Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik

# Information extraction from arcticles on the impacts of COVID-19 lockdowns on air quality

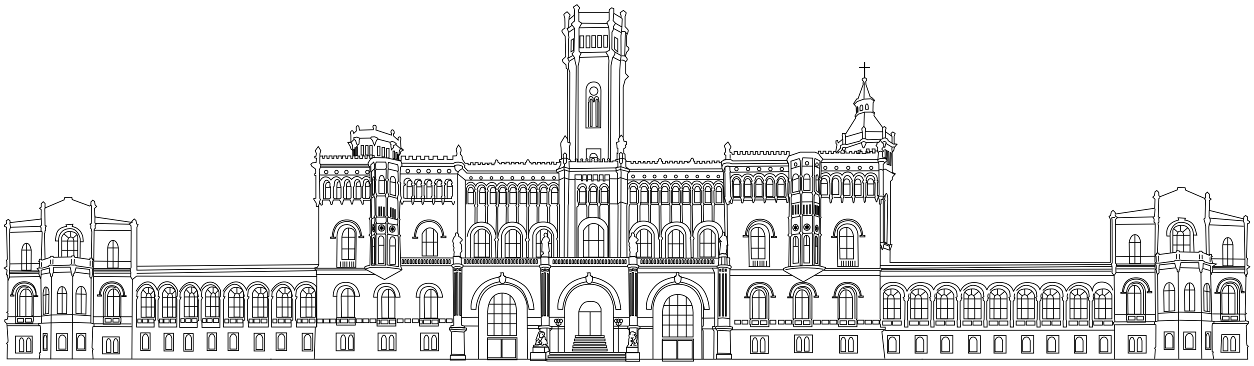
*A thesis submitted in fulfillment of the requirements for the degree of*
**Bachelor of Science in Computer Science**

BY

**Quentin Münch**
Matriculation number: 10031323
E-mail: quentin.muench@stud.uni-hannover.de

First evaluator: Prof. Dr Sören Auer
Second evaluator: Dr Jennifer D'Souza
Supervisor: Dr Markus Stocker

August 23, 2022



# Declaration of Authorship

I, Quentin Münch, declare that this thesis titled, 'Information extraction from articles on the impact of COVID-19 lockdowns on air quality' and the work presented in it are my own. I confirm that:

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

NAME

Signature: ______________________________

Date: ______________________________

"Mistakes are also important to me. I don't cross them out of my life, or memory. And I never blame others for them."

— Geralt of Rivia, book *Blood of Elves* (1994)

# *Acknowledgements*

I would like to express my gratitude to my primary supervisor, Dr Markus Stocker, who guided me throughout this project. Furthermore, I am grateful to Dr Jennifer D'Souza for giving me more profound insights into this topic. Additionally, I wish to show my deep appreciation to Neelam Khan and Dr Georgios Gkatzelis from the research centre Jülich for their excellent cooperation. Last but not least, I would like to thank Prof. Dr Sören Auer, who always advises me on questions.

# *Abstract*

In response to the COVID-19 pandemic, cities worldwide imposed lockdowns to combat the spread of the virus. Governments ordered people to stay at home. Therefore, vehicle and industrial emissions changed drastically. Several researchers studied the impact of such lockdowns on air quality. The research centre Jülich accumulated various articles to gather all information. They manually searched each article to extract the relevant information and created a database containing their findings. Using the gathered data, they developed a website to illustrate their findings to the community. Moreover, they published the data set for other researchers to use freely. However, searching the articles by hand takes significant time and resources. Since the number of articles in the database will continuously increase in the future, developing models for automated extraction of such data can be beneficial. Here, we present a script that utilises a rule-based matching approach to extract pollution data from articles automatically. Around 150 reviewed articles were split into 80% training and 20% test data. We utilised the training data to manually find rules for extracting pollutants, whereas the test data did not influence the creation of patterns. It only serves as a test data set for the evaluation of the model. By feeding the defined rules to the model, it learns to detect various patterns in sentences and how to extract relevant information from them. A significant problem for the automated extraction present tables. They contain a plethora of data. However, extracting information from one does not work appropriately, let alone detecting a table. After the training finishes, the program gets tested using the test data. It achieves a 22% recall and 43% precision value when executed. Compared to manual extraction by experts, this result is significantly worse. Nevertheless, by highlighting relevant text passages, the program offers a great starting point for manual extraction.

*Keywords: Information Extraction, air quality, COVID-19, lockdown*

# Contents

# List of Figures

# List of Tables

# Acronyms

**CO** Carbon monoxide

**COVID-19** Coronavirus disease 2019

**FN** False Negatives

**FP** False Positives

**IE** Information Extraction

**IF** Information Filtering

**IR** Information Retrieval

**NH3** Ammonia

**NMVOCS** Nonmethane volatile organic compound

**NO2** Nitrogen dioxide

**O3** Ozone

**PM** Particulate matter

**SO2** Sulfur dioxide

**TN** True Negatives

**TP** True Positives

# Chapter 1

# Introduction

The outbreak of the 2019 coronavirus disease (COVID-19) greatly impacted the entire world. Due to the virus being highly contagious, people had to minimise social interactions. While the precise measures to reduce the virus spread differed from country to country, almost every country in the world introduced lockdowns. This prohibited people from going outside unless it was essential for their living. Companies had to let the employees work from home, so they did not need to leave the house. Clubs, bars and restaurants had to close down because of government orders and a lack of customers. Considering these points, it is apparent that people were not driving as much as before the pandemic. Instead, they were staying at home, thus considerably reducing their transportation emissions.

Many scientists saw the unique opportunity to research how much of an impact such lockdowns have on air quality. While restrictions put daily life on hold, the streets of major cities were as empty as never before. Researchers published many articles regarding the change in air quality during these lockdowns, e.g. Patel et al. (2020) observed air quality changes in Auckland (New Zealand), Wang et al. (2020) focused on pollution changes in China, and Berman and Ebisu (2020) researched the air pollution in the US. In order to gather all the data in one place, the research centre Jülich collected every article about the change in air quality during lockdowns. However, not all articles were suitable for their project. The papers had to be peer-reviewed, accepted by September 30 2020, and written in English (Gkatzelis et al. 2021). These articles contain a significant amount of data that needs to be processed manually. The research centre Jülich processed each article individually and collected the air quality data. They set up a website which presents the results in various charts. These diagrams provide an excellent way for the community to stay up-to-date regarding the topic.

However, the process of manually having to check each research article is inefficient. Models for automated information extraction of such air quality data points could provide a fitting solution for the aforementioned problem. Since scientists are still researching the connection between lockdowns and air quality, these models would also be helpful for later database updates. Following the principles stated by Andersen et al. (1992), this thesis deals with developing a rule-based information extraction model. For this, the definition of several distinct patterns is crucial. The program then searches the text for these patterns and stores each match. Afterwards, it extracts the information contained in the matches and makes them available for further use. The time spent skimming through the papers will be significantly reduced by automatically extracting the data.

Conversely, the technology is error-prone. It is unlikely to find every single data point in the text. However, at the very least, it could provide a solid baseline for further examination.

The research questions addressed by this thesis are: **RQ1)** Is the proposed approach able to output the same or better results compared to manual extraction? **RQ2)** Is using the proposed approach faster than using only manual extraction? **RQ3)** Is using the proposed approach more accurate than using only manual extraction? **RQ4)** Would using the proposed approach in combination with manual extraction benefit the outcome?

## 1.1 Motivating Example

Let us look at an example to elucidate the importance of the work in this thesis. To do so, we need an article discussing the air quality change during COVID-19 lockdowns. We will use the article from Patel et al. (2020) for this demonstration. Over several pages, this paper states the data gathered by the researchers. However, not all the included information is essential for us. Thus, the reviewer must search through much irrelevant data to find the needed information.

This article contains around 7,000 words without considering figures and tables spreading over 14 pages. According to the colleagues at the research centre Jülich, the manual review takes, on average, 30 minutes per article. Additionally, we must include the time it takes to extract the data from the text and save it to the database. Inspecting these numbers, it becomes clear that there is much room for improvement. Focusing on the extraction script described in this thesis can potentially increase the efficiency of this activity.

