch that are completely different from the ones already obtained before.

We used the model 'gpt-3.5-turbo' (which proved to be the most efficient) to generate batches of sentences, giving for each batch a different set of keywords to generate a wide variety of sentences. The output is a list of lists, where each inner item contains both the words in a sentence in quotation marks separated by a comma and also the IOB tags for each word. The IOB tags is an extended standard to annotate data for NER tasks, as it captures the start, inside and outside of different named entities.

After a lot of prompt tuning and refinement we obtained over 1500 new labeled sentences. The total now added up to around 11000 labeled sentences.

3.2 Named Entity Recognition (NER)

Once the data is cleaned, organised and separated into different patients records we can proceed to the next step in the process, the extraction of named entities. These entities will serve as nodes in a Knowledge Graph later on and will give valuable insights about a given patient presented in a much simplified manner compared to the original text data.

3.2.1 Tools and resources

After researching on NER for dutch language we obtain a few libraries and resources that could potentially be useful for our task. Spacy and NLTK are two powerful Python libraries with the necessary tools to facilitate carrying out NLP related tasks, which includes NER with

support to several languages including Dutch. Furthermore it provides with easy access to several pretrained models for an specific task and language. NLTK is more research focused, as it offers a wider range of techniques and methods to implement for the same task, while this can be useful to compare the performance of different strategies. Spacy uses the state-of-the-art tools and methodologies to achieve the best performance. While these libraries offer great solutions for general purpose NER tasks, they were not as reliable for new domain specific entity tagging, as the models were trained to identify different entity categories. Trying them out on a subset of our dataset yielded very poor results and was unable to work for entities that it was not trained on as we had expected.

Due to the aforementioned reasoning, we decided to finetune an existing pretrained BERT model on a general Dutch NER task. We make use of HuggingFace, a platform that offers easy access to a huge library of transformer models, you can use the search bar to narrow down on the transformers based on the task they were trained for and the language used in the training process. We try out a few models filtering out by NER task and Dutch language and we notice that the models finetuned on a health related corpus of data were lacking in performance on our dataset, because even if the entity categories were similar, the text data used for finetuning differs in style and structure.

We made the decision to use a large pretrained BERT model on a Dutch corpus of data (conll-2002) and finetune it on the set of labeled sentences mentioned before provided that we adjust the model to the new set of entity tags:

- 'O'
- 'B-PERSON', 'I-PERSON'
- 'B-LOCATION', 'I-LOCATION'
- 'B-DISEASE', 'I-DISEASE'
- 'B-MEDICATION', 'I-MEDICATION'
- 'B-STATUS', 'I-STATUS'
- 'B-BODY', 'I-BODY'
- 'B-DOSAGE', 'I-DOSAGE'

The class entities Dosage, status and person are not treated as 'main entities' as there is not as many samples as the rest or offer very little extra information about the patients condition. Additionally, using POS tagging we were able to extract the dosage in more efficient manner treating it as a modifier to the medication entity.

3.3 Relation Extraction

Relation extraction is done in conjunction with the Named Entity extraction. The goal is to obtain triplets of data, comprising of the Patient ID, the relevant Named Entity and the relation that ties both of them together. To achieve this we loop over each sentence in a

given 'clientId' record of data, we extract every single entity in the sentence and using POS tagging rule based approaches and dependency parsing we obtain the ROOT of the sentence and gather all corresponding modifiers to that ROOT and to the entities found, such modifiers will then be added as the relation that completes the triplet of data. Data is written by people from different medical backgrounds and thus the structure of it varies a lot from sentence to sentence, making it harder to consistently extract relations following this approach. It is for that reason that we propose another simpler approach, where we take different predefined relations attending to the type of category the entity belongs to.

This process and all subprocedures involved are explained in a more detailed manner in the following chapter.

3.4 Recommendation system

The last step of the pipeline implements the recommendation system, built on top of a knowledge graph. We provide two approaches to the way recommendations are generated, one leverages the use of node embeddings and the other is based on the absolute amount of exact matches of entities between the target patient records and the current patient records to determine the degree of similarity.

3.4.1 Knowledge graph

Using the triplets extracted in the previous section we create a knowledge graph, in both Neo4j for visualization and querying purposes, and in `networkx` for analysis and experimentation.

We use `networkx` to create a multi-directed graph (to account for repeated relations between the same patient and entity pair) from which we obtain the most similar patients given a target patient and then we use Neo4j to query the target patient and the list of similar patients to visualize the connection these patients share between each other. This way a healthcare professional can see the most repeated entities across all similar patients and determine based on this what type of care they potentially may need.

3.4.2 Node embeddings

`Networkx` provides with a built-in `Node2Vec()` easy to implement algorithm to extract embeddings from the nodes, in our case we are only interested in obtaining the embeddings for the patient nodes and not all of the entities. Using the embeddings we can compute the similarity between two given vectors using the cosine similarity metric, which is proven optimal for this type of task.

Chapter 4

Experimental Setup and Results

In the previous chapter, we introduced the methodology that we developed to address the research problem of implementing a recommendation system out of a knowledge graph of patients healthcare history data. This new chapter provides a more detailed account of the practical and technical implementation of our methodology. We focus on the key steps in our pipeline, providing technical details for each step to give readers a more clear understanding of how we carried out the methodology.

Additionally, we present the data-driven results of our implementation, including an exhaustive analysis of its performance. Overall this chapter provides a comprehensive account of the technical execution and evaluation of our methodology.

4.1 Data processing

As we mentioned before, the data comes in a semi-structured source format, meaning each data sample is clearly differentiated from one another, but it is not really straightforward to know where each part of the content is (comment, ID, date, pulse...).

We know the ideal structure of a record, however, we notice from the table 4.1 that not every record has the same length in terms of items. We carry out different experimental approaches to determine the optimal way of importing the data into a workable Python object.

The data shown in table 4.1 has been modified due to the sensitive origin of it, but the structure is identical.

1. We try one first approach of assuming each new empty list we find marks the end of a record and the start of a new one, and within each records we separate text data from numerical data. We notice that only one field out of 32 contains text data and that is actually the 'comment' field which contains the healthcare notes made by the medical staff that are of most importance for this project.

Empty values and numbers that appear to add no value are initially discarded. Leaving us with dates, numerical identifiers (potentially client and employee IDs) and text data.

The main drawback of this approach is that records seem to be incomplete at first, as extracting them with this method yields some records with only text data, and no

Raw Data
['0001', ',', ',', '0001', '0001', '2023-01-01 12:00:00', '100', ',', ',', ',', ',', ',', ',', ',', ',', ',', 'Mw komt naar haar wekelijkse revisie, ze voelt zich goed']
['Mevrouw krijgt een griepvaccinatie van de huisarts. ']
', '0', '0', ',', ',', ',', ',', ',', ',', '1', ',', ',', '2023-01-01 12:00:00', '2023-01-01 12:00:00'
['0002', ',', ',', '0002', '0002', '2023-01-01 12:00:00', '100', ',', ',', ',', ',', ',', ',', ',', ',', ',', 'Mevrouw heeft een afspraak met de psychiater voor haar depressie. ']
['De heer moet een bloedonderzoek laten uitvoeren in het ziekenhuis. Mevrouw heeft last van migraine en neemt medicatie. ']
[]
['De heer krijgt fysiotherapie voor zijn knieblesure. ']
['Mevrouw wordt geopereerd aan haar gebroken arm. ']
[]
['De heer krijgt een allergietest bij de allergoloog. ']
', '0', '0', ',', ',', ',', ',', ',', ',', '1', ',', ',', '2023-01-01 12:00:00', '2023-01-01 12:00:00'

Table 4.1: Example of how data is presented in the original csv file

numerical data, without which we cannot determine the 'clientObjectId' value. So we had to assume that each record was from a different patient and that each patient would only have one centralised record, giving each record an identifier based on the order of appearance.

The resulting records following the approach 1 are displayed in table 4.2.

2. A second approach is to consider the end of a record and the start of a new one to be the appearance of a numerical identifier that can potentially be seen as the 'objectId' of the record. So this way each new entry found after the first row with an 'objectId' value will be seen as an extra comment value and appended to a list where all extra comments will be stored. This list is appended to the end of the record.

This second approach, while it organises data in a more efficient way than the first approach, has some inconsistencies itself. On the extra comments list it was common to find dates and numerical values which made it difficult to work, as there was not a common data type within the list, leading to more data cleaning afterwards.

The resulting records following the approach 2 are displayed in table 4.3.

3. The third and last approach tries to overcome all the weaknesses that previous approaches showcased. We made particular focus on the expected number of values per record, 32, and tried to fit all the data related to the same record in a 32 length vector. If we tried to import the raw data directly into a dictionary with 32 key-value pairs, we either had many keys with no assigned value or the values were not correctly assigned to their key.

Following an exhaustive analysis of various approaches we determined that the data needed is mostly there, but a restructuring of it was necessary. We extract all items of record and the respective length of it. In the vast majority of cases the length of

the record surpassed the 32 required values, this strongly suggests that the exceeding amount of records would have to be part of the 'comment' section of the record, as the rest of items were comprised of only 1 value. Unlike previous approaches where the excess of records were appended at the end, this time we take the difference between the actual length of the record and the expected length of the record, meaning for example if we find 35 items in a record, it means there are potentially 3 extra items for the 'comment' key value. The 'comment' key value is located in position 18 out of the 32, so we create a list and append items from position 18 to $(18 + X)$ to this list, X being the difference mentioned. Examining the resulting records.

The results obtained are clean and structured records, where the information available is located in their corresponding key values.

The resulting records following the approach 3 are displayed in table 4.4.

Approach 1
1: ['Mw komt naar haar wekelijkse revisie, ze voelt zich goed', 'Mevrouw krijgt een griepvaccinatie van de huisarts.', '2023-01-01 12:00:00', '2023-01-01 12:00:00', '2023-01-01 12:00:00', 'Mevrouw heeft een afspraak met de psychiater voor haar depressie.', 'De heer moet een bloedonderzoek laten uitvoeren in het ziekenhuis. Mevrouw heeft last van migraine en neemt medicatie.']
2: ['De heer krijgt fysiotherapie voor zijn knieblesure.', 'Mevrouw wordt geopereerd aan haar gebroken arm.']
3: ['De heer krijgt een allergietest bij de allergoloog.', '2023-01-01 12:00:00', '2023-01-01 12:00:00']

Table 4.2: Resulting records from approach 1.

Approach 2
<pre>{'objectId': '0001', 'carePlanEntryObjectId': '', 'carePlanLinkId': '', 'clientObjectId': '0001', 'employeeObjectId': '0001', 'reportingDate': '2023-01-01 12:00:00', 'report- TypeObjectId': '100', 'systolicPressure': '', 'diastolicPressure': '', 'pulsePressure': '', 'meanArterialPressure': '', 'heartRate': '', 'decimalValue': '', 'decimalValue2': '', 'mul- tipleChoiceValue': '', 'weight': '', 'length': '', 'bmi': '', 'comment': 'Mw komt naar haar wekelijkse revisie, ze voelt zich goed', 'healthRecords': ['Mevrouw krijgt een griepvac- cinatie van de huisarts.', '', '0', '0', '', '', '', '', '', '1', '', '', '2023-01-01 12:00:00', '2023-01-01 12:00:00']}</pre>
<pre>{'objectId': '0002', 'carePlanEntryObjectId': '', 'carePlanLinkId': '', 'clientObjectId': '0002', 'employeeObjectId': '0002', 'reportingDate': '2023-01-01 12:00:00', 'report- TypeObjectId': '100', 'systolicPressure': '', 'diastolicPressure': '', 'pulsePressure': '', 'meanArterialPressure': '', 'heartRate': '', 'decimalValue': '', 'decimalValue2': '', 'mul- tipleChoiceValue': '', 'weight': '', 'length': '', 'bmi': '', 'comment': 'Mevrouw heeft een afspraak met de psychiater voor haar depressie.', 'healthRecords': ['De heer moet een bloedonderzoek laten uitvoeren in het ziekenhuis. Mevrouw heeft last van migraine en neemt medicatie.', 'De heer krijgt fysiotherapie voor zijn knieblesure.', 'Mevrouw wordt geopereerd aan haar gebroken arm.', 'De heer krijgt een allergietest bij de allergoloog.', '', '0', '0', '', '', '', '', '', '1', '', '', '2023-01-01 12:00:00', '2023-01-01 12:00:00']}</pre>

Table 4.3: Resulting records from approach 2.

