Modelling and Evaluation of Serial and Parallel Density-Based Clustering for Acute Respiratory Distress Syndrome

Chadi Barakat

Faculty of Industrial Engineering, Mechanical Engineering and Computer Science
University of Iceland
2019

MODELLING AND EVALUATION OF SERIAL AND PARALLEL DENSITY-BASED CLUSTERING FOR ACUTE RESPIRATORY DISTRESS SYNDROME

Chadi Barakat

60 ECTS thesis submitted in partial fulfillment of a
Magister Scientiarum degree in Bioengineering

M.Sc. committee
Dr. Sigurður Brynjólfsson
Dr. Morris Riedel

Examiner
Dr. Helmut Wolfram Neukirchen

Faculty of Industrial Engineering, Mechanical Engineering and Computer Science
School of Engineering and Natural Sciences
University of Iceland
Reykjavik, May 2019

Modelling and Evaluation of Serial and Parallel Density-Based Clustering for Acute Respiratory Distress Syndrome
Clustering ARDS using Serial and Parallel DBSCAN
60 ECTS thesis submitted in partial fulfillment of a M.Sc. degree in Bioengineering

Faculty of Industrial Engineering, Mechanical Engineering and Computer Science
School of Engineering and Natural Sciences
University of Iceland
Tæknigarður - Dunhagi 5
107, Reykjavik
Iceland

Telephone: 525 4700

Bibliographic information:
Chadi Barakat, 2019, Modelling and Evaluation of Serial and Parallel Density-Based Clustering for Acute Respiratory Distress Syndrome, M.Sc. thesis, Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland.

Reykjavik, Iceland, May 2019

To my parents and my brothers for keeping me grounded,
to my professors for pushing me forward,
and to my friends for reminding me to have fun along the way.

Abstract

Acute Respiratory Distress Syndrome (ARDS) is a highly heterogeneous condition that affects a relatively low number of mechanically-ventilated intensive care patients, though its mortality rate is quite high, hence the need to develop early diagnosis methods. In this thesis we developed, modelled, and evaluated an approach to extract patient data and evaluate it, perform pre-processing steps, and cluster it using the density-based DBSCAN clustering algorithm. At the same time we made use of available High Performance Computing (HPC) technology in order to parallelise the data manipulation processes. Our implementation showed a significant speed-up in the performance of the highly compute- and data-intensive tasks when using the parallel computing infrastructure as opposed to running them in serial. Additionally, this process was used to assist in selecting the most effective values for the clustering parameters. These results set the path for future implementations of parallel computing in the large scale data extraction and analysis steps necessary for real-time diagnosis in a hospital environment. Furthermore, the pre-processing methods developed in this thesis can easily be adapted to other conditions in the ICU through the use of a HPC-ready Jupyter Notebook, which ultimately makes parallel computing accessible to clinicians without the need for expertise in parallel computing script development.

Útdráttur

Brátt andnauðarheilkenni er fjölþætt ástand sem hefur áhrif á tiltolulega fáa sjúklinga í öndunarvél á gjörgæslu en hefur hins vegar háa dánartíðni, því er nauðsyn að þróa snemmskimunaraðferðir. Í þessu verkefni var þróað og sett upp líkan, aðferð til að sækja gögn sjúklinga ákvörðuð, forvinnsluskref framkvæmd og gögn þyrpuð eftir þéttleika með því að nota reikniritið DBSCAN. Einnig var notuð háafkasta útreikningatækni (e. High-Perfomance Computing (HPC)) til að samhliða reikninga. Útfærslun sýndi marktæka hraðaaukningu í afköstum útreikninga og meðferð gagna þegar notaðir eru samhliða útreikningar miðað við raðbundnar keyrslur. Þetta ferli var notað til að velja á gildi fyrir þyrpingar. Þessar niðursör leggja grunn að rauntíma samhliða vinnslu á gagnagreiningu í umhverfi sem er á sjúkrahúsum. Ennfremur er hægt að aðlaga forvinnsluaðferðirnar í þessari rannsókn fyrir aðrar aðstæður innan gjörgæsludeilda með því að nota hugbúnaðinn Jupyter Notebook, sem gerir klínískum sérfræðingum kleift að nýta sér samhliða gagnagreiningu án þess að hafa þá sérfræðiþekkingu á samhliða gagnagreiningar kóða.

Contents

List of Figures ix

List of Tables xi

Abbreviations xv

Acknowledgments xvii

1. Introduction 1
- 1.1. Motivation . . . 1
- 1.2. Research Questions . . . 2
- 1.3. Thesis Structure . . . 2

2. Foundations 5
- 2.1. The MIMIC-III Database . . . 5
- 2.2. Acute Respiratory Distress Syndrome . . . 6
- 2.3. Unsupervised Learning and the DBSCAN Algorithm . . . 7
  - 2.3.1. Unsupervised Learning . . . 7
  - 2.3.2. Principal Component Analysis . . . 8
  - 2.3.3. Clustering using DBSCAN . . . 9
- 2.4. Parallel Computing and Clustering . . . 10
  - 2.4.1. Parallel Computing and High Memory . . . 10
  - 2.4.2. HPDBSCAN and Further Work . . . 11

3. Related Work 13
- 3.1. Clustering and Feature Analysis in the MIMIC Database . . . 13
- 3.2. Use of HPC Technology in ARDS Data Analysis and Clustering . . . 14

4. Data Analysis and Modelling Process 17
- 4.1. Business Understanding . . . 18
  - 4.1.1. Business Objectives . . . 18
  - 4.1.2. Situation Assessment . . . 18
  - 4.1.3. Goal Description . . . 20
  - 4.1.4. Project Plan . . . 21
- 4.2. Data Understanding . . . 22
  - 4.2.1. Collecting Initial Data . . . 22

4.2.2. Data Description . . . 22
4.2.3. Data Exploration . . . 23
4.2.4. Verifying Data Quality . . . 26
4.3. Data Preparation . . . 26
4.3.1. Dataset Description . . . 26
4.3.2. Rationale for Data Selection . . . 27
4.3.3. Cleaning the Data . . . 29
4.3.4. Constructing the Data . . . 29
4.4. Modelling . . . 31
4.4.1. Selecting the Modelling Technique . . . 31
4.4.2. Designing the Validation Process . . . 32
4.4.3. Building the Model . . . 32
4.4.4. Assessing the Model . . . 36

5. Evaluation **39**
5.1. Experimental Setup . . . 39
5.2. Evaluating Relevance of Data and Code Provisioning for ARDS . . . 40
5.3. Pre-processing and Clustering Result Evaluation . . . 42
5.4. Gridsearch Result Evaluation . . . 43
5.5. Review of Applied Processes . . . 43

6. Conclusion **47**

References **48**

Bibliography **49**

A. Appendix **53**

List of Figures

2.1. Description of DBSCAN Clustering Parameters. 10

4.1. Overview of the modelling process. 17

4.2. Number of admissions for an attributeID in the MIMIC database. . . 24

4.3. Number of patients for an attributeID in the Imputed Data. 25

4.4. Number of patients for an attributeID in the reduced data. 28

4.5. Result changes at three steps of pre-processing. 30

4.6. Component analysis of the data. 33

4.7. K-distance graph of the data. 35

5.1. Graphical depiction of attribute selection at the major steps of the modelling process and pre-processed data provisioning pipeline for medical users. 41