Depending on the length and complexity of the article, the script needs at most 10 seconds to finish processing. During this time, it automatically performs various actions. First, it extracts all relevant information it can find and simultaneously writes these results into a CSV file. This file is the core result of the implementation containing every piece of extracted data. The program also marks each extracted sentence so that a reviewer can quickly identify the quality of the extracted result. All of this happens in a matter of seconds.

On top of that, the script can handle batch processing, meaning it can process endless amounts of articles after another. Theoretically, even multiple script instances can run in parallel, further improving the performance. After the program has finished running, a manual review is, however, still necessary. Unlike the manual review without running the program, we now have annotated articles. These annotations enable the reviewer to focus on the relevant passages in the text quickly. We estimate a reduction in reviewing time by at least 30%, resulting in a time saving of 9 minutes per article. As the number of papers regarding air pollution during lockdowns increases, such insignificant seeming time savings can add up. Already processing only five articles amounts to 45 minutes worth of time freed up.

Considering these benefits of using an automated extraction script, the motivation for using such a program becomes evident.

## 1.2 Contributions

This thesis provides a script for the automated extraction of pollutants and their change in value. It can read PDF articles and can search them for relevant information. Although it has limited performance, it nevertheless contributes to the community. Apart from extracting values, it also annotates the processed papers. The program visually highlights the sentence when a text passage matches a previously defined pattern. This procedure simplifies the manual revision because the reviewer can easily recognise the target sentence and decide whether it was a correct extraction or not.

There are currently only 150 articles in the database of the research centre Jülich. Since scientists have published and will publish more research articles regarding the pollutant changes during COVID lockdowns, more data extraction will be needed. By providing the result of this thesis as a resource, we offer a solution to speed up the extraction process. Therefore, we enable the colleagues at the research centre Jülich to work more efficiently.

Additionally, we contribute to the database's maintenance by ensuring the entries are all correct. Because our evaluation compares the extracted values to the manually extracted values in the database, we can quickly notice errors. This way, if there is an oversight in the training data, the script will find the correct values instead. Since machines do not suffer from careless mistakes, wrong entries in the database will be minimal. Thus, offering a cleaner data set in general. A high quality database is vital for any researcher who intends to process the data further.

Overall, this thesis contributes a solid data extraction script while also providing helpful evaluation tools for database maintenance. The developed script plays a part in empowering the community to find accurate pollutant data in one place. Given the CC BY-NC licence, any scientist interested in the data can access it and further process the results. It thereby underlines the importance of collaboration in the community because it offers solutions for other community members' problems. This partnership is even more essential when two different scientific fields work together. Here the field of computer science collaborates with the field of atmospheric science. These two fields complement each other very well, especially regarding information gathering and storing. There is probably even further collaboration possible, where both sides can learn much from each other. This project offers a great first collaboration result which already finds its application in supporting the reviewing of the latest air quality articles.

## 1.3 Structure of the Thesis

In chapter 1, we introduced the context of the study. We have stated the problem and have sketched our approach. The value of such research was underscored by presenting a motivating example and its contributions.

Chapter 2 will outline the fundamental knowledge needed to understand the thesis concepts. It will give an overview of the basics of information extraction and present previously conducted research related to the topic.

In chapter 3, we will explain the different approaches to this project.

Chapter 4 will describe the implementation of the extraction program. It will thoroughly explain every function and module used in this project.

In chapter 5, we will assess the quality of the implementation. For this, we will present the evaluation's realisation and the results.

In chapter 6, we will discuss the previously retrieved evaluation results. We will demonstrate the limitations of the project and mention opportunities for future work building on this thesis.

Finally, chapter 7 sums up the most crucial points in the conclusion.

# Chapter 2

# Background

There are vast amounts of text data on the internet. There is so much of it that no human can ever read and understand everything. That is why we try to use the computer to help us guide through the data. For this, we mainly use the concepts and techniques of information extraction. The following sections introduce the main concepts to better understand the thesis's development.

The first section provides an overview of the essential topics the reader needs to know. It supplies the knowledge to read the thesis. The second section outlines the research that has already been done on the topic of information extraction. It examines three scientific papers and presents their findings.

## 2.1 Essential Topics

The discipline of information extraction contains many different concepts. Since this thesis deals with extracting air pollutant values from scientific articles, some background knowledge is crucial. One needs to comprehend several topics to understand the content represented in this thesis. Therefore, the following sections give a brief overview of the issues discussed. First, we will look at the general term of information extraction. It will provide the baseline details regarding the topic. Because every experiment must have an evaluation to assess the quality of the result, we then describe how to perform such an evaluation. Finally, this section outlines the core air pollutants used in the articles.

### 2.1.1 Information Extraction

There are several strategies for gaining information from texts, the most common being information retrieval (IR), information filtering (IF) and information extraction (IE) (Cowie and Lehnert 1996). Information retrieval concerns itself with all the activities related to the organisation of, processing of, and access to, information of all forms and formats. It can also be seen as a document retrieval system since it is designed to retrieve information about the existence of documents relevant to a user query (Chowdhury 2010). Information filtering, on the other hand, aims to remove irrelevant data from incoming streams of data items (Hanani, Shapira, and Shoval 2001). Information extraction is the automatic extraction of structured information such as entities or relations from unstructured sources (Sarawagi 2008).

In contrast to IR systems, IE systems must extract facts from the documents. We can use the extracted data to construct databases, which are then available for various applications to process the data further (Klügl and Toepfer 2014). Since information often spreads across multiple sentences, natural language processing is fundamental to IE (Grishman 1997). This concept can be a significant challenge because computers process information differently than humans. While human perception can easily create relations between entities, the computer has to process each word bit by bit. Despite being able to save the words in memory, the computer has difficulties relating future references to the stored information.

Before the actual extraction occurs, it is often beneficial to employ various preprocessing techniques. These include splitting sentences into tokens (e.g. words, punctuation marks), recognising the end of sentences, detecting word types, tracing back words to their original form or even correcting small spelling mistakes (Klügl and Toepfer 2014). This preprocessing results in an enhanced performance during the extraction process. The program can concentrate on the main task by doing the hard work in advance. The document is now better structured and prepared for further analysis.

### 2.1.2 Evaluation methods

We use two metrics to measure data retrieval performance from a collection. Let $N_{correct}$ be the correctly extracted data, $N_{extracted}$ all extracted data and $N_{relevant}$ all relevant data. Then (Grishman 1997)

$$\text{precision} = N_{correct}/N_{extracted}$$
$$\text{recall} = N_{correct}/N_{relevant}$$

$N_{correct}$ is also known as true positives (TP). $N_{extracted}$ consists of TP and wrongly extracted data, called false positives (FP). $N_{relevant}$ includes TP and relevant data that the program overlooked, called false negatives (FN). Precision is a general metric to display the ratio of correctly and incorrectly retrieved data. Recall, on the other hand, is the percentage of how much of the relevant data the extraction found. There is also the possibility of combining these two measurements to create an "F-score". Traditionally the F-score is defined as

$$\text{F} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

However, there are also other possible specialisations of the F-score that further emphasise precision or recall (Grishman 1997). A higher score indicates a better performance of the extraction. Generally, we expect a program to reach a precision and recall value of 90%, especially in specialised domains (Klügl and Toepfer 2014).

For a successful evaluation, a sizeable data set is essential. This data splits into two different areas called training and test data. The training data is the most critical and typically gets the most significant portion of the available data. As the name suggests, we use it for training the model. This data includes all previously collected information that has already been verified and found to be correct. Using this information, we can now train the model. By feeding past results to the program, we aim to prime the model such that it identifies future results on its own. The larger the training data set is, the better this process becomes.