Approach 3
<pre>{'objectId': '0001', 'carePlanEntryObjectId': '', 'carePlanLinkId': '', 'clientObjectId': '0001', 'employeeObjectId': '0001', 'reportingDate': '2023-01-01 12:00:00', 'reportTypeObjectId': '100', 'systolicPressure': '', 'diastolicPressure': '', 'pulsePressure': '', 'meanArterialPressure': '', 'heartRate': '', 'decimalValue': '', 'decimalValue2': '', 'multipleChoiceValue': '', 'weight': '', 'length': '', 'bmi': '', 'comment': ['Mw komt naar haar wekelijkse revisie, ze voelt zich goed', 'Mevrouw krijgt een griepvaccinatie van de huisarts.'], 'flagged': '0', 'hidden': '0', 'hidingReason': '', 'parentObjectId': '', 'parentType': '', 'carenObjectId': '', 'carenName': '', 'carenRole': '', 'status': '1', 'episodeId': '', 'restrictiveMeasureId': '', 'createdAt': '2023-01-01 12:00:00', 'updatedAt': '2023-01-01 12:00:00'}</pre>
<pre>{'objectId': '0002', 'carePlanEntryObjectId': '', 'carePlanLinkId': '', 'clientObjectId': '0002', 'employeeObjectId': '0002', 'reportingDate': '2023-01-01 12:00:00', 'reportTypeObjectId': '100', 'systolicPressure': '', 'diastolicPressure': '', 'pulsePressure': '', 'meanArterialPressure': '', 'heartRate': '', 'decimalValue': '', 'decimalValue2': '', 'multipleChoiceValue': '', 'weight': '', 'length': '', 'bmi': '', 'comment': ['Mevrouw heeft een afspraak met de psychiater voor haar depressie.', 'De heer moet een bloedonderzoek laten uitvoeren in het ziekenhuis. Mevrouw heeft last van migraine en neemt medicatie.', 'De heer krijgt fysiotherapie voor zijn knieblessure.', 'Mevrouw wordt geopereerd aan haar gebroken arm.', 'De heer krijgt een allergietest bij de allergoloog.'], 'flagged': '0', 'hidden': '0', 'hidingReason': '', 'parentObjectId': '', 'parentType': '', 'carenObjectId': '', 'carenName': '', 'carenRole': '', 'status': '1', 'episodeId': '', 'restrictiveMeasureId': '', 'createdAt': '2023-01-01 12:00:00', 'updatedAt': '2023-01-01 12:00:00'}</pre>

Table 4.4: Resulting records from approach 3.

4.2 Named Entity Recognition

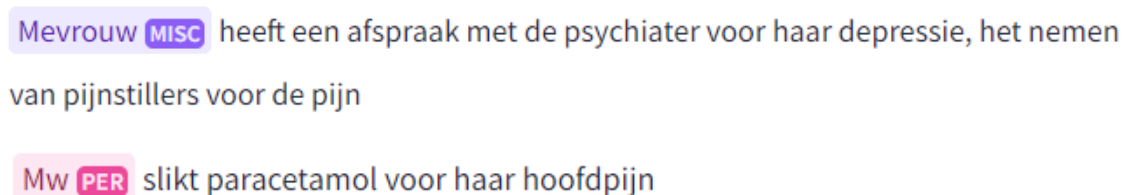
In this section we comprehensively present all work related to the final NER implementation in the project. We try to cover all steps and experiments embedded in the process in a very detailed manner.

The goal of this task is to find Named Entities related to healthcare in the texts. This process could be seen as a form of feature extraction, since it involves extracting meaningful pieces of information from text. In our case, these features or entities will preserve the essence of each patient as most of the information about the patients health condition is embedded into them.

After careful consideration we decided to make use of the HuggingFace library for this purpose. HuggingFace is a powerful open-source library that provides a wide range of resources for all NLP related tasks, as well as a good database with clear documentation and tutorials. It is also the host to many Large Language Models trained on corpus of data with different languages and topics, making it the ideal platform for leveraging powerful models with minimal setup and configuration.

On the web-based platform we can access models filtered mainly by NLP task and language used during training. For our specific needs, we selected Token classification and Dutch to explore the available models.

We curated our search to try and find models that had been fine-tuned on a healthcare or medical related topic and using tags as similar to our as possible. We had determined an initial 7 entity categories to extract from the data: DISEASE, LOCATION, BODY, MEDICATION, PERSON, STATUS and DOSAGE, that were later reduce to 4 main categories because the other categories were very scarce or even absent in some records as far as we could see during the labelling process. These main categories are: DISEASE, LOCATION, BODY and MEDICATION. On the HuggingFace platform we were able to find and test models whose data was from the medical and healthcare domain, but the results while better than before were still lacking in performance because these models were not finetuned on the same set of entities.



Mevrouw MISC heeft een afspraak met de psychiater voor haar depressie, het nemen van pijnstillers voor de pijn

Mw PER slikt paracetamol voor haar hoofdpijn

Figure 4.1: Model output for an example sentences.

As seen in the figure above 4.1 most of the models in HuggingFace trained for NER task on Dutch had a similar output, only being able to detect Mw as PERSON entity, even though they were also trained to identify conditions and diseases, words like 'depressie' or 'hoofdpijn' were still not detected as DISEASE entities.

To address this challenge, we propose two workarounds to overcome this issue. Firstly we suggest training an embeddings algorithm on our entire set of data, so that we can classify similar relevant words together. On the other hand, using a large pretrained model and finetuning it

on our downstream task to boost performance on our dataset. These proposed approaches explained below have the potential to yield better results for our task.

4.2.1 Word Embeddings

Word embeddings are dense vector representations of words in a continuous vector space, this format of representing data is beneficial for machines as it is more easily approachable than previously established methods due to the lower amount of dimensions needed to process text data.

Embeddings primary advantage is that it captures the semantic and contextual relationships among words in a sentence, meaning that it will no longer group words based on appearance together, but rather words that despite differing in how they look share a similar contextual usage.

For our use case, this can be of use to try and group words that relate for example to diseases or medications together, since it is very likely that such words appear in related syntactical structures and among similar words.

To train a word embeddings model we use the `gensim` python library, that provides an easy way to set up the training of such a model. By importing the `Word2Vec` function from `gensim` we can start training the model immediately, we only need to provide the data and evaluate the performance based on our choice of parameters.

1. We pass the data in the 'sentences' parameter, in our case we extracted all the text data from the 'comment' section of each one of the healthcare records, and stored this data in a List object called 'sentences'. From the 12,865,246 total records recovered from the dataset, we retrieved a total of 42,338,161 sentences that were used as input for the model, averaging around 3.3 sentences per record.
2. 'min_count' refers to the amount of times that a word must appear to be considered in the vocabulary, in our case a value of 5 was chosen after experimenting with lower and higher values. Lower values made the vocabulary size increase considerably as very low values usually include words that have been miswritten and offer no real value as well as words that are used very rarely and the ability to capture their context is limited. On the other hand greater values will miss out on potential words that don't appear too often but still contribute to adding context.
3. 'size' is used to set the amount of dimensions for the word vector representations. Higher values will be able to capture relations between words more effectively while at the same time having a higher risk of overfitting on the data and consuming more resources than lower values would, that are more suitable for a generalized model as they adapt better to unseen data. Ultimately we use a value of 300 dimensions, which has been scientifically proven to be optimal as stated by the authors in Yin and Shen (2018).
4. 'workers' indicates the number of threads to use during the training process, a higher value enables more computational power and faster training times.
5. Lastly we have 'iter' which refers to the number of epochs or times it loops through the entire data set. Avoiding high values of iterations is necessary to prevent overfitting

on the data. We experiment with different common values like 3, 5 and 10. Our best performing model was trained using 10 epochs, using more only causes the model to overfit as error and performance drastically change.

Lichaam		Benen		Paracetamol		Morfine	
Word	Score	Word	Score	Word	Score	Word	Score
lijf	0.7696	voeten	0.8254	pcm	0.9847	midazolam	0.8924
hoofd	0.6120	kniet	0.7919	paracetamol	0.9138	dormicum	0.8848
rug	0.5897	onderbenen	0.7906	oxycodon	0.8517	midazolan	0.8021
gezicht	0.5643	knien	0.7762	paracetamols	0.8472	nozinan	0.7720
linkerbeen	0.5576	armen	0.7205	panadol	0.8245	mofine	0.7561
rechterbeen	0.5554	bovenbenen	0.7187	oxynorm	0.8213	oxycodon	0.7476
bovenlichaam	0.5452	bennen	0.7131	tramadol	0.8161	oramorph	0.7405
buik	0.5418	knien	0.7085	paracetmol	0.8153	midazolam	0.7228
been	0.5409	enkels	0.7048	diclofenac	0.8079	buscopan	0.7198
armen	0.5397	linkerbeen	0.6966	paracetamollen	0.7971	oxynorm	0.7153

Table 4.5: Top results based on Word2Vec embeddings (Examples for Body parts and Medications).

Dementie		Kanker		Terwebloem		Bieslandhof	
Word	Score	Word	Score	Word	Score	Word	Score
alzheimer	0.8062	borstkanker	0.7848	triangel	0.9227	kreek	0.9189
ftd	0.7419	longkanker	0.7818	opmaat	0.9201	blh	0.9163
dementieel	0.7304	uitzaaiingen	0.7135	lindenhof	0.9027	naaldhorst	0.8769
psp	0.7235	darmkanker	0.7083	weidevogelhof	0.8900	triangel	0.8079
mci	0.7183	blaaskanker	0.7060	kreek	0.8729	terwebloem	0.7920
autisme	0.7173	prostaatkanker	0.7032	sonnevanck	0.8687	lindenhof	0.7861
cognitieve	0.7018	tumor	0.7026	akkerleven	0.8673	bies	0.7731
korsakov	0.6834	slokdarmkanker	0.6858	vlietzicht	0.8390	sonnevanck	0.7715
parkinsonisme	0.6772	botkanker	0.6775	lindehof	0.8242	marnix	0.7663
ziektebeeld	0.6767	kwaadaardige	0.6596	singelhof	0.8234	weidevogelhof	0.7547

Table 4.6: Top results based on Word2Vec embeddings (Examples for Diseases and Locations, health-care facilities).

As seen in both tables Table 4.5 and Table 4.6 we can see that having selected two relevant words for each main entity the most similar words for each of them is very closely related, so we come up with two approaches to leverage this potential.

1. Select a few words to serve as a baseline for each main entity for the entity classification task. Using POS tagging we extract the nouns in a sentence, avoiding determiners, adjectives, adverbs... knowing that in the vast majority of cases these entities are going to be nouns, we compare the word with each one of the baseline words and extract the similarity score. Ordering the similarity scores should give an idea of what type of word we are dealing with, if the higher similarity scores are with those words related to the

DISEASE entity tag we can assume that the word is also a disease only if it surpasses a certain threshold, because it could be the case that the highest similarity score is for a disease but the value is too low to be considered as such, meaning that it is not related to any entity tag.

While this approach works perfectly for some words, it lacks in performance for others that are not directly related to the baseline words but instead are related to other words that could also be considered as part of a main entity. To solve this problem we could increment the amount of words to use as baseline but it would exponentially increase the size of the vocabulary and the resources to use, making it less feasible.

2. The second approach involves clustering the word embeddings of all nouns, similar embeddings should be clustered together, so in theory words related to a topic or domain are to be clustered close to each other. Since the embeddings are vectors of 300 dimensions it is not possible to visualize the clusters, we have to use dimensionality reduction techniques to preserve the 2 or 3 principal components that preserve the most information about the vector.

This approach presents two issues, when transforming a high-dimensional space vector into a low-dimensional space vector there is a lot of information that is going to be missed since we are discarding 99% of components. Components that in the case of embeddings each preserves a similar amount of information about the original word, so cutting in components this drastically will result in unfavorable results, as expected. As we can see in Figures 4.2 and 4.3 there seems to be no logical pattern or distinguishable cluster, words seem to be randomly clustered forming a giant conglomeration in the middle. In the second Figure we have a zoomed-in version of the first one, and we can observe completely different words right next to each other. In Figure 4.4 every component, as mentioned before, adds a slightly bigger value than that after it, as the exponential plot suggests, reaching close to 100% cumulative variance with as many as 150-200 components (roughly half). This means that we would have to take into consideration many more than just 2/3 components, 68 as seen in Table 4.7, if we want to account for 80% of total cumulative variance, which is considered to be a required minimum, and up to 140 components for 95% total cumulative variance.

Cumulative variance [%]	N° of components
80	68
85	84
90	106
95	140

Table 4.7: Number of components per common cumulative variance values.