5.2. Different possible combinations of minpts and epsilon and the number of clusters produced 43

5.3. Four different clustering results with their respective parameters. . . . 45

5.4. ARDS patients as diagnosed using the Horowitz index. 46

List of Tables

2.1. Selected information about the MIMIC-III database[1]. 6

5.1. Technical specifications of each of the servers in the JURON cluster[2]. 39

5.2. Technical specifications of the DEEP-ER SDV Prototype[3]. 40

5.3. Time required to complete the pre-processing section of the code on each type of architecture. 42

List of Code Snippets

4.1. Print number of patients for each attribute in MIMIC 23
4.2. Extracting the number of patients from the imputed data represented in each attribute. 25
4.3. Selecting patients based on number of samples 27
4.4. Reading imputed data and extracting the relevant features. 28
4.5. Initial pre-processing steps, revised below. 29
4.6. Pre-processing the data by applying rolling average for smoothing and standardising. 30
4.7. Diagnosing ARDS using the Horowitz index. 32
4.8. Determining principal components of the data based on information content. 33
4.9. Applying PCA and computing the K-distance values. 34
4.10. Applying the DBSCAN algorithm and saving the results to an output file. 34
4.11. Applying gridsearch in serial. 36

5.1. Writing diagnosis results in the output file. 44

A.1. The code that was developed for this thesis. This code is only meant for serial implementation. 53

Abbreviations

- AF - Atemfrequenz (breathing rate)
- ARDS - Acute Respiratory Distress Syndrome
- ASIC - Algorithmic Surveillance of ICU Patients
- CPU - Central Processing Unit
- CRISP-DM - Cross-Industry Standard Process for Data Mining
- csv - Comma-Separated Values
- DBSCAN - Density-Based Spatial Clustering of Applications with Noise
- DDR - Directly Density Reachable
- DEEP-ER - Dynamical Exascale Entry Platform - Extended Reach
- EHR - Electronic Health Record
- eps - Epsilon
- F_iO_2 - Fraction of Inspired Oxygen
- HF - Herzfrequenz (heart rate)
- HPC - High Performace Computing
- HPDBSCAN - Highly Parallel DBSCAN
- ICCA - IntelliSpace Critical Care and Anesthesia
- ICU - Intensive Care Unit
- ICD - International Statistical Classification of Diseases and Related Health

Problems

- JURON - JUelich neuRON
- MIMIC-III - Medical Information Mart for Intensive Care III
- minpts - Minimum Number of Points
- P_aO_2 - Partial Pressure of Arterial Oxygen
- PCA - Principal Component Analysis
- PEEP - Positive End-Expiratory Pressure
- Pplat - Plateau Pressure
- RWTH - Rheinisch-Westfaelische Technische Hochschule
- SMITH - Smart Medical Information Technology for Healthcare
- SQL - Structure Query Language
- Vt - Tidal Volume

Acknowledgments

I would like to express my deepest gratitude to everyone who has been a part of my experience during these two years working towards achieving my Masters Degree at the University of Iceland, but most notably Prof. Sigurður Brynjólfsson and Prof. Dr.-Ing. Morris Riedel for their constant support, follow-up, and guidance. My thanks also go to every Professor and Instructor with whom I have taken courses, and the staff of the School of Engineering and Natural Sciences.

Additionally, I would like to thank Prof. Dr. med. Gernot Marx and Prof. Dr. rer. nat. Andreas Schuppert for their assistance at Uniklinik-RWTH Aachen and as part of the Joint Research Centre for Computational Biomedicine respectively, as well as members of staff in both institutions, most notably Oliver Maassen, Richard Polzin, and Konstantin Sharafutdinov.

Finally, my thanks go to the staff at Forschungszentrum Juelich for providing me with access to the JURON and DEEP-ER SDV clusters in order to complete my research, and specifically to Dr. Gabriele Cavallaro and Rocco Sedona for their assistance along the way.

1. Introduction

1.1. Motivation

In the statement of the rationale for the Algorithmic Surveillance of ICU patients (ASIC) use case, which is part of the Smart Medical Information Technology for Healthcare (SMITH) consortium [4], the researchers claim that around two million patients are currently being treated at Intensive Care Units (ICUs) around Germany and that number will only increase in the coming years [4]. It is safe to assume that this information is somewhat applicable around the world, and will thus pose new challenges that the medical community will have to overcome in order to keep providing effective healthcare. For this reason, and since the advent of Electronic Health Records (EHRs) [5], much research has been done towards the integration of technology in the analysis of patients' medical data in order to assist in forecasting, prevention, diagnosis, and treatment [5–11].

This thesis offers an approach to extract, process, and cluster patient medical information which highlights the benefits of applying advanced computing technologies for the service of healthcare research and industry. In essence, this master thesis investigates the use of parallel and High Performance Computing (HPC) technology to read information from the Medical Information Mart for Intensive Care III (MIMIC-III) database [1], extract relevant information towards diagnosing Acute Respiratory Distress Syndrome (ARDS) [12], perform pre-processing steps and dimensionality reduction, and apply density-based clustering on the resulting data.

In order to perform these tasks, we (a) use state-of-the-art tools including an HPC-ready implementation of the Jupyter Notebook as a programming environment, which in itself is an immense advancement in terms of making HPC more accessible and usable by clinical experts, (b) develop a pre-processing algorithm that can efficiently extract information from a medical database and can easily be adapted to any medical condition given that the required parameters are known, and (c) perform a gridsearch that allows the visualisation of the most effective parameter values for clustering the pre-processed data, thus performing an initial exploratory data analysis while more elaborate models will be investigated in subsequent PhD studies.

Through this approach, we aim to highlight the significant speedup that can be achieved in data read-write processes by using parallel programming and HPC architecture. To properly structure the description of the modelling and evaluation processes involved in this research, we structure them based on the Cross-Industry Standard Process for Data Mining (CRISP-DM) [13].

Finally, we use the Berlin definition [14] for ARDS diagnosis in order to develop a tool that creates a list of the patients from the available pool that fulfill the criteria for ARDS, which is then used to evaluate the clustering results.

1.2. Research Questions

In this thesis we discuss the background information acquired, the methods and approaches implemented, the obstacles faced, and the solutions developed in the process of answering the following research questions:

- Can we develop methods to extract relevant data from the MIMIC-III database, label patient data, and pre-process them for ARDS research?
- Can we employ current HPC technology and different available HPC architectures (JURON, DEEP-SDV) in order to accelerate the data pre-processing and clustering of large patient datasets while not losing sight of usability for medical experts?
- Can we investigate Density-Based clustering techniques on patient data of ARDS in the ICU as a first step of an exploratory data analysis?

1.3. Thesis Structure

This thesis is divided into six chapters as follows:

- Chapter 1 presents the general relevance of the research in question and introduces the research questions that we attempt to answer.
- Chapter 2 presents the background information and the fundamental concepts upon which our research is based.
- Chapter 3 reviews related work in the field and discusses the current state of

affairs in terms of clustering applications and HPC use in ARDS research.

- Chapter 4 follows the initial phases of the CRISP-DM model by introducing the tools that will be used and describes the model that was developed in order answer the research question from Chapter 1.

- Chapter 5 continues with further CRISP-DM phases and describes the experimental setup used to perform the modelling steps, discusses the results of our modelling, and evaluates them with respect to the available validation tools.

- Finally, Chapter 6 provides a summary of the work that was done and presents the future work that this research paves the way for.

2. Foundations

2.1. The MIMIC-III Database

Acquiring access to Electronic Health Records (EHRs) [5] of patients in the Intensive Care Unit (ICU) is in most cases problematic as it raises concerns in terms of the patients' privacy. It is well established in the medical field that some information is shared among medical professionals to draw conclusions as to a specific condition, prepare a treatment plan, follow up on progress of the medical condition, and if possible gain new information from the changes in the patients' state. It is however not always clear what extent of the data can be shared within the broader public medical sciences community, and what should be kept private. Similarly, dealing with real-world data that is constantly being updated poses the risk of corrupting the information contained within and is not a viable option when it comes to analysing procedures and their success rates. Hence, the MIMIC database was established and made available to the medical community as a free-to-access resource comprising a large population of ICU patients [1].