We use the remaining available data exclusively for testing. This data is necessary for the evaluation. To assess the quality of the model's extraction, we run the program on the test data. However, we do not use this data to modify the algorithm in any shape or form. We must compare the program's results with the already verified test data. For this, we use the previously mentioned precision and recall values.
A typical ratio between training and test data would be 80/20 or 70/30. Performing such a split provides ample training and testing data and ensures a well-rounded model.

### 2.1.3 Air pollutants

Since the articles for this project focus on the change in air quality during COVID-19 lockdowns, air pollutants play a significant role. Recognising the most critical pollutants is essential so we know what to look for in the text. The primary air pollutants are as follows:

1. nitrogen dioxide (NO2), which is mainly produced by the combustion of fuel
2. particulate matter (PM2.5, PM10) produced by combustion both in industry and in vehicle combustion engines
3. tropospheric ozone (O3), a result of NOx reacting with sunlight, thus increasing with stronger solar radiation and industrial combustion
4. carbon monoxide (CO) primarily produced by combustion of fossil fuels
5. sulfur dioxide (SO2) produced by industrial activity
6. ammonia (NH3) mainly produced in agriculture
7. nonmethane volatile organic compound (NMVOCS) produced by combustion

## 2.2 Related Work

With the internet becoming more and more popular over the past decades, available information online increased rapidly. Consequently, the need for efficient algorithms to find data reliably and quickly grew. Thus, extracting information from texts has manifested itself as a critical research field. Scientists have already conducted much research on this topic proposing different approaches. The following section describes two papers that examined distinct perspectives on information extraction. The first article follows a rule-based approach. The concepts described in the paper form the basis of this thesis. The second paper focuses on a machine learning approach. It is a relatively recent publication from 2020 and describes the current potential of the technology. Lastly, the section will outline an article that concerns itself with the advantages and disadvantages of such approaches.

### 2.2.1 Rule-Based Matching

Andersen et al. (1992) examine a template-driven approach to extract facts from press releases automatically. They developed a program called JASPER (Journalist's Assistant for Preparing Earnings Reports) which scans company press releases from PR Newswire. After identifying relevant articles, it then automatically extracts a predetermined set of information. Afterwards, it transforms the collected data into an individual news story and sends it to a journalist for verification (Andersen et al. 1992). One can gain an edge over competing companies by utilising such a program. The competitors will likely be unable to match the extreme speed of outputting accurate news. Because a journalist has to approve the article beforehand, high quality is also guaranteed.

JASPER functions on a rule-based system. It checks if an information type has not been extracted yet and searches for the appropriate patterns in the sentence. On a match, it then extracts and interprets the information from that sentence. It takes JASPER, on average, about 25 seconds to process a relevant article while maintaining high recall and precision values of around 80% (Andersen et al. 1992).

The principles applied in this thesis are similar to the study conducted by Andersen et al. (1992). A significant difference is that we have to deal with larger amounts of data in general and more intricately presented information.

### 2.2.2 Machine Learning

Due to the increasing amount of research on machine learning algorithms, tracking the current research progress has become difficult. Therefore, Kardas et al. (2020) developed an automatic machine learning pipeline called AxCell to extract results from papers addressing machine learning topics. The goal is to extract tuples containing relevant information such as task, data set, metric name and value (Kardas et al. 2020). They approached this problem by defining subtasks in the AxCell pipeline. First, it needs to identify relevant tables, then it classifies each table cell and finally retrieves the resulting tuples.

For the training and evaluation of the AxCell pipeline, a considerable amount of data is necessary. The primary input is the LaTeX source code of machine learning papers from arXiv.org (Kardas et al. 2020). In total, two data sets are essential for the training of this pipeline. The training begins with the arXiv papers as an unlabelled data set of over 100,000 machine learning papers. Since it consists of unlabelled data, the researchers used it for self-supervised language model learning. The second training data set trains the pipeline on table cell classification. It contains 1,994 tables with annotated table cells to learn. Following the training, they use a validation data set of over 200 annotated papers to adjust the pipeline's performance manually. Finally, they use a test data set of over 2,000 annotated articles as an evaluation tool. When considering the extraction of tuples for entire records (task, data set, metric, score), it achieves good performance with precision and recall between 20-40%. However, if we leave out the score, the performance increases to 45-70%. This difference in efficiency is due to the model having difficulties accurately predicting the score's location in the tables. The remaining tuple (task, data set, metric) can also often be concluded from other results reported in the paper (Kardas et al. 2020).

This research shows that a machine learning algorithm can provide solid results for a specified task when given enough data. It also lays out the difficulties such algorithms have to overcome.

### 2.2.3 Importance of each approach

Industry and academia are not united when choosing which approach to use. While recent academic research focuses mainly on machine learning, the commercial world prefers rule-based systems (Chiticariu, Li, and Reiss 2013). Chiticariu, Li, and Reiss (2013) display the reasons for such a separation and present solutions to reduce this gap. They believe that the disconnect arises from a difference in how the communities measure the costs and benefits of information extraction (Chiticariu, Li, and Reiss 2013). While academics evaluate their models using precision and recall on standard labelled data sets, it is not that simple for industry. Some parts of a process might be more important to resolve than others. Thus, an easy metric may not be enough. When companies decide to change their requirements, rule-based systems are superior. They are easier to understand, alter, and maintain than a machine learning model requiring a complete retrain (Chiticariu, Li, and Reiss 2013).

The paper states that another reason for the lack of rule-based research is the feeling that there is a lack of research problems. However, in the eyes of Chiticariu, Li, and Reiss (2013), this is not the case. Because the reviewed companies with a revenue of more than $100 million almost entirely rely on rule-based systems (Chiticariu, Li, and Reiss 2013), their importance becomes clear. The article proposes the development of a standardised information extraction rule language in combination with a regulated data model. Following this could replicate the success of the SQL language in connecting data management research and practice (Chiticariu, Li, and Reiss 2013).

However, this does not mean scientists should not research machine learning, quite the opposite. Especially when handling large amounts of data, machine learning is an excellent solution since it reduces manual effort. There are plenty of research opportunities available for machine learning algorithms. Nevertheless, it is essential not to disregard existing research. Sometimes, working on improvements for older solutions might present better results instead of getting lost in future technology.

# Chapter 3

# Approach

This chapter first introduces the central problem addressed by this thesis. It outlines what we aim to achieve with this project and states the opportunities for applying information extraction concepts to this problem. Afterwards, the section provides this thesis's approach to tackling the issue and extensively describes the reasoning behind following the proposed solution.

## 3.1 Problem Statement

During the COVID-19 pandemic, many countries declared nationwide lockdowns. Governments ordered people to stay home, significantly reducing traffic and industrial emissions. To research how much air pollution changed during this time, researchers across the globe thoroughly inspected the air quality. They published their results as articles in several scientific journals. To make these results more accessible and visually appealing, the research centre Jülich aims to gather all the information in one place. They manually read each article, searching for information regarding pollution changes. Afterwards, they compress the information and display it on their website. An approach like that has its positives and negatives.

On the one hand, a human can thoroughly search the entire document knowing that one probably did not miss anything. A human can also easily understand and recognise relations in text, which enables the correct allocation of pollutant value pairs. On the other hand, it takes a lot of time and resources to search for information by hand. When the amount of articles increases daily, it is challenging to keep up. On top of that, the concentration decreases steadily after reading through the articles for a while. This fatigue can, in turn, lead to mistakes or not finding all vital information in the text.

Since time is a valuable resource and errors are always undesired, getting the assistance of a computer may improve the situation. Computers are fast, efficient workers, albeit not particularly intelligent regarding understanding human concepts. They need precise instructions to exactly do what we need them to do. This simple knitted system that computers are based on also has its upsides. They are predictable. The computer will precisely do what we tell it if we write a program to execute a specific task. No conscience could influence its actions, nor will there be any lack of concentration. Thus a combination of the automatic extraction by a computer and the manual extraction by a human would provide an excellent trade-off between efficiency and effectiveness.