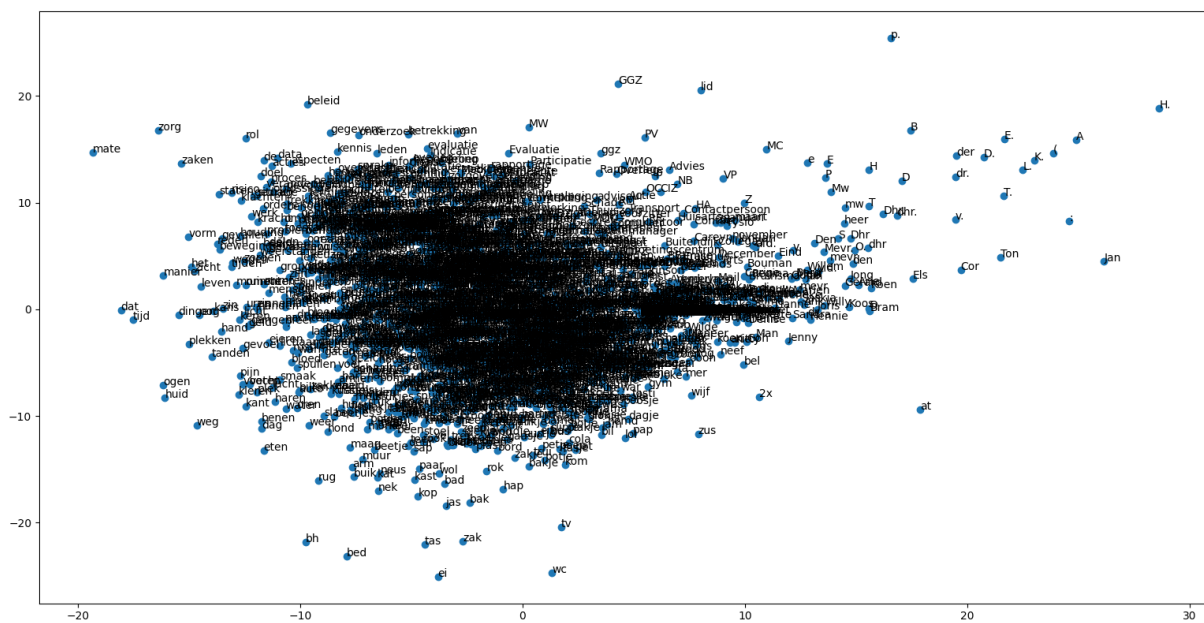


Figure 4.2: Visual representation of the word embeddings.

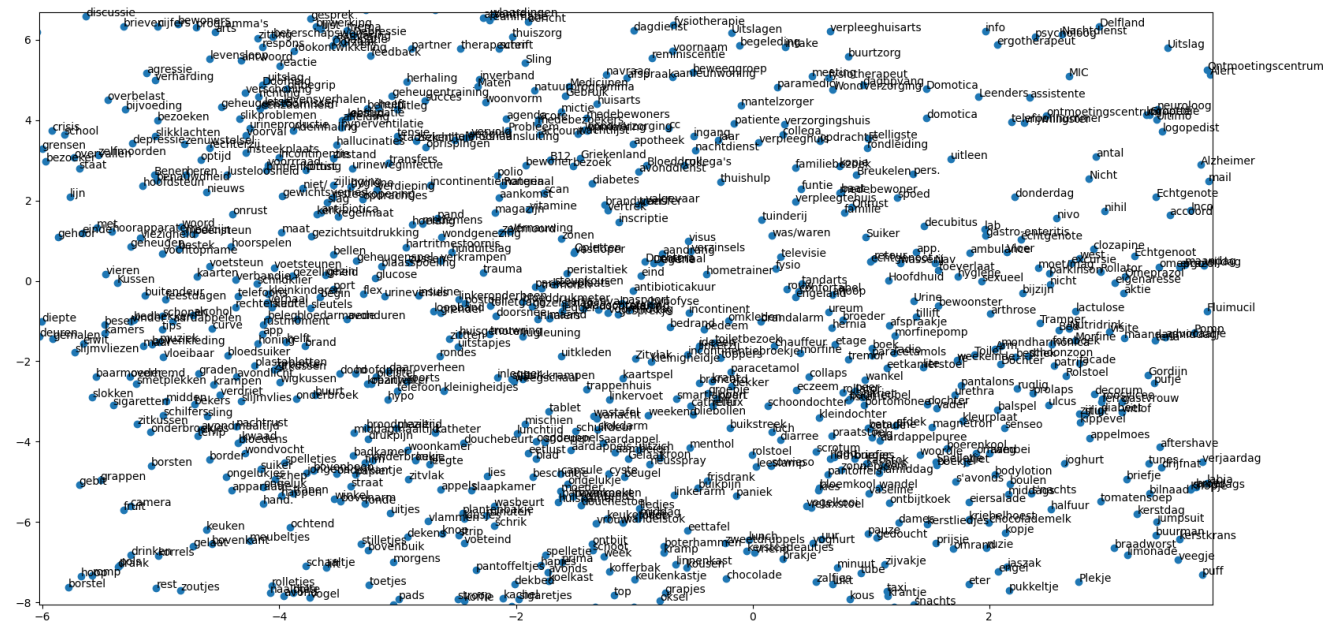


Figure 4.3: Zoomed-in visual representation of the word embeddings.

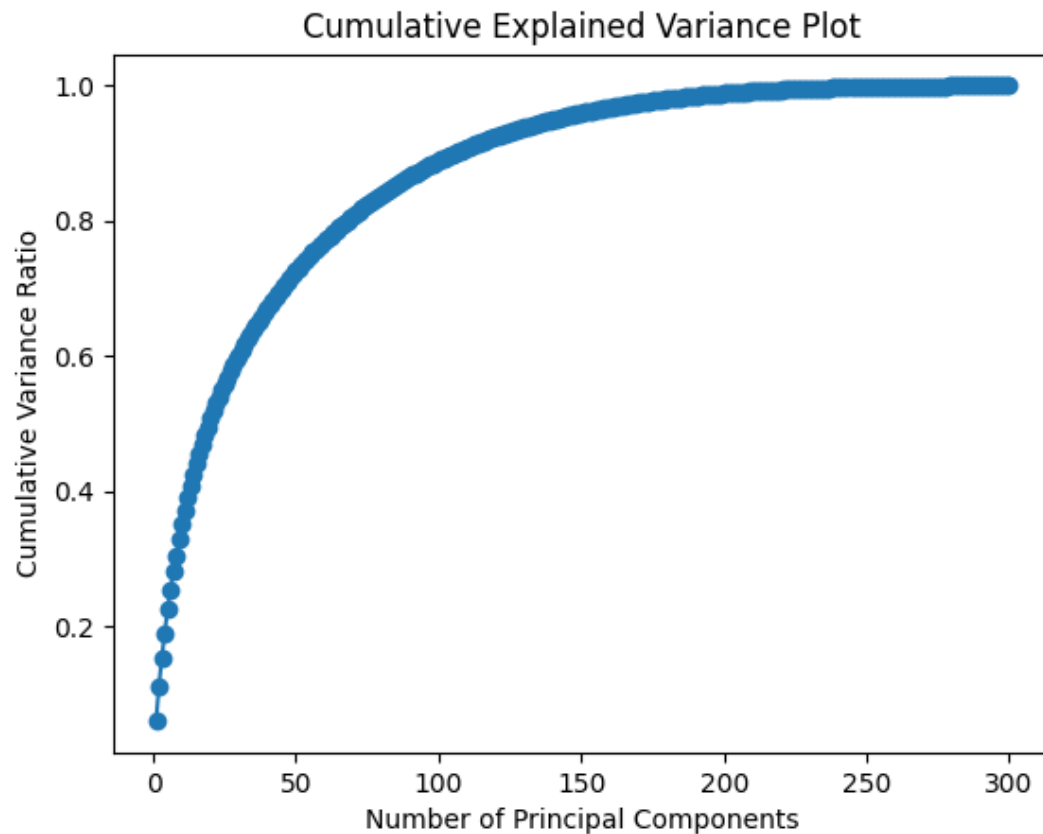


Figure 4.4: Cumulative explained variance plot.

We experiment clustering with different amounts of dimensions, preserving 80%, 85%, 90%, 95% and using the entire set of 300 dimensions. Unexpectedly, the results were far from acceptable, as words with different meanings were clustered in the same group in all the cases. One common technique to find the optimal number of clusters in a data set is the elbow method. This method poses an intuitive way of finding the optimal number of clusters by plotting the quality of clusters, that can be defined as to how well each amount of clusters represents the data, higher intra-cluster cohesion and lower inter-cluster separation yields better cluster quality. In the elbow plots, however, we didn't find any real elbow, implying the data doesn't follow any specific pattern. Modifying the number of clusters and exporting the words with their assigned cluster to a csv file we can see that the quality of the results didn't seem to improve from previous results. The main reason that can explain this behaviour is the lack in performance of clustering algorithms when it comes to handling very high dimensional data, hindering its ability to find meaningful distinct clusters.

For the first approach we obtained some promising results for words that are quite easy to classify, as it was able to consistently identify the category with which the word most relates to. We can see from the Table 4.8 below that the cosine distance for the adequate category/entity has a score way above the rest, making it a clear classification. All of the baseline words are related to the medical field, but each one of them is more closely related to one of the four main entities. The baseline words chosen have a very general meaning to avoid biases towards a certain type of diseases or a certain type of medications. In the table we have three sample words, where each one belongs to a different main entity. Highlighted in red we have the highest similarity scores per word, lighter red for second highest and from the results we can observe that it clearly identifies 'Paracetamol' as a medication, with almost double the score of the second highest which also indicates that it is a medicine, while the rest of the scores are around zero. The phenomena happens with 'Dementie' and 'Bieslandhof', it correctly identifies as their respective entities with a comfortable score margin, 'Dementie' as 'ziekte' (illness) and secondly as 'aandoening' (disease), 'Bieslandhof' as 'ziekenhuis', 'kliniek' and 'locatie' (hospital, clinic and location) all three synonyms of a care center.

	Paracetamol	Dementie	Bieslandhof
aandoening	-0.008	0.555	0.163
ziekte	-0.015	0.638	0.190
kwaal	0.047	0.173	0.093
botbreuk	0.031	0.295	0.007
kliniek	0.060	0.292	0.426
locatie	-0.066	0.177	0.392
chirurgie	0.055	0.126	0.229
orgel	-0.011	0.014	0.020
ziekenhuis	0.090	0.115	0.494
patient	0.086	0.327	0.180
medicatie	0.587	0.123	0.092
medicament	0.300	0.126	0.048
geneesmiddel	0.213	0.240	0.133

Table 4.8: Similarity metric between three samples words and the category words.

In some cases when comparing the target word with the category words resulted in confusing results, either the scores were too small to be considered as similar (below 0.2) or the most similar words were not from the correct category, we formulated a slightly different approach to overcome this. This approach consisted on taking the target word's most similar words based on their embeddings and comparing those similar words also to the category words, for each one preserving the highest score and category and ultimately having a more solid classification. The logic behind this is that, if a word is to be related to diseases, then the most similar words to it should also be related to diseases. Following this thought process we obtained more robust results, however this boost in performance was achieved for a very small portion of data as the time execution considerably grew due to the large amount of computations needed per step.

4.2.2 Fine-tuning transformer model

After all experimentation done it was quite clear that the best solution for our problem was most likely going to be to fine tune a Large Language Model already trained on some large corpus of data, as this has been proven by many researchers to yield state-of-the-art results across all domains.

As we mentioned before, to fine tune a model on our data to correctly identify entities it did not suffice to use a model fine-tuned on similar data, as it learned to capture contextual relations on another corpus of data, and while it may be perceived as similar for a person, the difference is very noticeable for a model.

Labeled data

We provide an explanation of all three sources used to obtain the labeled data to train our transformer model on.

1. Using Label Studio we manually label 2000 sentences.
2. We augment the data by creating synthetic new sentences labeled sentences from the ones already labeled in the previous source.
3. Using the chatGPT API to generate fake sentences, giving a set of instructions about formatting and vocabulary used to mimic as much as possible the structure of real sentences from the dataset.

As we can appreciate in Figure 4.5 we have a series of predefined entities that we can select to assign a specific word a different class. In this predefined set of entities we included three additional classes that we felt could add extra value to the patient's "profile".