MIMIC-III (Medical Information Mart for Intensive Care III) is the third and most recent version of the database, containing data from around 40.000 patient ICU stays between 2001 and 2012 at the Beth Israel Deaconess Medical Center, Boston MA. The patient information contained within the database is deidentified and made available freely to medical professionals and researchers, and has been used as the basis for research in many publications dealing with respiratory conditions [9], sepsis [7][8], and computer-assisted diagnosis of the patient's medical state [5].

Though access to the database is free, it is not fully open, but access is granted only to parties that have completed a course on the proper handling of medical and other such sensitive information, and who can provide a general direction in which the data will be employed as long as it is not for profit. In our case, the course was completed and access was granted based on proof of supervision as part of a Masters Thesis and a description of the type of research that the data will be employed in.

The information is stored in the database as part of tables, describing much of the patient's stay in the hospital from admission (e.g. including time, date, religion, basis

of admission, initial prognosis, International Statistical Classification of Diseases and Related Health Problems (ICD) codes etc.) through ICU stay highlighting ventilation parameters, physiological measurements, and blood gas and other such analysis data, and finally detailing discharge (whether to hospital, to home, or death of the patient). The database, the size of which exceeds 40GB, is obtained from the PhysioNet servers in Structure Query Language (SQL) format, while manipulation of the data in our case was done using code that queries the dataset, extracts relevant information to the research questions presented above, processes it, and outputs results as will be discussed in the further sections of this report.

Table 2.1: Selected information about the MIMIC-III database[1].

Data collected between 2001 and 2012
Beth Israel Deaconess Medical Center
26 Tables
> 40.000 Patients
All Intensive Care Unit patients
47GB Database size + 26GB for Indexes

2.2. Acute Respiratory Distress Syndrome

Acute Respiratory Distress Syndrome (ARDS) is a condition that affects mechanically ventilated patients in such a way that there is an observed increase in breathing rate, a drop in blood oxygen levels, a loss of lung compliance, and a presence of fluids in the alveoli [12] as first described by Ashbaugh *et al.* in 1967. At the time, it was noted that the patients exhibited similar symptoms to those observed in infantile respiratory distress syndrome, while onset of the condition was generally preceded either by severe trauma or viral infection. Additionally, the authors first described the condition due to the curious case that those affected would not respond to traditional respiratory therapies, while other treatments including Positive End-Expiratory Pressure (PEEP) control and corticosteroid therapy were successful to some extent [12]. Since then, much research has been done on the nature of ARDS, its onset, prevalence, prevention, diagnosis, and treatment as will be described hereafter, though it remains a highly heterogeneous condition.

In their paper on the incidence and outcome of ARDS, Villar *et al.* analyse hospital admissions in several areas in Spain between 2008 and 2009 and highlight ARDS diagnosis during this time interval [15]. Their findings highlight the low incidence of ARDS in patients (7,2 cases per 100.000 patients) while it sheds light on the high mortality rate of this condition where it proved fatal in 42.7% of diagnosed ICU

patients. Additionally, the heterogeneity of ARDS is obvious in the discussion of the paper's results where the authors mention the existence of different criteria for diagnosis of the condition depending on the location of the hospital.

To combat the confusion and uncertainty in defining ARDS, the Berlin Definition of ARDS was introduced in 2012 [14] which clearly defines the physiological parameters based on which accurate diagnosis of patients can be made. The Berlin Definition also takes into consideration defining the severity of the condition based on the PEEP value as well as the ratio of the partial pressure of arterial oxygen P_aO_2 to the fraction of inspired oxygen in the air F_iO_2 (i.e. the amount of oxygen supplied in the mechanical ventilation airflow) as follows:

- Mild ARDS: 200 mmHg $\leq P_aO_2/F_iO_2 \leq$ 300 mmHg, PEEP $\geq$ 5 cmH_2O.
- Moderate ARDS: 100 mmHg $\leq P_aO_2/F_iO_2 \leq$ 200 mmHg, PEEP $\geq$ 5 cmH_2O.
- Severe ARDS: $P_aO_2/F_iO_2 \leq$ 100 mmHg, PEEP $\geq$ 5 cmH_2O.

Even though since the implementation of the Berlin Definition the diagnosis of ARDS has been standardised in medical institutions, the condition remains at the centre of extensive research due to the difficulty in treating it and the different factors that lead to its onset.

Furthermore, in-depth analyses of ARDS patient data have shown that even though the condition is highly heterogeneous, two distinct phenotypes can be defined with one showing a lower mortality rate, while the other presents more severe inflammation. These findings were discussed in two separate papers by Bos *et al.* [16] and by Calfee *et al*[17]. In fact, Bos *et al.* define the subphenotypes as either "uninflamed" or "reactive" which coincide with "phenotype 1" and "phenotype 2" as defined by Calfee *et al.* These findings show that there are intricacies to this condition that physicians should take into consideration before deciding on the most effective treatment method.

2.3. Unsupervised Learning and the DBSCAN Algorithm

2.3.1. Unsupervised Learning

In machine learning applications, unsupervised learning is the process through which relevant information is obtained from a given dataset without any prior knowledge

of the data itself, as opposed to supervised learning where the program is provided with a learning set that allows it to draw features based on which future decisions will be made [18]. In essence, the totality of the data is loaded into the learning application and, based on distance or similarity metrics, relevance and/or internal relationships between the data points are highlighted.

Unsupervised Learning processes are often considered in the analysis of big data [18] and are used to get a better understanding of the data itself. Their potential applications are in clustering data, detecting outliers and noise, and reducing dimensionality in order to simplify grouping and analysis of the dataset in question. From these processes we are mainly interested in *dimensionality reduction* which will be applied to large datasets such as those obtained from EHRs and *clustering* which will link together points that are evaluated as being similar. The dimensionality reduction techniques, as well as the potential clustering approaches that will be discussed in the following section constitute part of the exploratory data analysis processes through which relevant information can be extracted from relatively noisy datasets [18].

Below we discuss one major dimensionality reduction technique, Principal Component Analysis (PCA) [18], then after introducing the feature engineering and selection techniques with dimensionality reduction, we discuss the DBSCAN algorithm, a clustering method that uses distance and density to draw relevant information from a given dataset.

2.3.2. Principal Component Analysis

Principal Component Analysis (PCA) is a commonly used method for dimensionality reduction in high dimensional data. Before explaining the process of how PCA is applied, we first need to introduce the concept of dimensionality reduction. As Shai *et al.* explain it, dimensionality reduction is the process of taking a high dimensional dataset and "mapping" it to a new space of lower dimensions which would make it easier and less compute-intensive to analyse, graph, or comprehend [18].

PCA achieves the tasks described above by performing linear transformations on the data in such a way that the variances of the original data with respect to the new space are sorted in descending order. These variances are seen as vectors and each represents one principal component of the data that represents a significant part of the whole. By analysing the information content of each vector we can determine how many components we need so as to represent the original dataset in a lower dimension, without too much loss of information.

2.3.3. Clustering using DBSCAN

Choosing a specific approach to clustering a dataset often has a great effect on how the results are interpreted in the end and therefore, it is essential to understand the data being analysed as well as to know the specific means by which the clustering algorithm will find relevant features within the information. Knowing that medical data is rarely ever complete, especially given the fact that the information contained in the dataset selected in this research is of a relatively large size and extends over a long duration, our selection of a clustering algorithm needs to take into consideration the fact that noise constitutes a major part of the dataset, that missing values have to be accounted for, and that the high dimensionality of the data might hinder the analysis process.

Ester *et al.* [19] proposed in 1996 in their paper "A density-based algorithm for discovering clusters in large spatial databases with noise" a clustering approach that fulfils most of the requirements described above. Their algorithm named DBSCAN (Density-Based Spatial Clustering for Applications with Noise) is one of the most widely used clustering methods that is currently still in use and has had many variations that have allowed it to adapt to a wide range of applications as will be discussed below.