## 3.2 Proposed Solution

Generally, there are two viable solutions that we can explore. These are rule-based matching and machine learning. In information extraction, there is an important concept to keep in mind. The more data is available for training one's model, the better it will be at the extraction process. For our project, we used the *COVID-19 Air Quality Data Collection* (2021) database version 4, last updated 2022-01-31. It consists of 153 articles containing over 1,000 points of data. These articles have already been manually searched by colleagues at the research centre Jülich. While 1,000 data points might seem like a lot initially, it is not that much in reality. Consequently, the quality of our model will be lower than desired.

### 3.2.1 Machine Learning

First, we will look at machine learning. As the name suggests, the computer should learn on its own how to accomplish a task. On top of that, it should improve its execution each time it tries anew. For this purpose, the program needs a fitness function that assesses the result's quality. The program then aims to maximise said function. Regarding information extraction, the fitness function could be the F-score, i.e. a compound between precision and recall of the training data set.

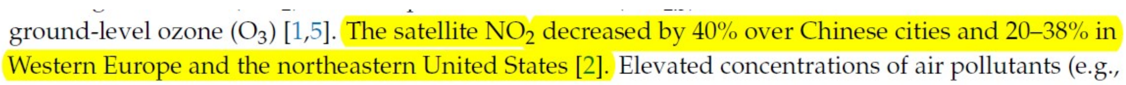
There are two types of machine learning approaches, called supervised and unsupervised learning. For supervised learning, the program gets to use labelled data. These labels help the computer improve accuracy and speed up the training process. Usually, a human has processed the data beforehand and labelled them accordingly. Unsupervised learning, on the other hand, does not use labelled data. It works on its own to determine the structure of the text. Hence, not having any sense of direction, there is a lot more training data needed. Otherwise, the results can end up considerably inaccurate. Generally speaking, this is the main drawback of machine learning. Models need enormous amounts of training data to produce satisfactory results. Comparing our 150 articles to the 100,000 articles used in Kardas et al. (2020) study, only about 150 articles for the training phase are undoubtedly insufficient. For this reason, we can not apply machine learning to our project and instead concentrate on the rule-based matching approach.

### 3.2.2 Rule-based Matching

The second possible approach to an IE problem is rule-based matching, the most basic information extraction form. It uses predefined rules to scan the text for patterns and extract information. For starters, we will have to investigate the articles themselves. Whenever the text contains any sign indicating a change in air quality, we need to analyse the sentence in question. We can then create patterns by analysing the kind of words that precede or follow the pollutant and the associated value. The created patterns now consist of words and numbers representing a sentence containing valuable information regarding air pollutants. Figure 3.1 shows an example sentence. Examining this sentence, we can deduce the first basic pattern. A pollutant (NO2) decreased by a certain amount (40%). We must let our program know that it should look for this pattern in the text. Each time a part of a sentence matches this pattern, the computer finds it and presents the contained information.

Unfortunately, having only this one pattern is not enough. There are various possibilities for describing a change in air pollution. Therefore it is necessary to look through more articles and thus define new patterns while further improving existing ones. In the end, there will be a wide range of patterns so that almost every possible expression is covered.

Although more data equals better results still applies here, it is not essential for the program's success. As long as one knows what one is looking for, one can think of different ways of communicating the information. That way, one can create patterns that one thinks might appear in a document, despite not having seen them. This



ground-level ozone ($O_3$) [1,5]. The satellite $NO_2$ decreased by 40% over Chinese cities and 20–38% in Western Europe and the northeastern United States [2]. Elevated concentrations of air pollutants (e.g.,

Figure 3.1: Example sentence containing a basic pattern

approach enables training beyond actual training data. However, one must carefully evaluate such patterns to avoid too many false positives. Since our training data was limited to 153 articles, we pursued this pattern recognition approach. In order to cover as many different sentences as possible, we created 65 distinct patterns in total. The program aims to look for these arrangements of words in the text.

# Chapter 4

# Implementation

For this project, we used the programming language Python. The entire implementation and the latest state of development are freely available on the GitHub page. Additionally, we published a release of the program's state used for this thesis on Zenodo (Münch 2022).

As the core of this project, we present the extraction script. It contains the code for the extraction of the values, as well as the highlighting of relevant text passages.

This chapter lays out the extraction script's structure and describes how it operates.

## 4.1 Modules

For this project, we need to import several external modules.

***pandas - Python Data Analysis Library* (2022)** Pandas is a valuable tool that provides data structures and analysis tools. It shows its full potential when working with tables. It can automatically convert a CSV file to a table in a pandas data frame. Conversely, it can also write a pandas data frame to a CSV file. Since the training data is in CSV format, using pandas for this project is an obvious choice.

***spaCy - Industrial-strength Natural Language Processing* (2022)** SpaCy is a natural language processing tool that offers various functionalities. It already has pre-trained pipelines for different languages, which enables it to predict linguistic attributes in context. Figure 4.1 shows an example dependency graph for a text passage that matched a pattern. It contains the distinct linguistic features the pipeline predicted. Using these features, we can enhance the ability to define concrete patterns for extracting the desired information.

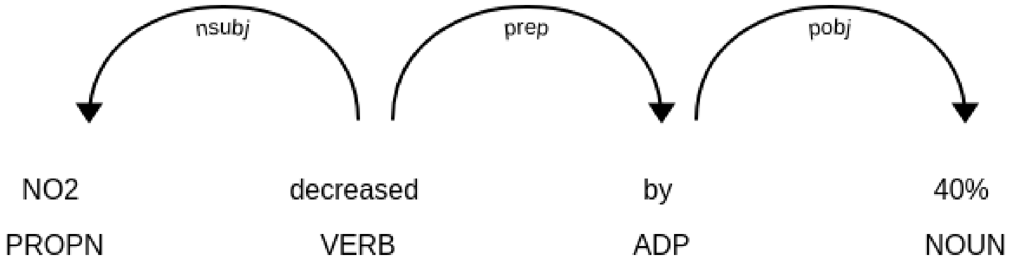
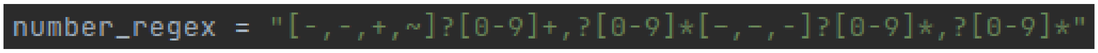


Figure 4.1: Example dependency graph for a matching pattern

***PyMuPDF*** **(2022)** PyMuPDF is a Python toolkit that can view and render different file formats. Since the articles for this project are PDF files, using a PDF reader is inevitable. PyMuPDF offers the best performance and highest quality results compared to other PDF modules for Python. On top of that, it allows highlighting text in a PDF file, which is helpful for manual evaluation.

***re - Regular expression operations*** **(2022)** This library contains regular expression matching operations. Since spaCy's pipeline predictions are sometimes not perfect, falling back to regular expressions is occasionally necessary. In figure 4.2, we can see the primary regular expression used throughout the project. It describes the different ways of expressing number values in the text.



```
number_regex = "[-,-,+,~]?[0-9]+,?[0-9]*[-,–,-]?[0-9]*,?[0-9]*"
```

Figure 4.2: Example regular expression for a number in the text

***tabula-py - PyPI*** **(2022)** Tabula is a Python module for reading tables from PDF files and converting them to a pandas data frame. Because the articles store a lot of the data in tables, having the ability to extract information from them is essential.

***os - Miscellaneous operating system interfaces*** **(2022)** This module provides operating system-dependent functionalities. For our project, we use it to iterate over files in a directory.

***io - Core tools for working with streams*** **(2022)** BytesIO is part of the io module. It enables Python to deal with different types of input and output. We need it to store the parts of the PDF articles that need highlighting.

## 4.2 General Structure

The program is divided into distinct sections. We begin with the definitions of different important variables. Noteworthy are the pollutants, trend words, and regular expressions that form a number. These three elements will appear in almost every declared pattern and are thus vital for the program's success.

There are five types of functions implemented. First comes the primary function *extract_text*. It contains the core logic of the program. Furthermore, there are different functions for extracting pollutants or values from a sentence, fixing pollutants' spellings, and the layout of pages. In the following, we will take a closer look at each of them.