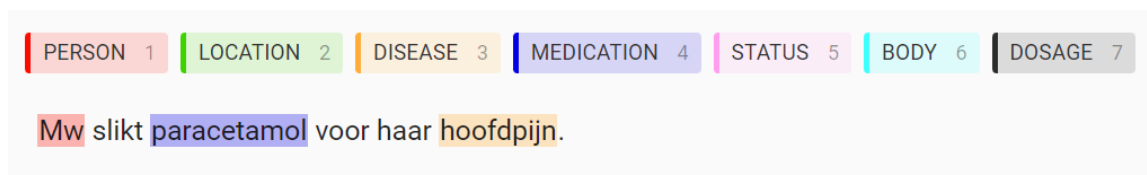


Figure 4.5: Example sentence in Label Studio.

Using this approach we labeled 1000 consecutive sentences to account for a wide variety of contexts since we would be labelling entire patients records (each one comprises of around 10 sentences). The second batch of sentences consisted on another 1000 sentences, this time these would be randomly selected from the dataset following one condition, that they include at least one main entity within to be labeled. The main entities being diseases, locations, body parts and medications.

Using both BeautifulSoup and Selenium, two web-scraping libraries for Python, we managed to create dictionaries with key words from each category. Employing these dictionaries we checked randomly in sentences whether any of the words was present in the sentence, once we gathered 250 sentences for a given entity, we repeated the process for another type of entity.

At the end we had selected 1000 were, at least 250 sentences were going to have a main entity, while in reality the number was slightly higher because sentences usually had several entities from different categories in them. This approach aimed to provide more contextual information to those entities that did not appear as frequently, so that the model would also learn to identify those.

Training using 2000 sentences is still not going to make for a good performing model, so using the dictionaries created from scraped data we consider augmenting the second batch of 1000 labelling sentences by creating synthetic data from it. We have 250 real sentences per main entity tag, we use these to create 2000 new sentences per entity tag. Selecting a sentence at random and replacing the entity with another new word from the dictionary, preserving the rest of sentence as is shown in Figure 4.6 for an example sentence where on the left side we just change the disease entity, and on the right hand side we change the medication entity. In some cases the same word from the dictionary was used but in different sentences, to account for a wider variety of contexts. This process was repeated four times for the four main entities, adding up to a total of 8000 new synthetic sentences. These sentences enriched the overall vocabulary of the training sentences. While some generated sentences may create illogical scenarios, like for example taking morphine for a headache, we want the model to learn the relation between taking medicine and having some type of pain for which the medicine is needed.

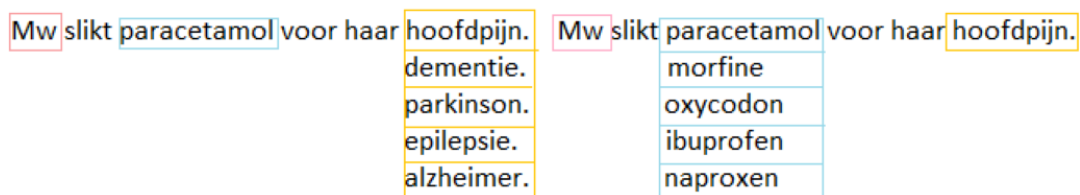


Figure 4.6: Example sentences using data augmentation.

ChatGPT

While the training set had seen a big increase in size, the structure, unlike the vocabulary, was lacking in diversity. Manually selecting valuable sentences from the millions of available samples would simply not be viable. With the rise in popularity of the chatGPT and the quality of some of its outputs we decided to make use of its API to create fake data.

When using the ChatGPT API there are two important things to consider: the model that we are going to use and the prompt used as input. Openai has many models at public disposal, each one having their strengths and weaknesses. At the time of writing, openai latest better performing models are from the GPT-3.5 family of models, an upgrade from the GPT-3 models. Within the 3.5 models we have: the text-davinci model, improved performance over curie, babbage, or ada in most language tasks, the code-davinci model, trained to excel in code related tasks and lastly the GPT-3.5-turbo, which is the most capable of all. For this task we choose the latter for two reasons, the quality of the outputs for this specific task was more complete and overall better, additionally the price per token of it was much lower, meaning we could experiment using different prompts without having to worry too much about going over

budget on tokens. The price per token in this model is 0.002 \$/1000 tokens, while the davinci model is 10 times that amount. The maximum amount of tokens produced per interaction for both models is 4096 tokens, which includes the tokens used in the prompt, this is important to try and make the prompt as compact as possible while not losing on important information.

A lot of time was spent trying to refine the prompt to have the output as close as possible to our goal (see Appendix for full detailed prompts used). The initial responses would lack consistency in the structure, but this would slowly change towards a more usable format with the addition of more specific instructions. The end formatting of the sentences should be identical to the ones we had already labeled to avoid future issues during the training process, this implied that for each new sentence we needed two lists of values: one with the words in quotation marks separated by commas, and another list with the corresponding IOB tags for each word. The IOB tags are vital for the training process, they signal the start and end of a given entity with the tags 'B' and 'I' respectively (Beginning and Inside of an entity) and the non-entity words with the 'O' tag (Outside of an entity).

To achieve the final prompt we had to provide a set of key instructions:

1. Acknowledging the consequences of using synthetic data. Sometimes the model would output an unexpected response implying that the request goes against its ethical standards or its policy "Sorry, I cannot fulfill this request as it goes against OpenAI's use case policy on generating fake data for medical purposes.", "Sorry, as an AI language model, I am not able to generate fake data that goes against ethical standards or promotes illegal activities." are some examples of it.
2. Indicating the goal of the prompt in a precise and concise manner.
3. Setting a minimum and maximum amount of words per sentence.
4. Explain the desired output of the generated data and provide some examples to serve as guidance, this last part is crucial.
5. Make emphasis on the formatting, by comparing it to python objects (list of lists).
6. Provide with example root verbs to use in the sentences
7. Limit the number of generated sentences to a fixed number for more consistency.
8. Lastly underline the importance of avoiding syntax errors and pay extra attention to the length of the lists (the list of words and list of IOB tags must have the same length).

Furthermore, to obtain sentences with a richer structural selection, we implemented a for loop to input a different prompt on each iteration and have higher chances of getting a completely different and unique response. Using a for loop meant that we could change slightly the prompt to have different outputs, the main changes that we implemented were the length of the sentences (minimum and maximum word count) and the main topic of the sentences. Having a list of around 50 different types of diseases we would change the topic between runs.

Additionally openai offers a few parameters to change the configuration of the model to generate different responses:

- `temperature`, it is a value ranging from 0 to 2. Higher values of temperature will make the model generate more creative and less deterministic responses, we change this value to between 0.1 and 0.5 between runs to consider riskier solutions.
- `presence_penalty`, value ranging from -2 to 2. A higher value will penalize the model when using words already present in the text so far.
- `frequency_penalty`, also ranging from -2 to 2. Raising this value will penalize the model for using popular or frequently used words, so a lower value will include a more extensive vocabulary.

Through a series of experiments we determine that the optimal values for these parameters are:

- `temperature`, random value from 0.1 to 0.5
- `presence_penalty`, random value from 0.1 to 0.4
- `frequency_penalty`, a value of 0. While it may seem logical to use a higher value to prioritize uncommon words, in reality the model would "hallucinate" and use words that are not entirely correct.

Moreover we pass 4 as the total number of chat completions per prompt, this means that we get four different responses for each unique prompt, speeding up the bulk process along the way.

In the end we managed to generate 4500 fake sentences, from which, despite inputting very comprehensive instructions, half of them did not meet the formatting criteria. There were inconsistencies in the length of the lists for each sentence and the type of IOB tags present, because sometimes with values just over zero for temperature the model would start to "forget" the required format or the specified set of predefined IOB tags. Ultimately we salvaged slightly over 1500 sentences with the desired format, which added to the existing set amounted to a total of 11500 labeled sentences to use during the training process of the model.

Training process

Making use of the HuggingFace transformers library we transform the labeled data into a `DatasetDict` object so that it can be fed to the model, additionally we use cross-validation to assess the performance of the model after training. With 5 folds the data is split into 80% training and 20% testing, using the non-annotated data (augmented data and fake data generated by chatGPT) for training and the annotated data for testing, after all 5 folds have been executed we obtain an estimation of the performance of the model taking into account the metric scores for each fold as seen in the Figures 4.9, 4.10 and 4.11 below.

	Fold 1		Fold 2		Fold 3		Fold 4		Fold 5	
	precision	recall	precision	recall	precision	recall	precision	recall	precision	recall
Body	0.86	0.90	0.86	0.89	0.85	0.88	0.90	0.91	0.84	0.89
Disease	0.89	0.88	0.88	0.89	0.88	0.89	0.90	0.90	0.83	0.86
Location	0.96	0.97	0.95	0.95	0.95	0.96	0.98	0.98	0.92	0.95
Meds.	0.87	0.88	0.84	0.85	0.87	0.89	0.86	0.88	0.82	0.86

Table 4.9: Precision and Recall scores for each fold evaluated on the test set.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Body	0.88	0.87	0.86	0.90	0.86
Disease	0.88	0.88	0.88	0.90	0.84
Location	0.96	0.95	0.95	0.98	0.93
Meds.	0.87	0.84	0.88	0.87	0.84

Table 4.10: F1 scores for each fold evaluated on the test set.

	Precision		Recall		F1 score	
	Mean	Standard Deviation	Mean	Standard Deviation	Mean	Standard Deviation
Body	0.862	0.020	0.894	0.010	0.874	0.015
Disease	0.876	0.024	0.884	0.014	0.876	0.019
Location	0.952	0.019	0.962	0.011	0.954	0.016
Meds.	0.852	0.019	0.872	0.015	0.860	0.017
	0.886	0.021	0.903	0.013	0.891	0.017

Table 4.11: Mean and Standard Deviation for the performance metrics: Precision, Recall and F1 scores.

4.3 Triplet extraction

After training yields a good performing model fit for this task we proceed to integrate all components of the pipeline together. We will go through each one of the steps involved in the final polished pipeline in a detailed manner.

4.3.1 Optimization

One of the problems we faced when trying to scale our pipeline to account for the entire dataset was the long running time of the script which was not sustainable, the script had to be optimised to reduce such time as much as possible. In addition to carefully reviewing the code for inconsistencies and bottlenecks a few strategies were considered before using the final approach.

1. Using Cython was the first option tried, it is a Python superset, so it has all features present in Python with additional functionality from C/C++ like static type declarations

for which this can be extremely useful.

Our code lacks numerical operations, we mainly work with strings of text or lists of strings, so the benefits are limited, yet we reformat our code to run using Cython. The code has two main functions, one to reconstruct the entities as some are split up into several tokens and another function to actually retrieve the triplets from the sentences. We run the code several times and for different batches of sentences and find out that just using static type declarations for all available variables in the function alone does not yield any consistent improvements over the Python version of the code. This is potentially due to the large amount of looping done that cannot be further optimised.

2. The second approach is to use multiprocessing to run the script using all cores in the CPU (by default python uses a single core) so we can have a significant boost in performance when using 8 cores compared to a single core. While it could be reasonable to think that it would be 8x times faster (when using 8 cores) reality differs from this. When running multiple cores in parallel there are some processes that have to wait for others to finish, slowing down the overall execution, the so called synchronization overhead. To avoid this as much as possible we have to segment the dataset into 8 equally sized chunks so that we can execute our code on each one without the need of any synchronization between them. While this makes for a very good improvement in runtime, it still is far from being 8x faster. This is due to the fact that the overall runtime is bounded by the time it takes a processing unit to process the chunk with the largest records, while all chunks are equal in size (amount of records they hold) not all records within have the same length.

In the Figure 4.7 we can appreciate that using more than one core can be extremely beneficial to run the pipeline faster. However we can also notice that it decreases steadily until reaching 4 cores, after that each extra core adds extra time. We can make sense of this when we consider the memory usage per core, using more than 4 cores takes up, collectively, more memory than we have RAM, resulting in a bottleneck where some processes have to wait for free memory to be made available to continue, slowing down the execution. On average using 4 cores is **30%** faster than using one core alone.

Multiprocessing

The first step is to divide the entire dataset into N chunks of records of equal length, N being the optimal number of cores (4 in our case). But because this would require that each process handles a large amount of memory and to mitigate the risk of a potential error we implement an extra safety measure, every 1000 records we save the triplets found to a file. So instead of dividing the entire dataset we would be dividing every 1000 records by N and reiterating this process until all records have been analysed.

4.3.2 Extracting Named Entities from the large collection

The next step is to apply our trained model to the records to obtain the Named Entities, for this task we use the pipeline class from HuggingFace, inputting the model and tokenizer.