In contrast to other clustering techniques like K-means clustering [20], DBSCAN algorithm does not take into consideration a pre-defined number of clusters. As shown in Figure 2.1, and in order to identify an arbitrary number of clusters, it rather defines each point as either a core, border, or noise point. This definition is done by counting the number of datapoints within a radius ε around each point; if the number of samples including the point being considered is equal to or greater than a predefined *minpts* value (minimum number of points), then the point is considered a core point. The points within the ε radius are thus directly density reachable (DDR) and belong to the same cluster. Points that do not achieve *minpts*, but are directly density reachable are considered to be border points and thus define the borders of a cluster. Finally, data points that are not directly density reachable and do not achieve *minpts* are thus considered noise points.

As ε and *minpts* are the only parameters necessary for the application of the DBSCAN algorithm, the process of selecting these values greatly affects the number, shape, and size of the obtained clusters. Accordingly, we choose *minpts* and ε based on specific criteria which will be introduced in the Modelling and Evaluation chapters.

One of the major advantages of density-based clustering is the fact that non-globular clusters do not pose a challenge to the algorithm, as opposed to K-means and hierarchical clustering. The approach of defining core, border, and noise points allows

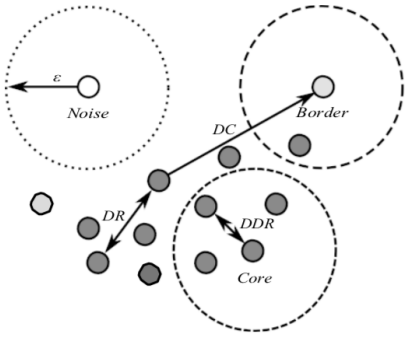

Figure 2.1: Description of DBSCAN Clustering Parameters.

DBSCAN to discover clusters within clusters. Furthermore, DBSCAN specifically presents the extra advantage of taking noise into consideration in its application, thus filtering the output results further and ultimately providing more precision in terms of clustering.

On the other hand, DBSCAN presents some shortcomings when the densities of the clusters vary greatly, thus causing erroneous readings. Additionally, the resulting clusters greatly depend on the choice of ε and *minpts* which requires the researcher to verify their choice of parameters or otherwise risk obtaining incorrect results. In fact, the interpretation of results based on the use of different parameters for clustering needs to be evaluated by medical experts and as such can not be fully performed in this master thesis. Instead as the SMITH Project advances, feedback will be provided in order to enhance these results. Finally, DBSCAN is not effective when it comes to clustering high-dimensional data which, especially in our case, would require a reduction in the dataset's dimensionality, a step which, if the data is not well understood, might cause erroneous readings.

2.4. Parallel Computing and Clustering

2.4.1. Parallel Computing and High Memory

Despite the constant advancements in technology leading to the development of cheaper, more efficient, and more powerful electronic components, Moore's Law, which states that the number of transistors within a processor will double every two years, seems to be reaching a plateau at this time as the size of the components draws closer to the atomic scale. This however, is countered by the implementation of parallel processing and High Performance Computing (HPC) technology which

allow for a great amount of scaling and speed-up.

HPC is the concept of developing architectures that allow Central Processing Units (CPUs) to intercommunicate efficiently via high-performing interconnections, thus completing highly compute-intensive operations by dividing the workload. This is possible through the use parallel computing which is a process by which several cores/CPUs perform a given task in a cooperative manner [21]. Programming in parallel, in any language, is a challenging prospect as it requires the person to think in such a way as to make use of the available architecture and cores in the most efficient manner. In this type of programming the maximum speed-up occurs when the load is balanced on all components. Among other things, this technology has allowed the visualisation and simulation of extremely complex concepts such as fluid flows, both laminar and turbulent.

In this report we perform analyses on two HPC architectures of the Forschungszentrum Juelich (FZ-Juelich) the properties of which are discussed in tables 5.1 and 5.2 presented in Chapter 5. These supercomputers are JURON (JUelich neuRON) and the Dynamical Exascale Entry Platform - Extended Reach (DEEP-ER), and will provide the platform over which data manipulation is performed in a semi-automated fashion, as well as to assist in the selection of the most effective parameters for the clustering algorithm.

2.4.2. HPDBSCAN and Further Work

Knowing that applying DBSCAN clustering on a dataset as large as the MIMIC-III database, which involves time series data for several attributes of tens of thousands of patients, would be a heavy load for any basic computer, the rational next step would be to perform the required calculations using more advanced architectures than standard laptop or desktop PCs. Hence, we look into the research of Götz *et al.* [22] in their paper "HPDBSCAN - Highly parallel DBSCAN" which presents an application of the aforementioned clustering algorithm that is adapted to parallel computing architectures including scalable data infrastructure. In their paper, the researchers describe their approach and compare it to other attempts at parallelising the DBSCAN algorithm, which comes down to three features: 1) reducing the total workload by splitting the tasks to be performed over the totality of the available architecture, 2) preprocessing the data indices so that all parts of the data are know by the system and every part can be traced and collected, and 3) a rule-based merging scheme for the created clusters whereby the clusters formed at each node can be linked together to define the final clustering scheme that was produced by the algorithm. They were able to show that HPDBSCAN presents sustainable scalability and speed-up with increasing size of a given dataset that was used for benchmarking as opposed to another approach at DBSCAN parallelisation. At the

time of writing, it appears to be the best parallel and scalable DBSCAN approach for HPC systems and is therefore picked in this thesis as one potential tool for data analysis.

3. Related Work

3.1. Clustering and Feature Analysis in the MIMIC Database

As described in the Foundations chapter, the MIMIC-III database is a treasure trove of information for biomedical researchers given that it deals almost exclusively with critically ill patients and contains a large amount of medical parameters, physiological signals, and waveforms. For this reason much research has been done on clinical conditions that occur in the ICU by analysing the data contained within the MIMIC database and drawing conclusions from the observed patterns. In this section we will consider two such approaches, one attempting to map patient trajectories using deep learning while the other deals with sepsis, and then discuss other work dealing with the computational processing of MIMIC and other EHR data. Hence, to the best of our knowledge, no published work on clustering with feature analysis on ARDS has been done.

Beaulieu-Jones *et al.* [5] implemented deep learning techniques on patient data extracted from the MIMIC database and were able to develop an algorithm that predicts patient state evolution and response to the healthcare being provided. In their approach, they applied autoencoders to reduce the dimensionality of the data and to group them into clusters. This data would then be analysed in a unsupervised manner (and evaluated using supervised learning techniques) to predict possible outcomes for the patient and select the most likely. No specific condition was analysed for this approach, though the authors refer to potential applications in chronic diseases such as type 2 diabetes.

Ghalati *et al.* [7] considered phase transition in their research on sepsis, focusing mainly on the warning signs that highlight the impending shift in a patients state from chronic to acute. Their approach required the implementation of machine learning techniques such as support vector machines and logistic regression as well as a new concept of surprise loss to analyse the data and differentiate non-sepsis patient from those developing septic shock, while also highlighting the markers leading to the tipping point so as to signal disease onset.

In their research, Afshar *et al.* [10] applied Natural Language Processing (NLP) and Machine Learning (ML) techniques on the medical records of burn and trauma ICU admissions in order to diagnose ARDS. Their approach provided diagnosis with an accuracy of 83% as opposed to 67.3% for the traditional approach of determining the P_aO_2/F_iO_2 ratio and PEEP at 7 days after the onset of lung injury.

Similarly, Huddar *et al.* [9] showed the potential effectiveness of analysing clinical notes in assisting diagnosis of critical respiratory failure. Their research takes into consideration an earlier version of the MIMIC database, and extracts from it features that indicate the onset of post operative respiratory failure well before it is diagnosed by the clinicians involved.

Komorowski *et al.* [8] implemented machine learning techniques, namely reinforcement learning to create a computational model that is able to diagnose sepsis and suggest treatment strategies in order to curb the progression of the condition. Their research shows the potential that such machine learning and automated data analysis techniques have as diagnosis tools as they are able to detect patterns in large scale data that would otherwise go unnoticed by a human observer.