### 4.2.1 extract_text function

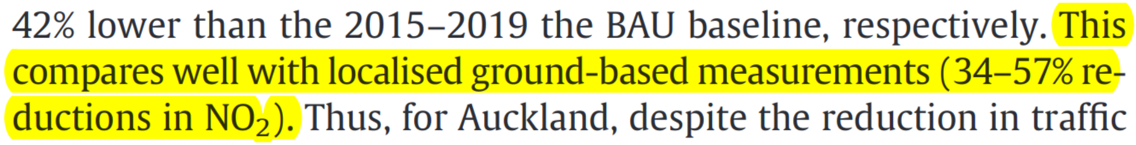
**Algorithm 4.1** General structure of program

```
$nlp \leftarrow spacy.load(''en\_core\_web\_sm'')$          ▷ load nlp pipeline
$matcher \leftarrow Matcher(nlp.vocab)$          ▷ initialise the matcher's vocabulary
for $i = 0$ to $n$ do          ▷ define n patterns
    define pattern $i$
    add pattern to matcher
end for
for each article do
    for each page do
        if no DOI found yet then
            search for DOI on page
        end if
        $matches = matcher(page)$          ▷ run matcher on the current page
    end for
    $total\_data.append(article\_data)$          ▷ append extracted data to the total data
end for
export $total\_data$ to a csv file
```

This function is the heart of the approach, as it contains the declaration of the various patterns and the matcher. Algorithm 4.1 shows the instructions procedure after the function's execution.

It starts with initialising the NLP pipeline and the matcher. Since the articles are written in English, we must load the English language package. Other languages are available, too, if needed. When doing that, however, we would also have to adjust the patterns to the language.

Next are several patterns covering various occurrences of pollutant changes in texts. They then get added to the matcher. These rules each invoke an individual function, where the information gets extracted.
Following the previously defined patterns, the program looks at each article in the directory. Since PyMuPDF works on a page-by-page basis, each article page gets looked at individually. In order to identify the article and provide a better evaluation, the function first searches for the article's DOI. We can accomplish this initial extraction by looking at embedded links on the first page. If that delivers no results, we must search each line for anything resembling a DOI. Important to note is that when a line break splits a word, PyMuPDF does not automatically combine the parts back together. That is why we need the function *squish_page*, which converts all page content to a single line. As shown in Figure 4.3, the word ”reductions” is split by a line break. Without the *squish_page* function, the program would recognise the word as two words, *re-* and *ductions*. This misreading could lead to the matcher not reacting to the pattern, although it technically matches.



42% lower than the 2015–2019 the BAU baseline, respectively. This
compares well with localised ground-based measurements (34–57% re-
ductions in $NO_2$). Thus, for Auckland, despite the reduction in traffic

Figure 4.3: Example word split by line break

Finally, we need to activate the matcher on the current page. That way, it will scan the specified page for any previously defined pattern and call their respective extraction function on a match. It will then add the resulting extracted data to the article's dictionary. After processing every page of the article, the program appends the collected article data to the total data list. This list contains every piece of extracted information and will get exported as *./extracted_data.csv* at the end.

### 4.2.2 squish_page function

This function is vital because PyMuPDF does not merge split words back together after a line break. Thus, we circumvent this issue by converting a page to a single line. In Algorithm 4.2, we can see the structure of the function. Initially, the program retrieves the content of the page. It checks for each line on the page whether a split occurred or not. If there is a split, the ”-” gets removed. Otherwise, we need to

**Algorithm 4.2** Format page to single line

```
$lines \leftarrow page.get\_text().splitlines()$        ▷ get content of the page as lines
$page\_text \leftarrow$ ""        ▷ initialise variable for the text
for line in lines do        ▷ iterate over each line
    $line \leftarrow line.strip()$        ▷ remove redundant spaces at beginning and end
    if line ends with "-" then
        $page\_text = page\_text + line[: -1]$        ▷ remove "-"
    else
        $page\_text = page\_text + line +$ " "        ▷ add space if no split
    end if
end for
return $page\_text$
```

insert a white space to separate the two words. When finished with the page, the function returns the text as a line.

### 4.2.3 get_pollutant function

There are two distinct functions to extracting the pollutants from a sentence. The first is called *get_pollutant*, which returns the first pollutant it finds in the text. The other function is called *get_all_pollutants*. Every pollutant in the sentence gets returned as a list, not just the first one. The matcher needs these functions when a match occurs. There it calls the respective function where the program then relates the pollutants to their values.

### 4.2.4 get_values function

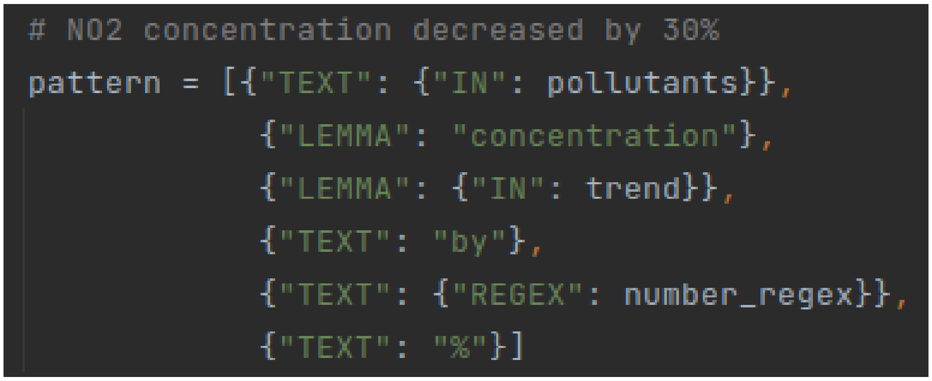
There are four different functions to extract values from a sentence. Similar to the pollutants, there is a *get_values* and a *get_all_values* function. This time, *get_values* extracts all numbers followed by a percentage sign and returns them as a list. *get_all_values*, however, extracts every number that occurs in the sentence, regardless of preceding and following characters. There is another function called *get_plus_minus_values* which specialises in numbers preceded by ±. The fourth function, *get_no_trend_values*, extracts values that already have a minus or plus sign in their name. Contrary to the previous three functions, this one only gets called once there is no trend word in the sentence. Therefore it needs to check the preceding character for the trend.

### 4.2.5 fix_pollutant function

This function, on the one hand, improves the structure and readability of the output and, on the other hand, simplifies the evaluation process. Since there are many different possibilities of spelling the pollutant, we need the *fix_pollutant* function. It checks the spelling of the current pollutant and returns it correctly spelt.

## 4.3 Patterns

Patterns are the foundation of this program. They provide the rules for what the matcher should look for in the document. Generally, one can say that the more patterns are defined, the better the result will be. SpaCy offers a rich tool set for describing patterns. Figure 4.4 shows a simple pattern definition. In this case, the text passage has to lead with a pollutant. The program stores every possible pollutant in the *pollutants* variable as a list. The pattern checks if the list contains the current word. If yes, it moves on to the next word. Using spaCy's linguistic features, we can look for the base form of a word. We can do this via the *lemma* keyword. It is also possible to make a word optional. Since not every pattern has to include "concentration", it could be a good idea to apply that here. Following is a trend word, e.g. *decrease* or *reduce*. The program also stores these in a predefined list. After the preposition "by", a number follows. We implement this using the regular expression shown earlier in figure 4.2. Finally, the pattern ends on a percentage sign.



```
# NO2 concentration decreased by 30%
pattern = [{"TEXT": {"IN": pollutants}},
           {"LEMMA": "concentration"},
           {"LEMMA": {"IN": trend}},
           {"TEXT": "by"},
           {"TEXT": {"REGEX": number_regex}},
           {"TEXT": "%"}]
```

Figure 4.4: Example pattern definition using spaCy

Using this pattern as a baseline, we can continuously define more patterns to cover many distinct constellations of words. In the end, we formed 65 patterns to guide the data extraction.

## 4.4 Matcher

Now that we defined the patterns, the matcher becomes effective. It takes the previously stated rules as input and starts searching the text. As soon as a sentence matches a known pattern, it calls the corresponding matcher function. There are 14 distinct matcher functions in total.