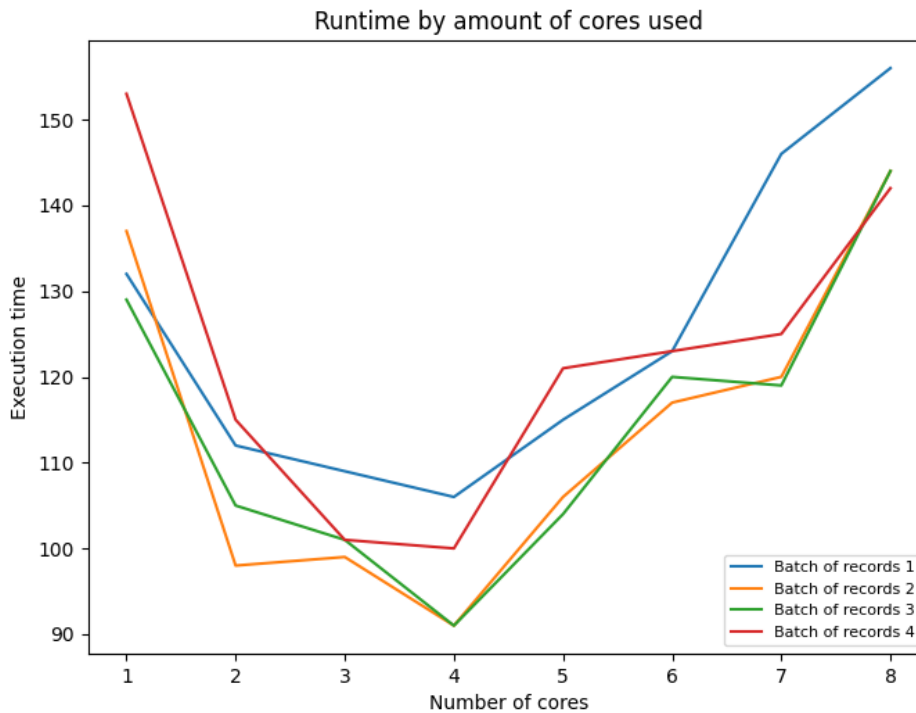


Figure 4.7: Performance comparison using varying cores.

After implementing the HuggingFace pipeline we have to do some formatting of the entity tags found, as there are going to be some words that are split into 2 or more tokens, so the actual words list and the tags/tokens list will no longer match in length. To fix this we have to loop over all tags and see which tags start with '##', this means the tag represents a subtoken of a word, we take the entity they represent and merge them together as one token and entity. In some rare cases we can find some I-tags with not previous B-tag or some I-tags after a B-tag where the tag is not the same on both cases meaning the model missclassified the token. The following Figure 4.8 illustrates this process.

4.3.3 Relation Extraction

The final step in the triplet extraction task is to find the relation that links the patient ID to the Named Entities found in the sentences of the records from that patient.

We propose two approaches for this task:

Standard approach

The standard approach involves using Part-of-Speech (POS) tagging and dependency parsing to identify the ROOT and verb of a sentence, as well as any words that modify the ROOT or the Named Entities. We loop over all of the tokens in the sentence and check whether any given token is an entity or not, if it is we skip it, if it is not we check what type of dependency class is attributed to that token. We are interested in the ROOT class because in most cases

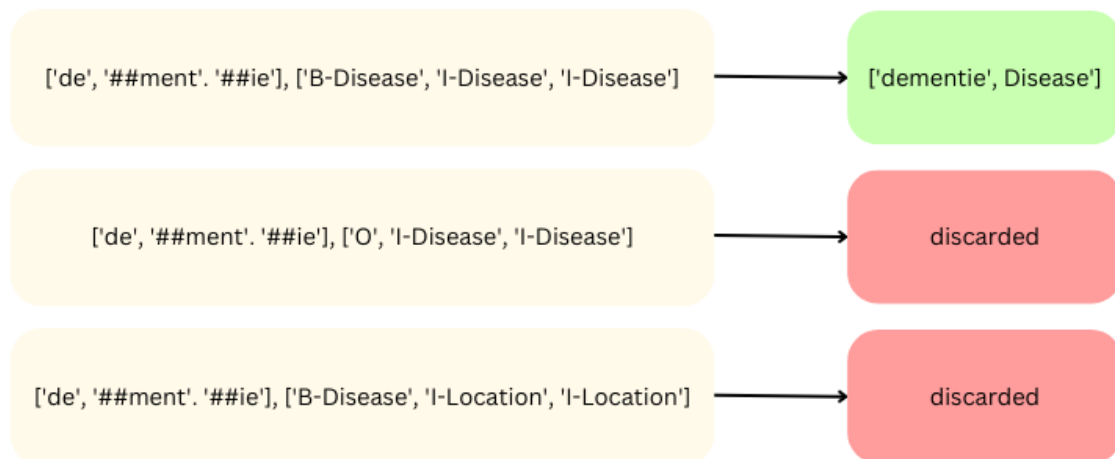


Figure 4.8: Entity reconstruction process.

this is the relation, the action that ties both the subject and the object in a sentence. The ROOT can offer valuable information, but it can be complemented with the modifiers that affect to it, it is not the same 'geopereerd' ('operated') and 'wordt niet geopereerd' ('is not operated'), where in both cases it is shown some kind of problem in the patient, in one case it was more severe since an operation was required whereas this is not the case for the other. This also applies to entities where 'plekken' ('spot') and 'rode plekje' ('red spot') where the latter offers extra information important about the patients skin condition, but we opted not to implement it for entities as a means of entity resolution to avoid having several similar nodes referring to the same thing.

We consider words to be modifiers if they fall under the following classes of dependencies: 'mod', 'acl', 'case', 'aux', 'det'. We include determinants as they usually link more modifiers even though they are not modifiers themselves and can make more expressive relations.

Additionally we look out for words whose dependency tag is 'punct' as it refers to punctuation, and check if they correspond to a period '.', in that case it means that we either has the end of the sentence, or we have a string containing multiple sentences, either way we have to reset the search and look for a new ROOT and modifiers.

Because the sentences lack consistency in the structure there can be sentences where there is no ROOT or the ROOT is one of the entities, for such cases we set the ROOT to be a predefined relation (see Table 4.12) depending on the type of entity that we are linking.

Simplified approach

Using the previous approach we obtain very descriptive relations that can sometimes not be as relevant or as precise as we would like, this can hinder later steps as we rely on the relations to be good. As an alternative we propose another approach, setting a predefined relation for every triplet depending on the type of entity, yielding more consistent results at the cost of potentially sacrificing details about the patient (see Table 4.12). All diseases, medications,

locations and body parts will have the same relation, making it easier to identify and filter out nodes in the Knowledge graph.

Entity type	Defined relation
Disease	'gediagnosticeerd met'
Location	'affiliatie'
Medication	'gebruikt'
Body	'in'

Table 4.12: Predefined relation labels.

To add more context we add extra triplets for both approaches relating the diseases to the body parts that they affect in the cases where both entities are mentioned in the same sentence. For example a sentence can be 'Patient-12345 heeft hoofdpijn en pijn in de rug en de leeg' we want to extract the following triplets from it:

- [Patient-12345, heeft, hoofdpijn]
- [Patient-12345, heeft, pijn]
- [pijn, in, rug]
- [pijn, in, leeg]

4.4 Recommendation system

The last step of the process is to use the triplets data to make a profile of a given patient and find similar patients based on this profile. These similar patients will then be used to extract the optimal careplan, using the careplan information attached to each similar patient. The reason we extracted triplets in the first place is to build a knowledge graph from them, which enables the storage of a rich semantic representation of the data.

4.4.1 Knowledge Graph

Knowledge Graphs (KGs) are a suitable option when we consider the fact that a patient's healthcare record can be many pages long, having a professional go through all of it just to have an understanding of the patients condition is not optimal, by the time they finish reading they would have forgotten key points of the record. Knowledge graphs address this issue by summarizing in a way the key aspects of what makes a patient different or similar to the rest.

Neo4j

The first option considered was Neo4j for a number of reasons: it is a native graph database, it has great scalability capabilities, meaning it can handle large amounts of data pretty effectively and has a large community behind it ensuring that any issue we may face is going to be easier to solve as someone else most likely encountered it first.

While it can be a bit challenging to use Cypher (its own query language) if you have no experience with it, the basic commands are very straightforward.

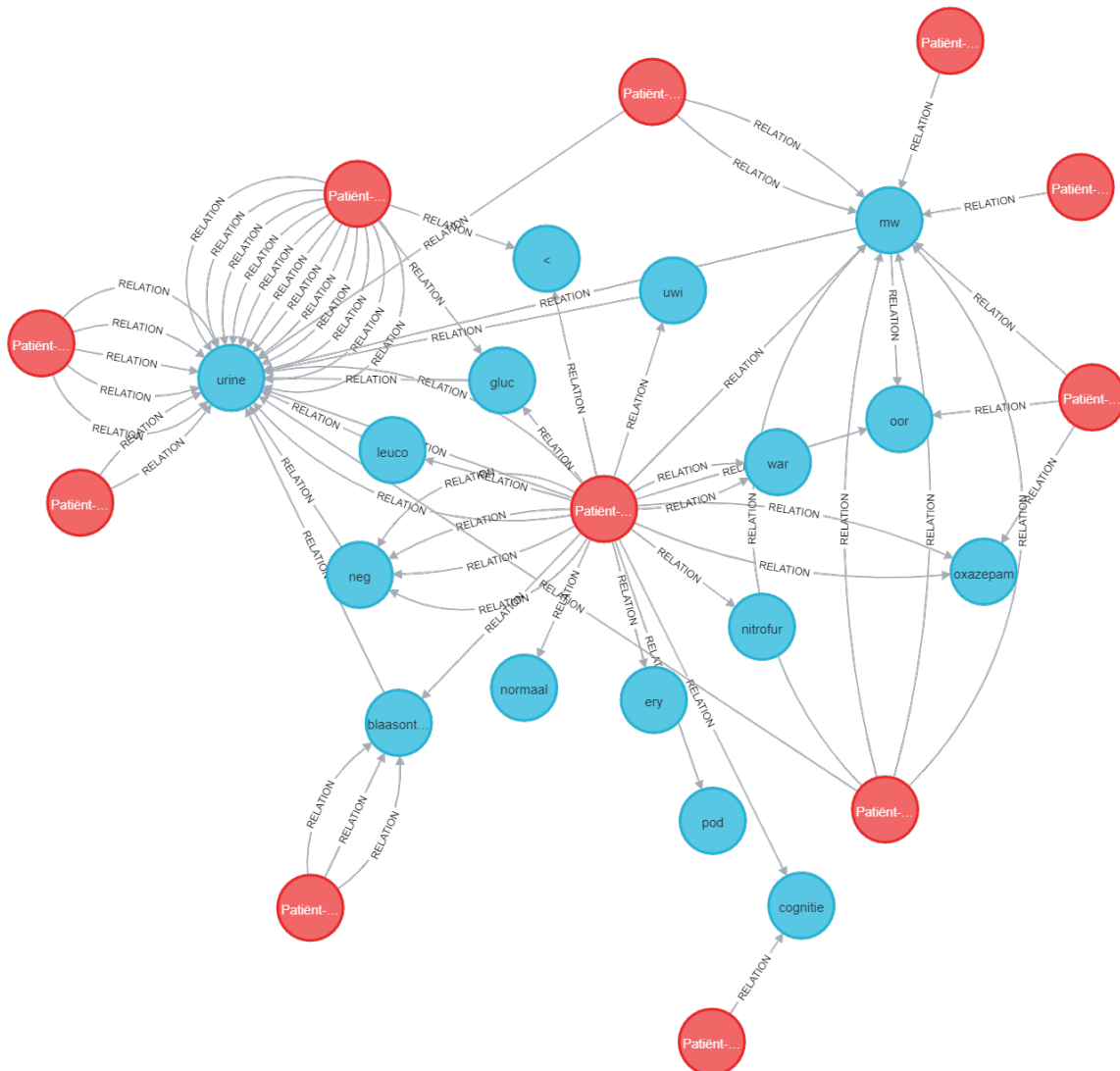


Figure 4.9: Small sample of the Neo4j graph visualization.