3.2. Use of HPC Technology in ARDS Data Analysis and Clustering

While reviewing related work and publications regarding ARDS and HPC, we found that, regardless of the research done on both of these concepts separately, little to no work has been done on combining them. However, we did find a paper that, instead of analysing or clustering ARDS data, attempts to model it in a simulation environment on HPC. The paper by Das *et al.* [11] describes the development of a model that simulates the effects that ARDS has on cardiovascular and pulmonary systems of patients. Their model, which runs on HPC systems at the University of Warwick, takes for input patient data and implements it in such a way as to be able to respond to changes in the environment where the researchers were able to highlight similarities between the outputs of the virtual patient and the clinical trials.

This work defines the potential applications that HPC has in ARDS research as it provides an efficient platform over which to develop simulations that could either be used to train physicians or adapted to predict the direction in which a patient's health state is evolving.

On the other hand, the available literature does present a growing interest in HPC

applications in healthcare as can be seen in the workshop report on "High Performance Computing in Health Research", detailing the discussions and outcomes of the workshop organised by the European Commission in Brussels in October of 2014 [6]. In this report we can highlight the trend towards encouraging the development of HPC in the European Union, as well as the growing market for this technology in healthcare applications, mainly in drug development and health research.

4. Data Analysis and Modelling Process

After introducing the basic concepts that govern the work presented in this thesis, and describing research that attempts similar approaches or deals with similar medical conditions, we move towards describing the work that we set out to achieve in our study. The process by which the data is selected, manipulated, and analysed described in this chapter, as well as the evaluations steps in the following chapter are organised and presented following the Cross-Industry Standard Process for Data Mining (CRISP-DM) [13]. The major steps of this process are laid out in Figure 4.1 presented below.

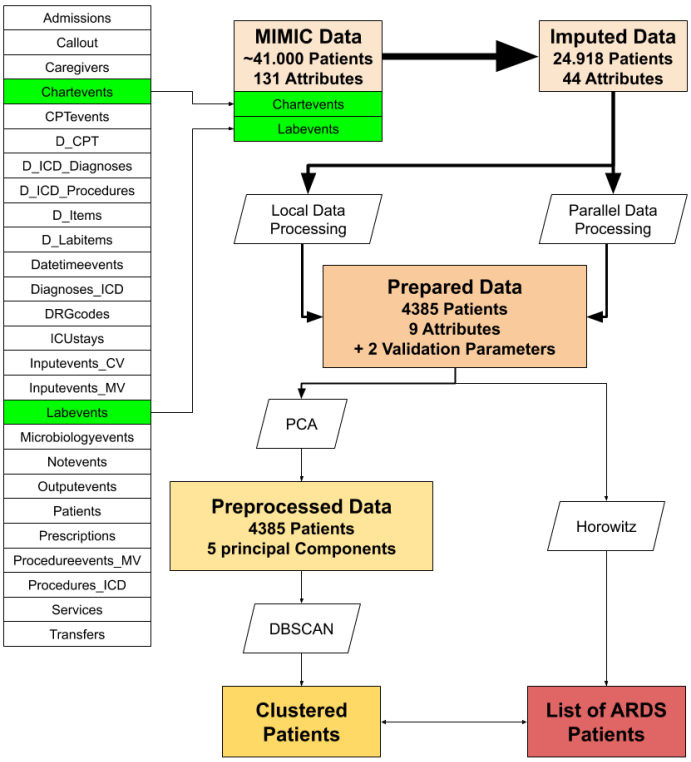

Figure 4.1: Overview of the modelling process.

4.1. Business Understanding

4.1.1. Business Objectives

As discussed in Chapter 2, ARDS is a highly heterogeneous condition that affects a significant number of mechanically ventilated ICU patients and has a high morbidity and mortality rate (42.7% [15]). Treatment methods of ARDS vary in effectiveness depending on their implementation period, and therefore early detection and warning systems would be an invaluable tool in combating this condition.

The business objectives of this thesis, as defined by the research questions presented in Chapter 1, are to develop a system that can analyse an input set of ICU patient information, uncover underlying interactions and connections, and cluster the data in an unsupervised manner which will allow us to identify subgroups of patients that may develop ARDS.

In the process of achieving the objectives described above, we provide a tried-and-tested method of data manipulation in terms of pre- and post-processing, analysis, and storage, both in a serial and parallel environment, that will be freely accessible by colleagues in the Algorithmic Surveillance of ICU Patients (ASIC) use case of the SMITH project in specific, as well as future researchers in this field in general. In terms of the specific clustering results obtained using our method, namely DBSCAN, the analysis method used is a novel approach and the parallelisation of the work paves the way for new methods for dealing with compute- and data-intensive analyses of medical information. It goes without saying that objectives with respect to the clear interpretation of clustering results and further predictive models require medical know-how and is as such out of scope of this master thesis.

4.1.2. Situation Assessment

Inventory of Resources

For the purpose of completing the tasks defined in our research objectives we employ an extensive set of resources and tools; these include the database itself, the programming language used to analyse it and the modules it employs in order to perform the tasks, and finally the platforms, both hardware and software, over which the process is done. What follows is a description of these tools.

The MIMIC-III database is available for download from the Physionet servers for

confirmed and approved research purposes as was described in the foundations section in more detail. After approval, querying the data is done using a SQL connection via the programming software being used. In our case, all programming was done in Python running on a Jupyter Notebook [23], with the scikit-learn, feather, pandas, numpy, matplotlib, and mpi4py modules installed. These modules are mainly adapted for analysis, lightweight storage, data manipulation and read-write processes, array operations, and data visualisation respectively.

Keeping in mind the size of the database (upwards of 40GB), the platform over which the work is done is of great importance as storage capacity and memory will ultimately affect performance. For this reason, the analysis is first performed locally on a personal computer in order to provide a benchmark representing serial working environments and illustrating their shortcomings. The local machine is a MacBook Pro (2015) with a 2,7 GHz Intel Core i5 processor, 250GB of internal SSD storage capacity, and 8GB of 1867 MHz DDR3 RAM.

Another resource that will be used in the remainder of the data analysis is the JURON cluster of the Jülich Supercomputing Centre, which is a pilot system developed by IBM and NVIDIA, and the DEEP-ER SDV cluster, which is a prototype cluster developed as part of the DEEPprojects, a collaborative research project funded by the European commission. JURON is made up of 18 servers, while the DEEP-ER cluster has 16 nodes and a booster of 8 CPUs. The specifications of both HPC clusters are presented in Tables 5.1 and 5.2 provided in Chapter 5.

Constraints

In terms of data use, manipulation, and distribution, a lot of the work done in this project involves dealing with information that is either meant for personal use or completely confidential. Certainly this would pose several risks which will be discussed below, however the restrictions on the data also put limitations as to how it can be used. In essence, though the MIMIC database is open, access to it is limited to research purposes that have been approved by the providers, and accordingly is not meant for distribution. Thus, the results of the research being performed can be discussed, though special care must be placed in the process so that none of the original data is divulged.

Similarly, analysis that was performed by the Bayer pharmaceutical group resulted in a list of subjects from the database that were confirmed ARDS patients, which would greatly simplify the validation and evaluation of the clustering results. However, since their research has not been made public and is completely confidential, it cannot be used in this master thesis, but is available for further research in the joint SMITH project.

Risks and Contingencies

In terms of risks involved in the process of answering the research questions listed above, the analysis being performed is expected to not lead to divulging any information that is determined to be confidential or for internal use. The data obtained from Bayer cannot reasonably be used without risk of it being inferred from the results and is thus dropped from analysis.