**Algorithm 4.3** Basic matcher function

```
$span \leftarrow doc[start : end]$                 ▷ get the excerpt that matched the pattern
$pol \leftarrow get\_pollutant(span)$                                    ▷ get the pollutant
$values \leftarrow get\_values(span)$                                       ▷ get the values
if pol not in $article\_data$ then              ▷ check if pollutant already has entries
    $article\_data[pol] = values$                  ▷ if no → add data to dictionary
else
    for value in values do
        if value not in $article\_data[pol]$ then
            $article\_data[pol].append(value)$                 ▷ if yes → append data
        end if
    end for
end if
$highlight\_match(span.sent.text)$                        ▷ highlight match in the text
```

Algorithm 4.3 shows the basic structure of a matcher function. Initially, we need to retrieve the text matching the pattern. We do this by getting the start and end index of the match and applying that to the document. With the help of the previously defined *get_pollutant(span)* and *get_values(span)* functions, we extract the pollutant and the values from the text. To store the extracted data, we utilise the associated *article_data* dictionary. It is essential to check whether the pollutant already has entries in the dictionary. If that is the case, we need to examine every value not to produce duplicates. Finally, we highlight the sentence containing the pattern.

There are various matcher functions that all have unique alterations. For some sentences, there might be no pollutant present in the excerpt. In this case, the *get_pollutant(span)* function would return an empty string. Therefore we implement the *no_pollutant_match* function. It calls the *get_all_pollutants(span)* function on the previous sentence instead of the current one and takes the last pollutant it found.
Another important matcher function is the *multi_matcher*. Sometimes a sentence contains a sequence of pollutants followed by a sequence of values. While the primary

matcher function can only handle one pollutant, the *multi_matcher* handles multiple at once. We do this by calling *get_all_pollutants(span)* and relating each pollutant to precisely one value. For this, checking that the number of retrieved pollutants equals the retrieved values is crucial. Otherwise, a mismatch can likely occur.

There are other matcher functions, e.g. when there is no trend word in the sentence, but most are combinations of the previously explained functions and work similarly. Additionally, there are matcher functions for extracting data from tables. Since the layout of the tables can differ, the implementation is challenging. Generally, we try to find the column or row that contains the values and relate that to a pollutant. Likewise, there is also a function that searches the captions of tables and highlights them if they seem interesting.

# Chapter 5

# Experimental Evaluation

Using the evaluation, we can assess the quality of the implementation. Then we can answer our research questions and discuss whether the project was successful or not. This chapter outlines how we evaluated our proposed extraction script and presents the results.

## 5.1 Evaluation Implementation

The evaluation is essential for the scientific process because it rates how well the proposed solution to the research problem performs. Here, we present a program that automatically compares the result of our implementation with the manually reviewed data. In the end, this script outputs precision and recall values of the automated extraction in addition to the F-score.

This section covers all the modules used in the evaluation program and lays out the general structure it follows.

### 5.1.1 Modules

Similar to the implementation, we also need to import some external modules. The only new in this case is the sys module. Additionally, we need the pandas module again. It enables us to access the extracted and the training data CSV files and convert them to pandas data frames. These data frames provide quick and easy access to all available information.

***sys - System specific parameters and functions* (2022)** Among other things, the sys module allows us to retrieve command line parameters. These parameters are important because they provide us with the tools to conduct different evaluations using different parameters dynamically.

### 5.1.2 General Structure

Three different evaluations are available for the primary function - one for test data, another for training data and the last for training and test data together. Additionally, these evaluations use several helper functions to assist with the assessment.

#### get_needed_pollutants function

This function iterates over every possible pollutant. The function appends the pollutant to a list if it exists in the extracted data. In the end, it returns the final list containing the pollutants in the extracted data. We need that to quickly and accurately access the columns of the extracted and training data.

#### get_total_data function

This function counts the entries in the training data. It checks every cell in the table and increases the count whenever a number is inside. The total amount of entries is essential for the final recall calculation.

#### convert_to_list function

The extracted data is saved as a list in a pandas data frame during extraction. When reading in the file for the evaluation, pandas does not recognise the content as a list. It presumes the file contains strings. Since working with strings is always undesirable, we utilise the *convert_to_list* function. It converts every cell in the table to a list, so the data is easily accessible.

#### get_correctly_extracted function

This function is where the actual evaluation happens. Here we compare the extracted data to the training data. A precise comparison is possible by matching the DOIs in both data sets. Whenever a value in the extracted data matches the corresponding values in the training data, it increases a counter. After checking each extracted value, the function returns this counter.

### calculate_score function

The final helper function calculates the precision and recall values using the previously retrieved information. It prints both values to the console, thus enabling a clear depiction of the evaluation result.

### main function

The primary function combines all the helper functions. It has one input parameter, which is the desired data. This parameter can either be the test data, the training data or all available data. It then converts the training and extracted data to a pandas data frame. Additionally, it renames the training data columns for easier referencing. Subsequently, the function calls all helper functions in the order mentioned above. After calling the last function, the evaluation process finishes.

## 5.2 Data Preparation

Before starting with the model training, i.e. the manual creation of extraction rules, we have to separate the available training data. We decided to do an 80/20 split. Therefore we deducted 20% of the articles and used it solely as test data. Since the total number of articles is around 150, we selected 30 for testing. These articles do not contribute to the training phase, so we must keep them separate. The file containing the training data also splits into two files. The first one contains only test data, and the second one only training data. This split is crucial so that we can evaluate each data set individually. Additionally, we keep the initial training data file as a union of the other two. The data preparation is complete now that we have separated the articles and assembled the three files.

## 5.3 Evaluation Setup

To evaluate the script, we first need an output. For that, we need to know which evaluation we want to perform. Generally, the test data is the primary evaluation focus. As a consequence, it is essential that we only include the articles marked as test data in the extraction process. Once that is verified, it is time to run the extraction. By executing the command `python Extraction.py`, the computer searches the articles and outputs the results to the *extracted_data.csv* file. The evaluation can commence now that the extracted data and the training data are present.

By comparing the set and actual values, we can assess the quality of the extraction. Since the evaluation focuses on the test data, we should run the command `python Evaluation.py test`. The script compares each entry of the extracted data with the test data and outputs the result to the console.
It is also possible to evaluate the entire data or only the training data. To do this, we need only to use the desired articles when running the extraction. Then we can run `python Evaluation.py training` or `python Evaluation.py` to evaluate the training or all data. It is now possible to answer the research questions using the gathered results.

## 5.4 Evaluation Results

| | only test data | only training data | all data |
|---|---|---|---|
| recall | 22.36% | 23.83% | 23.46% |
| precision | 43.32% | 44.08% | 43.90% |
| F-score | 29.50% | 30.94% | 30.58% |

Table 5.1: Evaluation results

Table 5.1 shows the results of the evaluation script. The test data compare well with the training data with a recall value of 22.36% and a precision of 43.32%. The F-score represents the harmonic mean of these two values and thus equates to 29.50%. While the training data delivers slightly better outcomes (23.83% and 44.08% for recall and precision, and 30.94% for the F-score), the test data is still within a margin of two per cent. When we consider all the available data, the numbers returned by the evaluation of 23.46%, 43.90% and 30.58% for recall, precision and F-score, respectively, are similar.

The evaluation also reveals a few oversights made during the manual reviewing process, namely

- getting two pollutants mixed up (5 times)
- registering the wrong number in the database (5 times)
- saving rounded values instead of precise values (3 times)
- registering the number in the wrong column (once)

Fixing these errors in the data set would, in turn, increase the evaluation's total precision and recall values. Table 5.2 presents the results with a corrected database. For the test data, we can observe a slight increase of 0.3% for recall and 0.6% for precision. The changes in the training data are more noticeable, with an improvement of 0.63% and 1.34% for recall and precision.

| | only test data | only training data | all data |
|---|---|---|---|
| recall | 22.66% | 24.46% | 24.01% |
| precision | 43.92% | 45.42% | 45.05% |
| F-score | 29.90% | 31.80% | 31.32% |

Table 5.2: Fixed evaluation results

Finally, we notice a trend for the highlighted text passages when looking at the annotated articles. Most notably, we can locate the highlighted sections in the abstract or the results chapter. Although the conclusion also frequently contains annotations, these are usually not complete. The results section generally fully incorporates the relevant data.