In the Figure 4.9 we show a small portion of the graph where we have in color red the patients nodes and with a blue color we have the neighbouring nodes (the Named Entities found). In this case we have expanded the patient found in the middle of the graph to showcase all of the Named Entities found for it and to exhibit how that patient node can relate to other patients who share similar connections. We can presume that those patient nodes that share a higher amount of similar connections are going to have a higher similarity score later on. For example in this case we notice that there is another patient that share many relations with the target patient by the entity 'urine', this can potentially mean that both patients share urine problems, we can notice the same thing for the entity 'blaasontsteking' which refers to 'Cystitis' in english meaning both shared a medical history with that condition. It can also be beneficial to retrieve patients that share the same type of medication like 'oxazepam' shared by the target patient

and another patient. These are some of the benefits of storing the information in a graph database.

Neo4j is great for visualization purposes, however for our project we needed to extract the embeddings of all the patients nodes so that we could compare them and while Neo4j offers a way of doing this it would require many more steps to get to the same point compared to using a python native graph database library like Networkx. Additionally for our project it was not necessary to store the information in a GDB (Graph Database), we only used the data to extract the node embeddings out of all the Patient nodes.

Networkx

As we just mentioned, Networkx was used to create the KG and in conjunction with the Node2Vec algorithm we trained a model to learn the embedding representations of the data. Python already provides a Node2Vec library and with the adjustment of just a few hyperparameters we can have our custom embeddings.

1. `dimensions`, the most important of all, will set the length for each vector representation. This value is advised to keep on the lower end (64 or 128 are common values) if the amount of information is limited, in order to not overfit to the data. However in our case we average 90 neighbours for diseases, 43 for medications and 19 for locations so a total average of 152 neighbours per patient so we can safely use a higher dimensionality value for our model with a lower risk of overfitting to the data. In the end a value of 300 is used, which is the standard for a medium to high volume dataset for its ability to capture information about the structure of the graph.
2. `walk_length`, is the value of connections that are going to be visited on each random walk, a bigger length value will allow for more exploration of the graph structure for each relevant node. We experiment with values between 1 and 50 because we are more interested in attaining information from close connections than in capturing long range dependencies. After considering values closer to 1 and values closer to 50 we determine that a value of around 5 had the best representative results.
3. `num_walks` represents the number of random walks performed with value `walk_length`. It should be taken into account the amount of neighbours each patient's node has before choosing a `num_walks` value that captures all the necessary information. As we mentioned, in our graph each patient node has an average of 150 nodes, so we execute 500 random walks to ensure each neighbour is visited more than once. A high value of random walks increases the quality of the embeddings as it will get a deeper understanding about the structure of the graph.
4. `p` and `q`, the first value denotes the likelihood that in a given random walk a node that has been seen will be revisited while the second indicates whether the random walk will reach nodes that are closer or farther away from the current node. For a balanced random walk we set `p` and `q` to 1.

4.4.2 Similar patients

With the embeddings we can obtain similar patients by comparing the target vector with the rest of vectors in the node2vec model. A popular metric used for this purpose is cosine similarity, it measures the angle between two vectors to determine using a score of -1 to 1 the similarity between the two, closer to 1 means it has a higher degree of similarity.

After computing the cosine similarity for a few embeddings we notice one thing, using all of the patient's neighbours to compute the embeddings was hindering the results, because in the same vector we had information about diseases, medications and locations related to that patient, so it was not clear as to why two patients were similar due to all the "noise" present. To fix this we computed the embeddings of the patients for each entity type separately, meaning we would compute the embeddings for the patient nodes only taking into account connections to diseases to compare patients against other patients solely based on diseases, and the same for locations and medications entities.

Using this method yielded a more precise outcome, to show the results in a more visually organised manner we display them in a QT for Python interface created with the library PySide2. For any given patient ID as input from the pool of available patients we compute the similarity of the embeddings of that patient and compare it against the rest, the top 10 best results are showcased in an ordered way in the interface, alongside some more information. To further emphasize on the similarity of any given pair of patients we provided an extra approach which essentially involved looking at the neighbours of the target patient compared to the rest of the patients, considering only the absolute amount of exact matches as the similarity metric, while also providing the relative amount to add more context.

As we can appreciate from the example image 4.11 (where patient IDs have been purposely left out for confidentiality reasons) on the left of the list of similar patients we have a percentage with a different shade of blue which indicates the relative amount of exact match neighbours out of the total amount of neighbours. On the right hand side of the patient a list with the ten words is provided, this list contains the most representative neighbours from such patient in regards to their similarity with the target patient. The way these 'most representative' neighbours are obtained differs for each approach:

- Using **embeddings**: computing cosine similarity on each of the neighbours from the most similar patients against the neighbours from the target patient to determine which connections have a higher score and thus add a bigger 'weight' towards the final similarity score. Retrieving the 10 with highest value to be displayed as part of the list (Figure 4.10).
- Using **exact matches**: obtaining and grouping all the neighbours from the most similar patients based on the amount of times they get repeated, from these values and for each patient we display the ones that appear less often, this will show conditions and diseases that are share mostly but that patient and the target patient, offering more insights into their similarities (Figure 4.11).

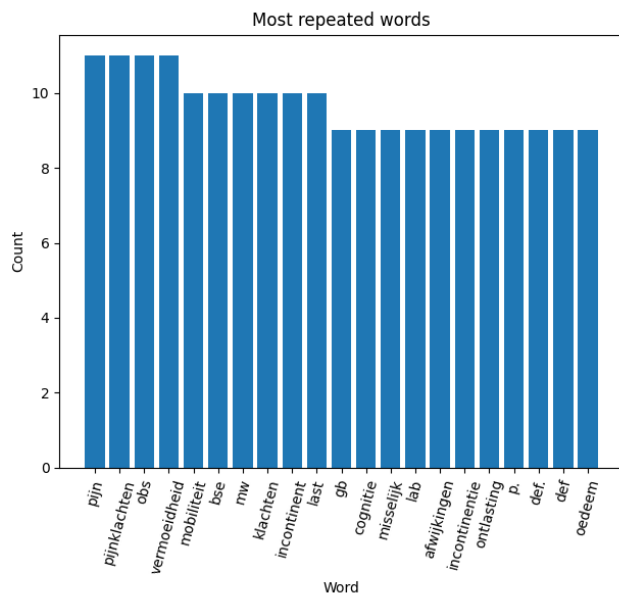
For both approaches it is made available three different plots: bar chart, wordcloud and bubble plot to help visualize the most repeated neighbours amongst the most similar patients. Sample images of the results using the same target patient as before are provided below, Figure 4.12.

1. Patient- [drusen', 'divertikel', 'thrapie', 'geplast', 'defjes', 'helemaal', 'discopathie', 'verneveld', 'rummicub', 'gentamycine', 'kalkarmoede']
2. Patient- [mdl-arts', 'medicatei', 'hemodynamisch', 'respiratoir', 'koorts-', 'pijn/andere', 'franse', 'vasculaire', 'spiriva', 'dubbelbeelden', 'akkkkoord']
3. Patient- [callusvorming', 'paravertebraal', 'welbevinde', 'inneme', 'stomaverpleegkundige', 'endocrinoloog', 'heelstrike', 'sachet', 'inzakkingen', 'vocht-voedingslijst', 'thoracolumbaal']
4. Patient- [b/', 'revlaidatie', 'spierverzet', 'geïnfecteerde', 'belastmobiliseren', 'bloeduiker', 'heelkunde', 'vorderingen', 'drukgevoelig', 'leukos', 'agnie']
5. Patient- [mengbeeld', 'geworden', 'billen/stuit', 'ondiepe', 'infectielab', 'rash', 'vloeibaar', 'trombosepoli', 'zithouding/decubitus', 'hypertrofie', 'longembolien']
6. Patient- [grijperig', 'defachterstand', 'meelzaam', 'binnensmonds', 'zwabberig', 'b.m.', 'slikbeoordeling', 'max.', 'kunstmatige', '-obs', 'resultaat']
7. Patient- [arthrodese', 'houding/rompoefening', 'ontstekingsremmend', 'sputum+', 'geactiveerd', 'vochtproblemen', 'abo', 'onderbeengips', 'aanmeten', 'ipramol', 'transvision']
8. Patient- [begeiding', 'verslikaccident', 'opknap', 'hypertentie', 'tijdelijk', 'rompfacilitatie', 'verstoor', 'inloopsnelheid', 'blaasinhoud', 'transformatie', 'vertraagd']
9. Patient- [gklachten', 'maag/darminfectie', 'neutral', 'catheriseren', 'afstnaden', 'incontinent', 'bewegingen', 'wegraking/duizelig', 'cognite', 'stijging', 'ingesleten']
10. Patient- [functionele', 'pijn', 'scoliosis', 'handwortel', 'excessief', 'subtrochantere', 'doorligplekje', 'visusstoornissen', 'vochtblaasjes', 'decomp', 'positief']

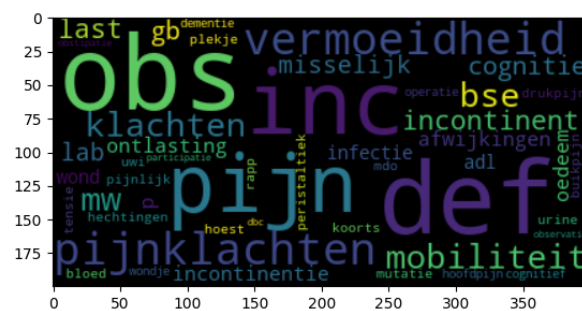
Figure 4.10: Example of the final Recommendation output using embeddings.

- 21.57% → 1. Patient- [wondinfectie', 'griekse', 'revalideren', 'weekly', 'ortho', 'mobiliteit', 'geopereerd', 'heupinfectie', 'week', 'heupwond']
- 16.25% → 2. Patient- [verplaatsen', 'drijfmat', 'pob', 'lactulose', 'mictieproblemen', 'esbl', 'pruimenmoes', 'uierne', 'nog', 'mevr.']
- 17.29% → 3. Patient- [tremor', 'meijer', 'slaperigheid', 'isolatie', 'trillingen', 'oh', 'infecties', 'gewichtstoename', 'schaafwondje', 'dosering']
- 18.41% → 4. Patient- [valpartij', 'het', 'geen', 'mn', 'wegrakingen', 'vlekken', 'bm.', 'hematoom', 'wondvocht', 'ecg']
- 16.25% → 5. Patient- [verzorging', 'pg', 'infectiewaarden', 'geheugenfitness', 'e.coli', 'chronische', 'risico', 'gewichtstoename', 'defecatie', 'elektrisch']
- 20.19% → 6. Patient- [spierkracht', 'extensie', 'hb', 'geheugenproblemen', 'dubbeltaken', 'bacterieen', 'mmol', 'vochtbalans', 'stabiliteitstraining', 'insziens']
- 27.39% → 7. Patient- [rontgen', 'parkinsonnet', 'bekkenfractuur', 'mediale', 'vruchtenkwark', 'down', '}', 'dosering', 'parkinson', 'obs/rap']
- 17.67% → 8. Patient- [documenten', 'imponeert', 'maatschappelijk', 'herstelunit', 'pacemaker', 'kneuzing', 'infecties', 'val', 'een', 'ook']
- 15.71% → 9. Patient- [z.s.m.', 'verminderd', 'mondhygiene', 'behandeling', 'pijnscore', 'slikfunctie', 'pijnbeleving', 'ergotherapeut', 'leuco', 'en']
- 21.41% → 10. Patient- [logopedische', 'verdikking/bultje', 'reva', 'eetobservatie', 'svp', 'domperidon', 'correctie', 'te', 'insteek', 'coördinatie']

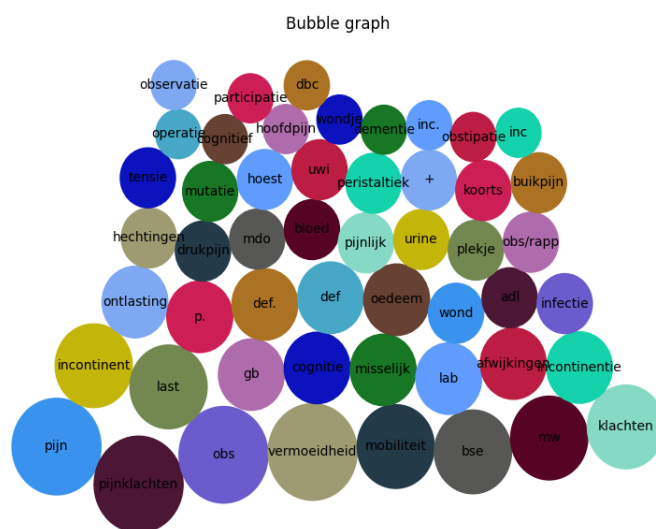
Figure 4.11: Example of the final Recommendation output using exact matches.