On the other hand, information is freely transferred between the partners in the ASIC use case for the purpose of advancing the project towards fruition, however it is not yet published and should thus not be made public in any way that may undermine the work being done. In the same scope, the work that is described in this thesis will ultimately benefit the partners in this project in terms of the data manipulation that was performed, the programming that was done, and the hardware that was employed. On the larger scale, other potential researchers in the diagnosis and treatment of ARDS will benefit from this research after the publication of our approach and results. By taking care of leaving sensitive material out and discussing with project partners, it is clear that the Master Thesis can be published as is.

4.1.3. Goal Description

While attempting to uncover features and signs that pinpoint the onset of ARDS in the patient data is our ultimate goal, we first need to understand the data itself. This becomes mostly clear when we are faced with the large amount of information that every individual patient ID is associated with in terms of admission notes, vital signs, drug supply, nursing and doctor's notes, ICD codes, etc. Hence, it is essential to perform a significant amount of data reduction and manipulation before even attempting to consider the individual patients themselves. Afterwards, further analysis and reduction is applied to the dataset that will result in data that can be clustered in a meaningful manner. This process of navigating the database, highlighting the relevant information, collecting, cleaning, and analysing it, and finally storing it in easily accessible files for clustering is in itself one of the ultimate goals of this research. The reason for this inclusion is to simplify the preliminary work that future researchers in this area will have to perform by providing a road map for adapting MIMIC-III data so that it can be used in ARDS research.

In parallel, the goal of unsupervised learning in general is to underline features within a given dataset that would imply correlation, causality, or similarity, which would not be immediately apparent. This goal is mirrored in our approach of cluster-then-predict where we intend to highlight inherent differences between ICU patients that

would ultimately lead some of them to develop ARDS, as well as between subclasses of ARDS patients as was discussed in the papers by Bos *et al.* and Calfee *et al.* Another goal is to use High Performance Computing (HPC) to speed-up the data analysis process and to provide access for medical experts in a seamless fashion (e.g. using Jupyter Notebook instead of SSH Connections). This is needed since medical experts require access to the clustering tool, but are not technical experts to perform SSH-key based access or to use scheduling principles of HPC machines.

4.1.4. Project Plan

In attempting to answer the research questions proposed in the introduction of this thesis report, we employ several tools and techniques as described above that will assist in the development of our problem solving approach. The choice of these tools is described hereafter, though it is mainly based on the current state of the art in terms of technology, trends in programming, and in some cases, necessity.

In first place, performing our analysis on the MIMIC-III database is an essential part of the project and is based on its ready availability for research such as ours. Logistically speaking, working with a team from the Uniklinikum Aachen, we would have been able to request access to the IntelliSpace Critical Care and Anesthesia (ICCA) database [24], however it is significantly smaller than the MIMIC dataset, is proprietary, confidential, and only accessible to faculty and staff of the University of Aachen.

Second, and given our understanding of the data, we know that the samples are inherently noisy, numerous, and high-dimensional. The choice of DBSCAN, an algorithm having noise reduction and outlier detection as a major part of its application, with confirmed uses on large scale datasets [25], for clustering the data is thus explained.

Third, employing Python in all phases of project development, from pre- to post-processing, passing through clustering, is based on this specific programming language's extensive relevant literature, modularity, and widespread implementation in machine learning applications. It is a relatively easy language to learn and apply, and the add-on modules available make it a sturdy tool for use in data manipulation and unsupervised learning approaches as the ones described in this report.

Finally, and given the data- and compute-intensive nature of the application in question, we seek out the substantial speed-up and storage that can only be achieved using parallel computing architectures. Though the database and all processing was initially performed on a local machine, loading the data onto the clusters and implementing parallelisation of the program presented larger capacity for file storage

and scale-up. The benefit of scale- and speed-up however should not come at the cost of usability for medical expert users and as a consequence, Jupyter Notebook is planned as a compromise between technical difficulty and usability.

4.2. Data Understanding

4.2.1. Collecting Initial Data

After completing a required course on the ethical handling of patient data and obtaining access to the MIMIC database, it can be cloned onto a local machine and read via a SQL database management software as described in the extensive tutorial posted on the Physionet website. In our case, we obtained access to the RWTH-Aachen University servers which contain a copy of the database (thus eliminating the need to store it locally) and can be accessed using a Python code. By calling the function getCnx() defined in this code, and naming the specific required table (from the list presented in the Foundations section), we can load patient information into a Pandas dataframe [26] for analysis.

The tables that were loaded are "chartevents" and "labevents" as they hold the attributes most closely related to the research we are performing. The obtained dataframe consisted of approximately 45.000 unique patient IDs. It is worth noting at this point that these patients have all been admitted to the ICU but, given that we are only interested in mechanically ventilated patients as the population most likely to develop ARDS, further data reduction is necessary.

The reduction was done by project partners from the University of Aachen who were able to select only the mechanically ventilated patients from the list and present them as individual files in .feather format [27]. The number of patients was thus reduced to 24.918 while some data manipulation was performed on the attributes and the values contained within each file.

4.2.2. Data Description

The data that is loaded from the MIMIC database is a time series data that has many attribute IDs, several of which represent the same measured value. This is made obvious in the figures presented below in our exploration of the data, though in this section it is mentioned in order to justify the data imputation that was performed. A sample of the attributes is presented in Chapter 5.

The imputed data that was received from our partners at Aachen University, besides having a smaller number of patients (having only selected the ones undergoing mechanical ventilation), also has a smaller number of attributes, a more consistent timestamp, and no missing values beyond the first recorded value for each attribute, i.e. the only "NaN" values present in the data are either at the start of the signal, or in completely empty attributes. In total, each patient has a "charttime" attribute which is the timestamp value in 30 minute increments beginning at admission and ending at either discharge or death, followed by 44 attributes listing different vital signs including, but not limited to, heart rate, breathing rate, tidal volume, blood pH, breathing parameters, blood gas analysis results, and blood test results. Each patient's information is saved in a feather format file with title including the unique patient ID. Though this step reduced the overall size of the dataset, it was still somewhat bulky, taking up a total of 5,3GB of space.

4.2.3. Data Exploration

In this section we describe the attributes present in the dataset at the three major steps before we begin our main pre-processing phase. These steps are (1) the complete MIMIC dataset, (2) the imputed data obtained from our partners, and (3) after selection of the most relevant attributes to our research.

What follows is a visualisation of the number of patients who have data recorded for a given attribute in the datasets. The first graph presents the results obtained when querying the MIMIC database, and it clearly shows the large number of attributes and patients, while also the fact that many of them are empty.

Code Snippet 4.1: Print number of patients for each attribute in MIMIC

```
params_intersection = pd.DataFrame()
for i,attr in enumerate(attids):
    query = """(select distinct(subject_id) from
        ↪ labevents where itemid={0}) union (select
        ↪ distinct(subject_id) from chartevents where
        ↪  itemid={0})""".format(attr)
    query = query.replace('[', '(').replace(']', ')')
    data = pd.read_sql(query, cnx)
    params_intersection = pd.concat([
        ↪ params_intersection,data], ignore_index=
        ↪ True, axis=1)
    print(i)
params_intersection.columns = attids
params_intersection.columns = labels
# for each attribute ID list the number of admissions
width = 18
height = 14
plt.figure()
params_intersection.count().sort_values(ascending =
    ↪ False).plot.bar(figsize=(width,height), color
```

```
    ↪ =(0.2,0.4,0.6,0.6))
ax = plt.gca()
ax.set_ylabel("Number of admissions with values for an
    ↪  attributeid", size = 20)
ax.set_title('Number of admissions for an attributeid,
    ↪  mimic', size = 24)
ax.set_xlabel('Attributeid in mimic', size = 24)
ax.grid(True)
ax.tick_params(axis='both', which='major', labelsize
    ↪ =16)
plt.tight_layout()
plt.show()
```