# Chapter 6

# Discussion

In this chapter, we analyse the result. We need to find out why we received such an outcome, what this means for our project and how we can build on that. The first section interprets the results and, by doing that, answers the research questions. Following that, we discuss the limitations of the proposed solution by finding out what the script can and can not do. The final section proposes opportunities for future work. These include improvements to the extraction, as well as chances for further processing of the data.

## 6.1 Interpretation of the results

The first striking thing to note is that even though we trained the model using the training data, the precision and recall values for that specific data set are comparatively low. This observation also answers RQ1. The automatically extracted results are worse than the manually extracted ones. This result is not surprising since humans can find text relations much easier than machines. The script finds only about one out of four words with a recall value of less than 25%. In comparison, this is far away from the near 100% the manual extraction would reach.

However, this does not mean that the manual extraction is flawless. The evaluation also outlines various mistakes during the manual reviewing process. However, as stated in the contributions section, having a high-quality database is vital when the data is part of further research. Since the research centre Jülich offers this database to the community, anyone can work with the data and expects that it is not faulty. Thus, we should eradicate any error that occurs.

While achieving high precision is always the ambition, the focus for this project lies more within the recall. It is desirable to get more results for the IE task at hand because, unless we can achieve a 100% precision and recall value, manual supervision of the data is still needed. When manually searching the text, the highlighted parts quickly grab the reviewer's attention. This feature can substantially speed up the process of finding relevant text passages. Knowing that the program highlights the majority of essential sections, the analyst can concentrate on those sections. Despite taking a few minutes to execute, the automatic extraction undeniably finishes faster than the manual extraction. Moreover, we argue that using the script can even increase the speed of the manual extraction. This result, therefore, answers RQ2.

Regarding RQ3, the program does not have great accuracy, with a precision of less than 50%. Albeit being of lesser concern for this project, it is evident that manual extraction yields more accurate results.

To answer RQ4, using the script has no significant downside. It might take a few minutes to set up and execute, but after all, it provides highlighted articles in combination with an easily editable extracted file. However, it is essential to remember that there will likely be values the program did not catch. Thus, we still need some manual effort for the extraction process.

Ultimately, the test data does not reach the desired 80 to 90% precision and recall. Considering there are several problems with the evaluation, as well as difficulties in the extraction, the result is nevertheless still respectable.

## 6.2 Limitations

Looking at the presented results of the evaluation, it becomes clear that there must be several challenges the approach has to overcome. Both the extraction as well as the evaluation have issues. The following section elaborates on these problems and summarises the script's limitations. First, we will look at the extraction program and discuss the flaws of the implementation. Afterwards, we will investigate the limits of the evaluation script.

### 6.2.1 Extraction

As stated previously, the training data only reaches a recall value of around 24%. In theory, this should not be the case since the whole point of the training data is to train the model. If only about a quarter of the data is usable for training, the quality of the model drastically reduces. There are several valid reasons for such a poor performance.

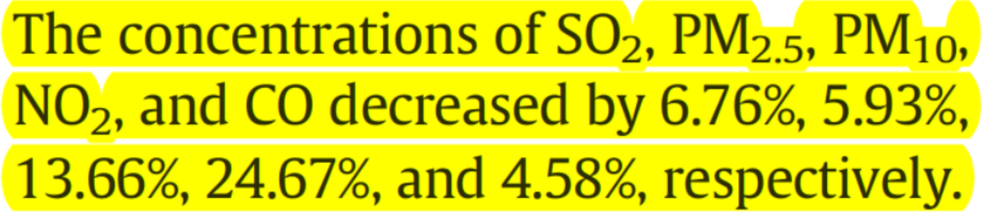
First and foremost, working with PDF files is always a challenge. They are usually not meant to be processed again. Also, there is no general layout for PDF files, so almost every file has unique features. Since most articles use multi-column text, it amplifies this problem even further. Even using conversion tools such as *GROBID* (2022) is not a viable alternative because there will always be some information loss. PyMuPDF is rather good at accessing the information in PDF files. However, a problem occurs since it processes documents on a page-by-page basis. If a pattern stretches over two pages, the matcher will never find it. Furthermore, cramming the entire document into a single line is not an option because the page number is necessary for highlighting.

Another problem with PyMuPDF is the lack of table recognition. A solution for this is using tabula. Tabula can commonly recognise tables, but its automatic detection still has issues. The fact that each table has a different layout further intensifies this issue. There is no general extraction rule that we can apply here. This uncertainty presents a substantial problem. Tables are rich data sources, and if the retrieval of such information is that inconsistent, much potential is lost. Similar to tables, some graphics include information. Graphics are even harder to process since one would need a tool for analysing images.

Another difficulty is that sometimes characters can be interpreted differently by PyMuPDF. Especially the *"-"* sign has many different characters used interchangeably, signifying the same. This confusion can lead to patterns not matching because it was another occurrence of the same character.

Finally, some general difficulties are hard to fix. A good balance is crucial when defining patterns. On the one hand, the pattern must be simple to match more than one sentence. On the other hand, it must also be precise not to produce many false positives. Additionally, patterns might overlap, which could lead to incorrect results. Figure 6.1 shows an example of such an overlap. Two distinct patterns activate in this sentence. The first one matches *PM10, NO2, and CO decreased by 6.76%, 5.93%, 13.66%.* This match, however, yields invalid results since CO did not decrease by 13.66% but rather 4.58%. The second match finds all five pollutant number pairs and would be the only correct choice.

One could try to fix this issue by only matching when *"and"* precedes the last number. However, there are several other occurrences where the extraction would be correct when a comma precedes the last number. Thus, we end up in a situation that is difficult to fix.



The concentrations of $SO_2$, $PM_{2.5}$, $PM_{10}$, $NO_2$, and CO decreased by 6.76%, 5.93%, 13.66%, 24.67%, and 4.58%, respectively.

Figure 6.1: Example overlap of patterns

### 6.2.2 Evaluation

Not only the extraction but also the evaluation faces challenges. The primary reason is the application of different methods of manually extracting and saving the data to the training data file, e.g. where the program extracted several (correct) values, but the training data only contained the averaged values for the comparison. Likewise, we also find manually rounded numbers in the training data.

Another problem for the evaluation is human error. Humans are prone to commit mistakes. Pollutants get mixed up, characters get interchanged, or cells get confused. By utilising the extraction script, we can minimise these errors. Therefore, the accuracy discussed in RQ3 would increase when combining automatic and manual extraction.

The final difficulty for the evaluation is that a DOI is required. If there is no DOI in the document or the extraction delivered a wrong DOI, we can not automatically evaluate the extracted information for that document. There are seven articles that either have no DOI or whose DOI fails to get correctly extracted.

When taking all the problems mentioned above into account, the evaluation should, in theory, yield slightly better results. However, the improvement would presumably be less than a few percentage points.

## 6.3 Future Work

There are several opportunities for future work. Since working with PDF files is sometimes problematic, a different approach could be interesting. Because most articles are open access, HTML scraping might provide another solution. Following the approach of Kardas et al. (2020), working with the LaTeX source code of the articles may present another possibility. This method would also improve the recognition of tables because when using LaTeX, the tables are clearly defined. However, getting the source code of the articles could be challenging since it is mostly unavailable, and not everyone writes their papers using LaTeX.

Currently, the program only extracts percentage values. Adding the detection of absolute values should be another future task. On top of that, additional improvements to the recognition of values are perpetually helpful.

In the future, more articles should be available regarding the change in air pollution during lockdowns. Therefore the training can be improved even further. Moreover, it might be possible to apply machine learning at some point. Using these new articles, one can further conduct a new study examining the helpfulness of the program. It would be interesting to see how much the manual extraction benefits from running the extraction script in advance.