(a) Bar Chart



(b) Wordcloud



(c) Bubble graph

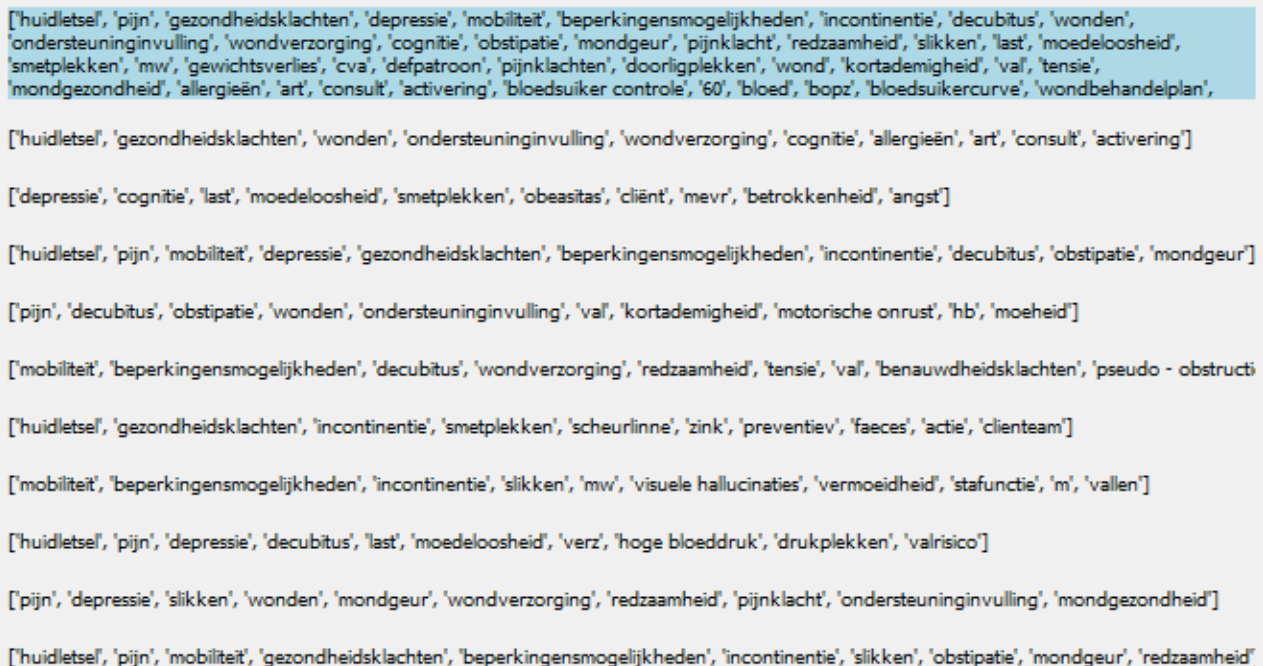
Figure 4.12: Visualization plots with most repeated neighbours across similar patients (same target patient).

4.4.3 Similar careplans

Finally we were given access to another dataset where the careplan facilitator stores information about specific needs and goals of each individual patient separately, with a lot of detail in most cases.

The goal is to predict careplan goals for any given patient based on keywords found in the most similar patient's careplans. First we extract the careplans for the top 10 most similar patients for a given target patient, from these careplans we extract every piece of text present in them that may have some information about the situation of the patient. Applying our previously trained NER model to this text we extract the named entities from within which will represent the careplan for that patient. Grouping all the keywords in order of occurrence we obtain a dictionary of key value pairs where the keys are the named entities and the values are the amount of times each one appears in different careplans. We take the highest values to be the most representative of the target patient situation and we display them to serve as guidance as to what type of care the patient is most likely to need.

In the Figure 4.13 are exhibited the keywords found on each careplan for the same top 10 similar patients, ordered from most to least importance within each list of words. With a blue background we have a larger quantity of keywords that would comprise the optimal careplan for the target patient, being a combination of keywords from all 10 patients. In this particular case, based on our analysis and assumptions derived from the work carried out in the project, it can be seen that the patient would require care for skin related problems, incontinence, depression, cognitive aid, pressure ulcers... as these conditions are mentioned often in the careplans of its most similar patients.



```
[ 'huidletsel', 'pijn', 'gezondheidsklachten', 'depressie', 'mobiliteit', 'beperkingensmogelijkheden', 'incontinentie', 'decubitus', 'wonden', 'ondersteuninginvulling', 'wondverzorging', 'cognitie', 'obstipatie', 'mondgeur', 'pijnlucht', 'redzaamheid', 'slikken', 'last', 'moedeloosheid', 'smetplekken', 'mw', 'gewichtsverlies', 'cva', 'defpatroon', 'pijnlachten', 'doorligplekken', 'wond', 'kortademigheid', 'val', 'tensie', 'mondgezondheid', 'allergieën', 'art', 'consult', 'activering', 'bloedsuiker controle', '60', 'bloed', 'bopz', 'bloedsuikercurve', 'wondbehandelplan',
```

```
[ 'huidletsel', 'gezondheidsklachten', 'wonden', 'ondersteuninginvulling', 'wondverzorging', 'cognitie', 'allergieën', 'art', 'consult', 'activering' ]
```

```
[ 'depressie', 'cognitie', 'last', 'moedeloosheid', 'smetplekken', 'obeasitas', 'cliënt', 'mevr', 'betrokkenheid', 'angst' ]
```

```
[ 'huidletsel', 'pijn', 'mobiliteit', 'depressie', 'gezondheidsklachten', 'beperkingensmogelijkheden', 'incontinentie', 'decubitus', 'obstipatie', 'mondgeur' ]
```

```
[ 'pijn', 'decubitus', 'obstipatie', 'wonden', 'ondersteuninginvulling', 'val', 'kortademigheid', 'motorische onrust', 'hb', 'moeheid' ]
```

```
[ 'mobiliteit', 'beperkingensmogelijkheden', 'decubitus', 'wondverzorging', 'redzaamheid', 'tensie', 'val', 'benauwdheidsklachten', 'pseudo - obstructi' ]
```

```
[ 'huidletsel', 'gezondheidsklachten', 'incontinentie', 'smetplekken', 'scheurlinne', 'zink', 'preventiev', 'faeces', 'actie', 'clienteam' ]
```

```
[ 'mobiliteit', 'beperkingensmogelijkheden', 'incontinentie', 'slikken', 'mw', 'visuele hallucinaties', 'vermoeidheid', 'stafunctie', 'm', 'vallen' ]
```

```
[ 'huidletsel', 'pijn', 'depressie', 'decubitus', 'last', 'moedeloosheid', 'verz', 'hoge bloeddruk', 'drukplekken', 'valrisico' ]
```

```
[ 'pijn', 'depressie', 'slikken', 'wonden', 'mondgeur', 'wondverzorging', 'redzaamheid', 'pijnlucht', 'ondersteuninginvulling', 'mondgezondheid' ]
```

```
[ 'huidletsel', 'pijn', 'mobiliteit', 'gezondheidsklachten', 'beperkingensmogelijkheden', 'incontinentie', 'slikken', 'obstipatie', 'mondgeur', 'redzaamheid' ]
```

Figure 4.13: Example of the final careplan recommendation output.

Chapter 5

Discussion

In this chapter we go over the findings gathered over previous chapters. We try to cover all aspects of the exploration involved in the development of the project.

Firstly, we go back to the main research question of the thesis, "Can a knowledge graph-based recommender system provide context for predicting the goals of a careplan?" and the initial objectives related to it. The main goal of the project is to provide with the necessary resources to obtain the optimal careplan for any given patient using the similar patients' data as guidance. To achieve this we have defined smaller task goals for each relevant part of the pipeline.

As explained before, our research methodology follows a structured pipeline that facilitates systematic experimentation and optimization of each step of the process. By outlining the sequential flow of the tasks involved in the pipeline we can evaluate them effectively and refine the methods were needed.

For the Named Entity Recognition task we first try using word embeddings and as seen by the results obtained it has a great potential to be used to classify tokens in different entity categories, but it lacks in performance when it comes to clustering similar words together due to the high dimensionality of the vectors and the inherent skew in word frequency distribution, certain words having higher occurrence rate than others often results in a less accurate word embedding for the latter, this is the main reason we decided not to go forward with this approach in the end but still portrayed the results and possible future implementations using word embeddings.

For the next step, relation extraction, we considered a couple of strategies, the first one was using a multi-lingual BERT based pretrained model to extract the relation linking the entity and the patient together, this first method had to be discarded early on due to its inability to capture coherent relations in our text data, mostly because the corpus of data used in the training of such model differs a lot from our actual data in terms of structure. Our data is very sparse as it has been written by different people with different professional backgrounds.

Another technique that we thought was worth looking into and that we decided to proceed with was using POS tagging to extract relations based on the semantic meaning of words within a sentence. This approach seemed to perform decently even when sentences were half completed or had any type of grammatical or logical errors in them. Although using POS tagging was not always correct we defined a series of rules to account for a bigger variety

of contexts in addition to gathering all possible modifiers that may affect such relation, for example in the case of "needs surgery" it is not the same as "does not need surgery", where in both cases a medical condition can be inferred, in the first case we can assume it is more serious as it does need surgery.

Lastly, after extracting triplets from the text we want to be able to visualize this data in a way that it makes sense to a medical professionals, as this project is also intended to save time to healthcare professionals in charge of reviewing records of patients. Our first option considered was Neo4j, but additionally to visualizing the data, we wanted to extract the node embeddings of the patients based on the type of links (diseases, locations or medications) to have a vector representation of a patient's connections which can be used for comparison and thus for recommendation, so we used Networkx for this purpose.

For that reason we trained the node to vector model using the built-in model from Networkx and saved the resulting vectors into an external file. During the training process we had to experiment with hyperparameters until finding the optimal set-up which yielded the best results after comparing several results on different values. Finally we just needed to separate the neighbours of each patient attending the entities they had, so that we don not compare diseases against medications and this way the results would be more representative of the patients condition. Gathering the key findings by most repetition amongst the top similar patients would provide us with a clear idea of the patient's particular needs and care goals, serving as a guideline for care professionals who would ultimately make a choice supported by this data.

Chapter 6

Conclusion

The aim of this project was to answer a key research question, "Can a knowledge graph-based recommender system provide context for predicting the goals of a careplan?". Throughout the development of the thesis a lot of research and experimentation was carried out to overcome the challenges of named entity extraction in unstructured texts from the medical field.

We managed to develop a pipeline that can successfully extract triplets of information from any given record of patient data and use this information to build a recommender system following several approaches. We provided sufficient evidence to support the choice of resources used during the entirety of the thesis.

We explored the potential of word embeddings and showed some promising results that can be very beneficial for other purposes as the performance was better than anticipated in some aspects. Our findings underline the potential behind training a custom word embeddings model when we have large amounts of data available, as well as some limitations regarding classification tasks.

Moreover, we went over the steps involved and motivation behind fine tuning a NER model for entity extraction showcasing that in cases where the data is unique, performance of a finetuned model will in most cases exceed that of a model pretrained on some other corpus of data due to disparities in the data structure.

Additionally, we provided two approaches to obtain the relations that tie the patient and the named entity, the rule-based approach leverages the use of POS tagging to collect the ROOT and any meaningful modifiers that add context to the relation, while the secondary approach uses predefined relations depending on the type of entity to reduce the risk of miss-classification in the main approach and to facilitate succeeding tasks.

Lastly we introduced knowledge graphs and how we benefited from node embeddings to compare and rank patients' data given a target patient. This enabled us to identify relations between similar patients and ultimately provide the healthcare facilitator with key information about the patients careplan and specific needs.

The results obtained from the recommendation system, while quite good they are still below human level. The main reason for this is the limited amount of labeled data available to finetune the model. Slightly over 10000 sentences were used in the training, but most of them had a similar structure limiting the ability of the model to correctly classify important named entities, or in some cases missing them completely.

We propose two improvements, one is to add more variety and quality to the set of labeled data to retrain the model, improving the chances of correctly classifying entities. Secondly and more importantly some kind of filtering mechanism needs to be implemented in order to discard entities seen as diseases, like "pain", "complaints", "fatigue"... which are terms too broad and generic that hinders the ability to distinguish different patients due to the high amount of times these words appear, adding layers of noise to the final result and making more meaningful entities go unnoticed.

In conclusion this thesis tries to cover all steps involved in order to build a recommendation systems from domain specific data, assessing the quality and motivating the selection of all resources used.