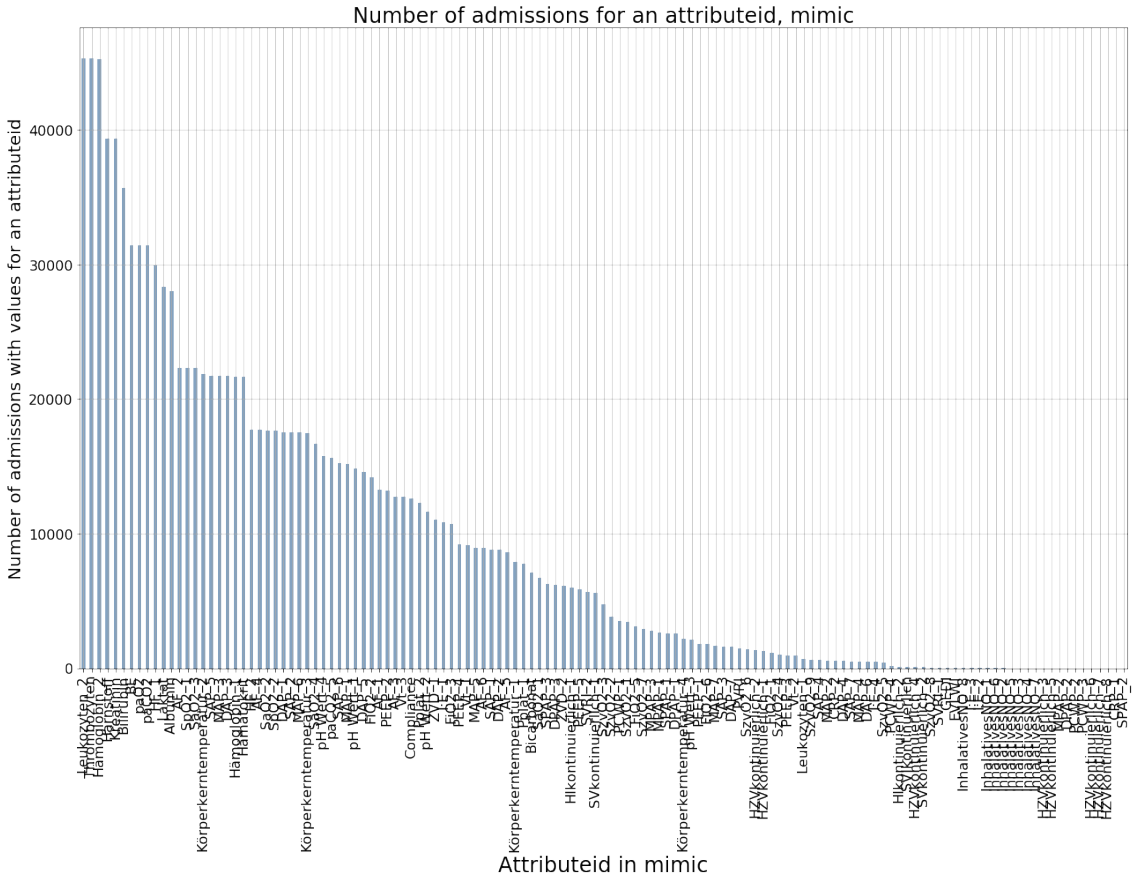

Figure 4.2: Number of admissions for an attributeID in the MIMIC database.

The graph in Figure 4.2 was obtained after running the code presented in Code Snippet 4.1 which queried the database and sorted the output. This code was adapted from the code that our partners at Aachen University use to access the ICCA healthcare database.

Clearly, the number of attributes with little or no values is large and they can be removed; many attributes figure more than once and they can be merged, and given our knowledge that some of these patients were not mechanically ventilated we can

further reduce the dataset. Hence we perform similar steps to chart the number of attributes that figure in the imputed dataset obtained from our partners. The code to perform these actions is provided below, followed by the resulting bar graph.

Code Snippet 4.2: Extracting the number of patients from the imputed data represented in each attribute.

```
for file in list(glob.glob("mimic/*.feather")):
    ds = feather.read_dataframe(file)
    abc = ds.isna().all()
    abc ^= True
    abc = abc.rename(file.replace('mimic/data','').
        ↪ replace('.feather',''))
    out = out.append(abc.drop('charttime'))
out.sum(axis=0).sort_values(ascending = False).plot.
    ↪ bar(figsize=(width,height), color
    ↪ =(0.2,0.4,0.6,0.6))
```

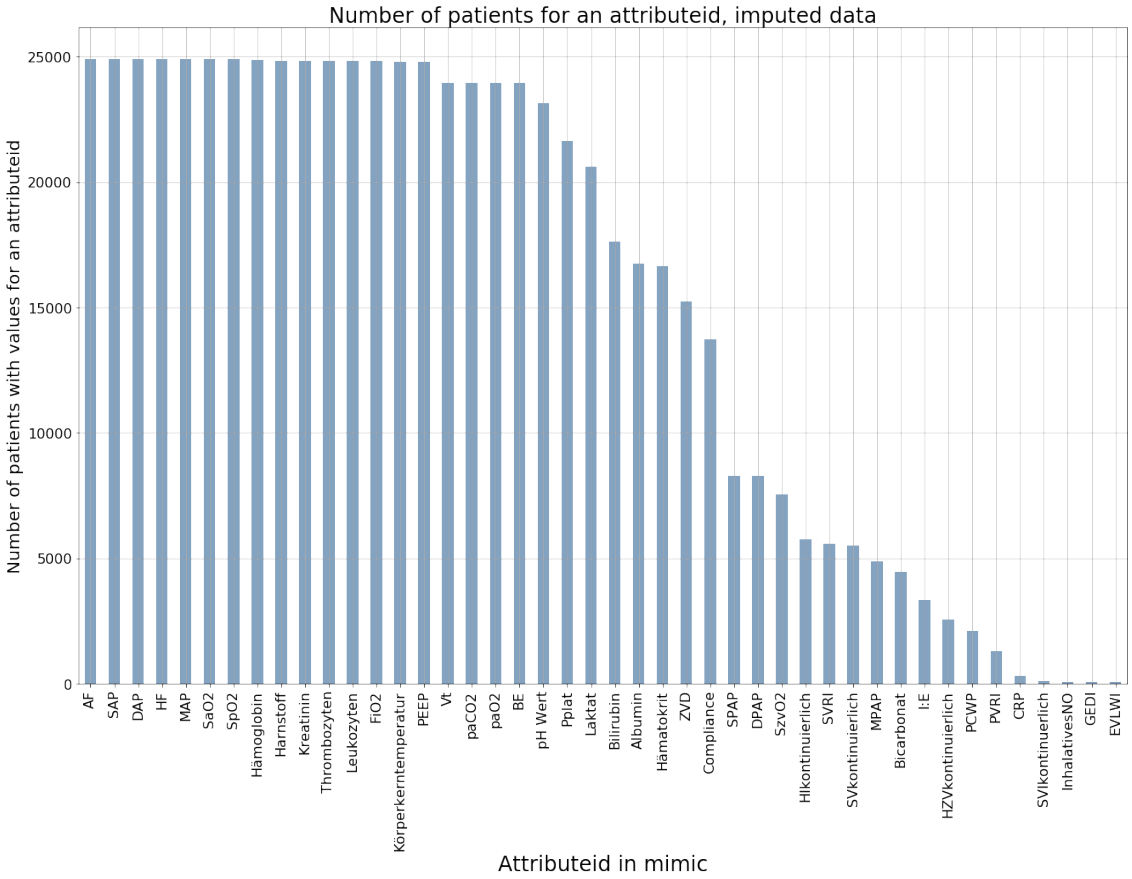

Figure 4.3: Number of patients for an attributeID in the Imputed Data.

Here we can see the immediate reduction in the number of patients from around 45.000 to 24.918 while also the total number of attributes was reduced to 44. We can still observe that many attributes are not represented in a majority of the patients,

while that total number of attributes itself would result in very high-dimensional data. For that reason, we decide to further reduce the selected number of attributes to those present in at least two thirds of the total number of patients as will be discussed below.