Having extracted all relevant information correctly, the opportunity for further research based on the pollution data presents itself, i.e. one could integrate the available information into other infrastructures. These integrations could include visualising the data differently or deeper analysing connections between the pollutants and their values.

# Chapter 7

# Conclusions

The COVID-19 pandemic offered scientists a unique research opportunity to study the impact of lockdowns on air quality. They published various research articles covering the situation in cities around the world. To gather all available information in one place, the research centre Jülich manually extracted the information regarding changes in pollution from every article. However, doing this by hand is inefficient. Since the number of papers concerning the air quality changes during lockdowns still increases, automatic extraction of relevant information could improve the process. Therefore, this thesis presents a rule-based matching approach in the shape of a Python script to automate this procedure.

Extracting information from the given COVID-19 lockdown articles proves to be complicated. In order to succeed, we need to overcome many challenges, from accurately working with PDF files to defining fitting patterns to correctly recognising tables and extracting their data. Those and a lot more problems arise. Eventually, the script created for the automated extraction does not provide ideal results. In fact, with not even a recall value of 25%, it can not stand on its own.

However, this is not to say that the program has no use. Because it highlights the sentence whenever it finds a match, it can facilitate the manual reviewing process. By directing the reviewer's attention to the marked sections, the need for extensive searching shrinks. Thus, enabling an overall faster extraction process. Furthermore, the script's output gets stored in a CSV file. This file can easily be accessed and edited if there are changes necessary.

Moreover, the project revealed that we could commonly find the most critical information in the abstract or the results section. Following this, the manual reviewer should also shift its focus to these two chapters.

The main limitation of the program is the extraction of information from tables. The project will gain much value if the program can automatically recognise tables and then correctly transform them into machine-accessible data. Unfortunately, the used Python module tabula does not deliver excellent results. However, tabula also has a standalone program where one can manually select tables in PDF files. The manual selection works very well, though it is not helpful for automated processes such as the work presented in this project. The poor performance of the table recognition also has to do with the countless variations of table layouts. Each table has a unique layout, which in turn hinders not only the detection but also the extraction.
Another limitation is the comparatively small amount of training data. With only 150 articles to work with, the model can not unleash its full potential. There is not enough information from which the model can learn. This situation also forces us to use a rule-based matching approach instead of machine learning. Having alternative options when carrying out a project is always desirable, which is not the case this time.

In the future, many research opportunities will be available to build on this thesis. The possibilities range from little things such as adding new patterns and improving the existing ones to entirely changing the approach to the problem. However, new current data is necessary to achieve that. Since scientists are still actively researching the effects of lockdowns on air quality, the supply of articles regarding this topic will not fade soon.

# Bibliography

Andersen, Peggy M. et al. (Mar. 1992). "Automatic Extraction of Facts from Press Releases to Generate News Stories". In: *Third Conference on Applied Natural Language Processing.* Trento, Italy: Association for Computational Linguistics, pp. 170–177. DOI: 10.3115/974499.974531. URL: https://aclanthology.org/A92-1024.

Berman, Jesse D. and Keita Ebisu (2020). "Changes in U.S. air pollution during the COVID-19 pandemic". In: *Science of The Total Environment* 739, p. 139864. ISSN: 0048-9697. DOI: https://doi.org/10.1016/j.scitotenv.2020.139864. URL: https://www.sciencedirect.com/science/article/pii/S0048969720333842.

Chiticariu, Laura, Yunyao Li, and Frederick Reiss (2013). "Rule-based information extraction is dead! long live rule-based information extraction systems!" In: *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 827–832.

Chowdhury, Gobinda G (2010). *Introduction to modern information retrieval.* Facet publishing.

*COVID-19 Air Quality Data Collection* (2021). [Online; accessed 22-August-2022]. URL: https://covid-aqs.fz-juelich.de.

Cowie, Jim and Wendy Lehnert (Jan. 1996). "Information Extraction". In: *Commun. ACM* 39.1, pp. 80–91. ISSN: 0001-0782. DOI: 10.1145/234173.234209. URL: https://doi.org/10.1145/234173.234209.

Gkatzelis, Georgios I. et al. (Apr. 2021). "The global impacts of COVID-19 lockdowns on urban air pollution: A critical review and recommendations". In: *Elementa: Science of the Anthropocene* 9.1. 00176. ISSN: 2325-1026. DOI: 10.1525/elementa.2021.00176. eprint: https://online.ucpress.edu/elementa/article-pdf/9/1/00176/458795/elementa.2021.00176.pdf. URL: https://doi.org/10.1525/elementa.2021.00176.

Grishman, Ralph (1997). "Information extraction: Techniques and challenges". In: *Information Extraction A Multidisciplinary Approach to an Emerging Information Technology.* Ed. by Maria Teresa Pazienza. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 10–27. ISBN: 978-3-540-69548-6.

*GROBID* (2022). [Online; accessed 5-August-2022]. URL: https://grobid.readthedocs.io/en/latest/Introduction/.

Hanani, Uri, Bracha Shapira, and Peretz Shoval (2001). "Information filtering: Overview of issues, research and systems". In: *User modeling and user-adapted interaction* 11.3, pp. 203–259.

*io - Core tools for working with streams* (2022). [Online; accessed 4-August-2022]. URL: https://docs.python.org/3/library/io.html.

Kardas, Marcin et al. (Nov. 2020). "AxCell: Automatic Extraction of Results from Machine Learning Papers". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP).* Online: Association for Computational Linguistics, pp. 8580–8594. DOI:

10.18653/v1/2020.emnlp-main.692. URL: https://aclanthology.org/2020.emnlp-main.692.

Klügl, Peter and Martin Toepfer (2014). “Informationsextraktion”. In: ISSN: 0170-6012, 1432-122X. DOI: 10.1007/s00287-014-0776-6. URL: https://www.tib.eu/de/suchen/id/springer%3Adoi%7E10.1007%252Fs00287-014-0776-6.

Münch, Quentin (Aug. 2022). *QuentinBot/Bachelor: v1.0*. Version v1.0. DOI: 10.5281/zenodo.7012890. URL: https://doi.org/10.5281/zenodo.7012890.

*os - Miscellaneous operating system interfaces* (2022). [Online; accessed 4-August-2022]. URL: https://docs.python.org/3/library/os.html.

*pandas - Python Data Analysis Library* (2022). [Online; accessed 4-August-2022]. URL: https://pandas.pydata.org/.

Patel, Hamesh et al. (2020). “Implications for air quality management of changes in air quality during lockdown in Auckland (New Zealand) in response to the 2020 SARS-CoV-2 epidemic”. In: *Science of The Total Environment* 746, p. 141129. ISSN: 0048-9697. DOI: https://doi.org/10.1016/j.scitotenv.2020.141129. URL: https://www.sciencedirect.com/science/article/pii/S0048969720346581.

*PyMuPDF* (2022). [Online; accessed 4-August-2022]. URL: https://pymupdf.readthedocs.io/en/latest/#.

*re - Regular expression operations* (2022). [Online; accessed 4-August-2022]. URL: https://docs.python.org/3/library/re.html#.

Sarawagi, Sunita (2008). *Information extraction*. Now Publishers Inc.

*spaCy - Industrial-strength Natural Language Processing* (2022). [Online; accessed 4-August-2022]. URL: https://spacy.io/.

*sys - System specific parameters and functions* (2022). [Online; accessed 4-August-2022]. URL: https://docs.python.org/3/library/sys.html.

*tabula-py - PyPI* (2022). [Online; accessed 4-August-2022]. URL: https://pypi.org/project/tabula-py/.

Wang, Pengfei et al. (2020). “Severe air pollution events not avoided by reduced anthropogenic activities during COVID-19 outbreak”. In: *Resources, Conservation and Recycling* 158, p. 104814. ISSN: 0921-3449. DOI: https://doi.org/10.1016/j.resconrec.2020.104814. URL: https://www.sciencedirect.com/science/article/pii/S092134492030135X.