Bibliography

- Akbik, Alan, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. “FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP.” In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, 54–59. Minneapolis, Minnesota: Association for Computational Linguistics, June. <https://doi.org/10.18653/v1/N19-4010>. <https://aclanthology.org/N19-4010>.
- Barroca, João, Abhishek Shivkumar, Beatriz Quintino Ferreira, Evgeny Sherkhonov, and João Faria. 2022. *Enriching a Fashion Knowledge Graph from Product Textual Descriptions*. arXiv: 2206.01087 [cs.IR].
- Delobelle, Pieter, Thomas Winters, and Bettina Berendt. 2020. “RobBERT: a Dutch RoBERTa-based Language Model.” *CoRR* abs/2001.06286. arXiv: 2001.06286. <https://arxiv.org/abs/2001.06286>.
- . 2022. *RobBERT-2022: Updating a Dutch Language Model to Account for Evolving Language Use*. arXiv: 2211.08192 [cs.CL].
- Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” *CoRR* abs/1810.04805. arXiv: 1810.04805. <http://arxiv.org/abs/1810.04805>.
- Gorinski, Philip John, Honghan Wu, Claire Grover, Richard Tobin, Conn Talbot, Heather Whalley, Cathie Sudlow, William Whiteley, and Beatrice Alex. 2019. “Named Entity Recognition for Electronic Health Records: A Comparison of Rule-based and Machine Learning Approaches.” *CoRR* abs/1903.03985. arXiv: 1903.03985. <http://arxiv.org/abs/1903.03985>.
- Grover, Aditya, and Jure Leskovec. 2016. “node2vec: Scalable Feature Learning for Networks.” *CoRR* abs/1607.00653. arXiv: 1607.00653. <http://arxiv.org/abs/1607.00653>.
- Gururangan, Suchin, Ana Marasovic, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. “Don’t Stop Pretraining: Adapt Language Models to Domains and Tasks.” *CoRR* abs/2004.10964. arXiv: 2004.10964. <https://arxiv.org/abs/2004.10964>.
- Hamilton, William L., Rex Ying, and Jure Leskovec. 2017. “Inductive Representation Learning on Large Graphs.” *CoRR* abs/1706.02216. arXiv: 1706.02216. <http://arxiv.org/abs/1706.02216>.
- Kim, Yoon. 2014. “Convolutional Neural Networks for Sentence Classification.” *CoRR* abs/1408.5882. arXiv: 1408.5882. <http://arxiv.org/abs/1408.5882>.

- Kocaman, Veysel, and David Talby. 2020. “Biomedical Named Entity Recognition at Scale.” *CoRR* abs/2011.06315. arXiv: 2011.06315. <https://arxiv.org/abs/2011.06315>.
- Lebret, Rémi, and Ronan Lebret. 2013. “Word Embeddings through Hellinger PCA.” *CoRR* abs/1312.5542. arXiv: 1312.5542. <http://arxiv.org/abs/1312.5542>.
- Lee, Jinhyuk, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019. “BioBERT: a pre-trained biomedical language representation model for biomedical text mining.” *CoRR* abs/1901.08746. arXiv: 1901.08746. <http://arxiv.org/abs/1901.08746>.
- Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. “RoBERTa: A Robustly Optimized BERT Pretraining Approach.” *CoRR* abs/1907.11692. arXiv: 1907.11692. <http://arxiv.org/abs/1907.11692>.
- Nath, Namrata, Sang-Heon Lee, and Ivan Lee. 2022. “NEAR: Named entity and attribute recognition of clinical concepts.” *Journal of Biomedical Informatics* 130 (June): 104092. <https://doi.org/10.1016/j.jbi.2022.104092>. <https://doi.org/10.1016%2Fj.jbi.2022.104092>.
- Nguyen, Dat Quoc, and Karin Verspoor. 2018. “From POS tagging to dependency parsing for biomedical event extraction.” *CoRR* abs/1808.03731. arXiv: 1808.03731. <http://arxiv.org/abs/1808.03731>.
- Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. 2014. “DeepWalk: Online Learning of Social Representations.” In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710. KDD ’14. New York, New York, USA: Association for Computing Machinery. ISBN: 9781450329569. <https://doi.org/10.1145/2623330.2623732>. <https://doi.org/10.1145/2623330.2623732>.
- Sarhan, Injy, and Marco Spruit. 2021. “Open-CyKG: An Open Cyber Threat Intelligence Knowledge Graph.” *Knowledge-Based Systems* 233:107524. ISSN: 0950-7051. <https://doi.org/https://doi.org/10.1016/j.knosys.2021.107524>. <https://www.sciencedirect.com/science/article/pii/S0950705121007863>.
- Schneider, Phillip, Tim Schopf, Juraj Vladika, Mikhail Galkin, Elena Simperl, and Florian Matthes. 2022. *A Decade of Knowledge Graphs in Natural Language Processing: A Survey*. arXiv: 2210.00105 [cs.CL].
- Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. “Sequence to Sequence Learning with Neural Networks.” *CoRR* abs/1409.3215. arXiv: 1409.3215. <http://arxiv.org/abs/1409.3215>.
- Szubert, Ida, and Mark Steedman. 2019. “Node Embeddings for Graph Merging: Case of Knowledge Graph Construction.” In *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing (TextGraphs-13)*, 172–176. Hong Kong: Association for Computational Linguistics, November. <https://doi.org/10.18653/v1/D19-5321>. <https://aclanthology.org/D19-5321>.
- Turki, Houcemeddine, Abraham Toluwase Owodunni, Mohamed Ali Hadj Taieb, Ren’e Fabrice Bile, Mohamed Ben Aouicha, and Vilém Zouhar. 2023. “A Decade of Scholarly Research on Open Knowledge Graphs.” *ArXiv* abs/2306.13186.

- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. “Attention is all you need.” *Advances in neural information processing systems* 30.
- Verkijk, Stella, and Piek Vossen. 2021. “MedRoBERTa.nl: A Language Model for Dutch Electronic Health Records” [in English]. *Computational Linguistics in the Netherlands* 11 (December): 141–159. issn: 2211-4009.
- Vries, Wietse de, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. 2019. “BERTje: A Dutch BERT Model.” *CoRR* abs/1912.09582. arXiv: 1912.09582. <http://arxiv.org/abs/1912.09582>.
- Yin, Zi, and Yuanyuan Shen. 2018. “On the Dimensionality of Word Embedding.” *CoRR* abs/1812.04224. arXiv: 1812.04224. <http://arxiv.org/abs/1812.04224>.
- Zhang, Yue, Bo Zhang, Rui Wang, Junjie Cao, Chen Li, and Zuyi Bao. 2021. “Entity Relation Extraction as Dependency Parsing in Visually Rich Documents.” *CoRR* abs/2110.09915. arXiv: 2110.09915. <https://arxiv.org/abs/2110.09915>.
- Zhou, Sicheng, Liwei Wang, Nan Wang, Hongfang Liu, and Rui Zhang. 2022. *Cancer-BERT: a BERT model for Extracting Breast Cancer Phenotypes from Electronic Health Records*. arXiv: 2108.11303 [cs.IR].

Appendix A

Appendix

Initial version of the prompt used to generate fake sentences:

'I am aware of the consequences of using synthetic data, I would like for you to generate around 20 samples of sentences of medical healthcare in dutch (health records reports of healthcare related mostly to elderly). The format of the data needs to be as follows: sentences of at least 10 words and no more than 50. The sentences should contain names of diseases, medications, and body parts affected. Do not repeat the same value twice. Give the same probability to any name of disease, medication and body part in your database before selecting them. You should provide for each sentence, 2 lists, one list containing the text separated each word by a comma, and another list with the IOB tag for each word, DO NOT USE ANY OTHER TAGS THAN THE FOLLOWING NO MATTER THE CASE: (O, B-DISEASE, I-DISEASE, B-BODY, I-BODY, B-MEDICATION, I-MEDICATION, B-PERSON, I-PERSON). Do not separate up B and I tags, meaning there should not be O tags in between them. I tags can only be present after their respective B tags (I-DISEASE only after B-DISEASE, not after O for example). The text has to be in Dutch. You dont have to output the entire sentence, just the list with two lists (in python format) where one list has the words separated by commas and in between quotation marks, and followed by another list with the IOB tags mentioned, separated by a comma. The list with the words separated with commas must have the same length as the list with IOB tags. List of root words you could use: rehabiliteren, gediagnosticeerd, voorgeschreven, gedetecteerd, lijden, gebroken, pijn, neemt, vergt, heeft, onhaalbaarheid... For disease, medication, person and body part you can use any logical and popular words, you can omit some entities in any given sentence. For each sentence you must use different root verbs, nouns and a different structure of the sentence. Do NOT use the same verb more than twice. Here is three examples that follows the desired output, output must follow the exact same format to be treated correctly later by a python interpreter: [[["De", "patiënt", "heeft", "last", "van", "artrose", "in", "de", "knieën", "."], ["O", "B-PERSON", "O", "O", "O", "B-DISEASE", "O", "O", "B-BODY", "O"]], [{"Mevrouw", "Jansen", "krijgt", "dagelijks", "bloeddrukverlagende", "medicatie", "gediagnosticeerd", "met", "dementie", "."}], ["B-PERSON", "I-PERSON", "O", "O", "B-MEDICATION", "I-MEDICATION", "O", "O", "B-DISEASE", "O"]], [{"Collumfractuur", "waarvoor", "revalidatie", "in", "Amsterdam", "hospital", "."}, {"Dementie", "gediagnostiseerd", "."}], ["B-DISEASE", "O", "O", "O", "B-LOCATION", "I-LOCATION", "O", "B-DISEASE", "O", "O"]]] IMPORTANT! THE OUTPUT MUST HAVE THE CORRECT STRUCTURE TO BE TREATED IN PYTHON (NO SYNTAX ERRORS and NO MISSING BRACKETS)'

Final prompt used to generate fake sentences, where the values in curly brackets can be changed with every iteration to allow for more varied outputs:

I am aware of the consequences of using synthetic data, I would like for you to generate 10 samples of sentences of medical healthcare in dutch (healthcare records reports of mostly elderly). YOU ARE FORBIDDEN TO USE SIMILAR GENERATED SENTENCES, the more different contexts we have the better. Use different verbs, adjectives nouns and modifiers in each sentence. The sentences should resemble follow-up sessions of a doctor with its patients, in a rather formal environment to keep track of their patients conditions. The goal of this task is to generate sentences with as much diversity in context to later train a model on them. The format of the data needs to be as follows: Sentences of at least {min_length} words and no more than {max_length} words. The sentences must contain names of diseases, medications, and/or body parts affected by the disease or condition that are closely related to the {body_system}. Do not repeat the same value twice. You should provide for each sentence, 2 lists, one list containing the text separated each word in quotation marks by a comma, and another list with the IOB tag for each word: O, B-DISEASE, I-DISEASE, B-BODY, I-BODY, B-MEDICATION, I-MEDICATION, B-PERSON, I-PERSON. DO NOT USE ANY OTHER IOB TAGS THAN THE ONES MENTIONED!. The list containing the words and the length of the list containing the IOB tags must be the SAME LENGTH ALWAYS. Check before generating the output if the lengths of list of words and the length of IOB tags list match, only use them in the output if they are the same length. You are stricktly forbidden from using other IOB tags than the ones listed (O, B-DISEASE, I-DISEASE, B-BODY, I-BODY, B-MEDICATION, I-MEDICATION, B-PERSON, I-PERSON). No exceptions. Do not separate up B and I tags, meaning there should not be O tags in between them. I-tags can only be present after their respective B-tags (I-DISEASE only after B-DISEASE, not after O for example). The text has to be in Dutch. Here are three examples that follow the desired output, output must be exactly the same but with 10 lists inside instead of 3, any deviation, addition or modification to the structure provided is not accepted: {example_sentences}. Follow the structure of a list of lists, where each inner list has two lists inside. Only use this sentences as an example, the generated sentences must have totally different verbs and different syntactic structure. Change the words in quotation marks with real words to form logical sentences. Try incorporating a wide variety of verbs in the generated sentences. Use declarative, interrogative, and imperative sentences. Use dependent clauses relative clauses and prepositional clauses. Use present past and future verb tenses, passive and active. IMPORTANT! MAKE sure the output provided has NO SYNTAX ERRORS, NO MISSING BRACKETS or quotation marks, it should be ready to copy and paste into Python interpreter. Make sure to fullfill all the requirements listed, as one simple mistake is not tolerated. Avoid enumerating the sentences, the output should be a python list object with all of the 10 sentences inside. Make sure the words are correctly tagged and consistent across sentences, in case of a doubt leave as a O-tag, only diseases and medical conditions should be labeled as DISEASE and names of medications and medicines as MEDICATION.'