4.2.4. Verifying Data Quality

In terms of quality, the data that we are using in our research can be divided into two parts: the raw data obtained from the MIMIC database and the imputed data obtained from the Aachen University partners. The quality of the data figuring in the MIMIC database has been verified on several occasion and in several published papers, has been used on several occasions to uncover patterns in ICU, and has been the basis of much research on automation of diagnosis before rollout to the clinical environment.

The imputed data we received from our partners was drawn directly from the MIMIC database, with little manipulation beyond merging of the attributes and using internal averages to fill missing values. A similar approach was employed within the same team and employed in a published research concerning sepsis [7] and thus we can deduce that the quality of the data can also be verified. From this we can safely say that we base our research on information that is not altered in any way that can skew the results in any direction.

4.3. Data Preparation

4.3.1. Dataset Description

In the previous sections we described the goals that we aim to achieve in our project, at which point we highlighted the benefits that this work would yield regardless of its ultimate success at the clustering phase. Additionally, we also discussed in relative detail the dataset to be analysed in the process. At that level, it became clear that performing our analysis on the whole MIMIC-III database would be cumbersome and inefficient as it presents an excess of information, most of which cannot be used to answer our research questions.

Similarly, the imputed data that was provided by our partners from Aachen University, though significantly reduced in size and complexity, still contains data of little to no relevance to our research. Based on this information, we performed fur-

ther data manipulation that ultimately resulted in selecting the attributes that were judged as most relevant to our research, with a particular focus on ARDS, as well as decreasing the total number of unique patient files to be analysed. At the end of this process we obtained a total of 4385 individual patient files, taking up 1,2GB of storage space, and each containing 9 attributes that were decided to be the most relevant to breathing and oxygenation and most represented. Below we will discuss the process by which this reduction was done and the rationale behind it.

4.3.2. Rationale for Data Selection

After studying the data files we found that their sizes are greatly variable ranging from a minimum of 3 timestamps to a maximum of 14.133 timestamps. Knowing that files with very few samples would yield little relevant information while those with an excessively large number of samples would unnecessarily complicate the data analysis process, we opted to select the patient files that fit within a window of 800 to 2000 timestamps. Below is a snippet of the code that performs the necessary filtering and outputs the patient IDs to a .csv file followed by the bar graph of patients per attribute obtained after this reduction.

Code Snippet 4.3: Selecting patients based on number of samples

```
samples = 200
times = []
for file in sorted(list(glob.glob("mimic/*.feather")))
    ↪ :
    patient = feather.read_dataframe(file)
    if (4*samples < len(patient.charttime) < 10*
        ↪ samples):
        times.append([id])
times = pd.DataFrame(times)
times.columns = ['PatientID']
times.to_csv('allPatients.csv', index=False)
```

In parallel, and from Figure 4.4 presented above, we can see that the attributes are all related to breathing parameters (rate, volume, pressure), cardiovascular data (heart rate, arterial pressure), or blood analysis values (oxygen saturation, immunity, pH, etc.). From these parameters we disregard the ones that are represented in less than 75% of the total number of patients, that is we set out cutoff at 18.700 patients. From the remaining attributes we initially select F_iO_2 (the fraction of inspired oxygen), P_aO_2 (the partial pressure of arterial oxygen), and PEEP (positive end-expiratory pressure) which are directly related to ARDS diagnosis as per the Berlin Definition of the condition.

Additionally, and based on prior knowledge in physiology, we select the remain-

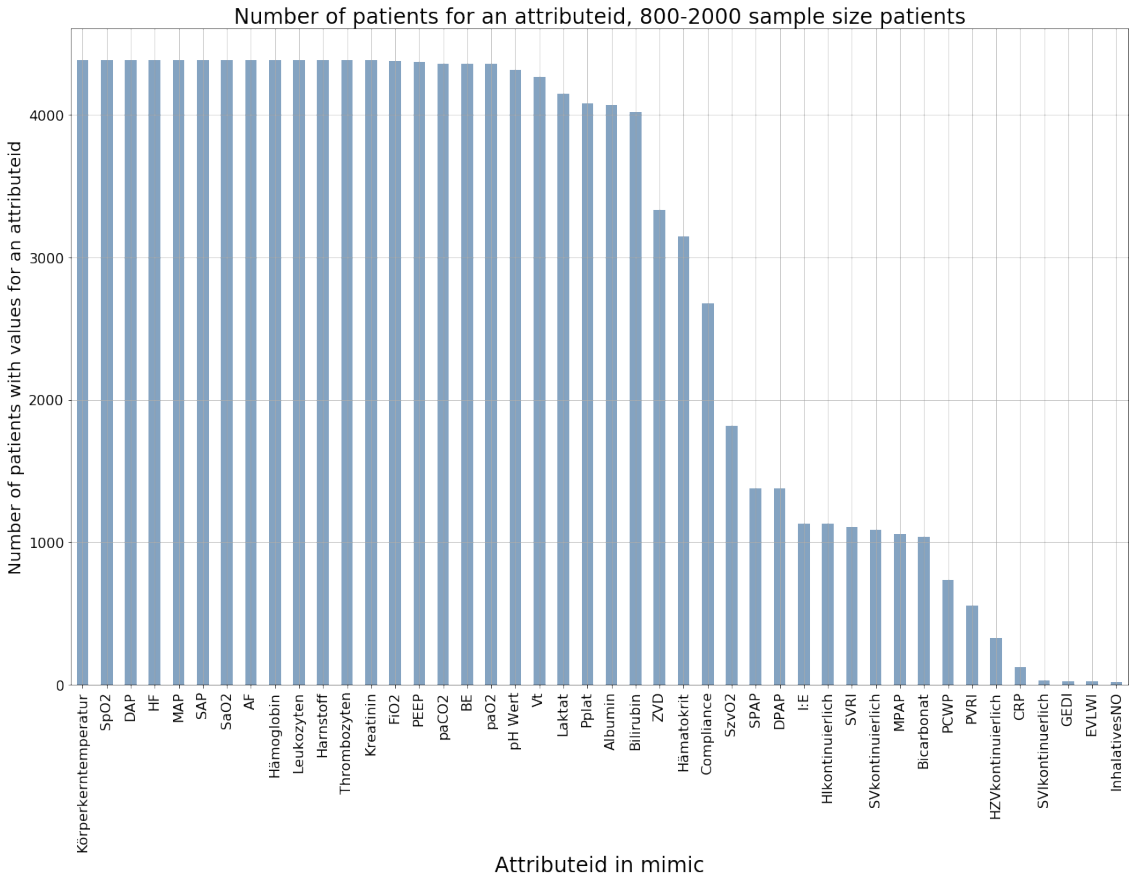

Figure 4.4: Number of patients for an attributeID in the reduced data.

ing attributes based on the fact that they provide further information about the patient's oxygenation, namely breathing rate (AF), tidal volume (Vt), heart rate (HF), hemoglobin levels, plateau pressure (pplat), and blood pH (pH Wert). We included hemoglobin and blood pH levels in our analysis as the latter indicates the reduction in oxygenation as blood becomes more and more acidic, while the former may explain such a reduction if breathing rate remains stable or increases.

The process described above is done using the code presented below which loads a file from the list generated in the first part, and extracts the required information. This code is expanded in the following sections in order to perform the necessary pre-processing steps before analysis.

Code Snippet 4.4: Reading imputed data and extracting the relevant features.

```
number = pd.read_csv('allPatients.csv')
for patient in number.PatientID:
    dataarray = []
```

```
ds = feather.read_dataframe("mimic/data{0}.feather
    ↪ ".format(patient))
data = ds[['charttime','AF','Vt','PEEP','Pplat','
    ↪ FiO2','paO2','Hämoglobin','HF','pH Wert']]
```

4.3.3. Cleaning the Data

When the imputed data was received from our colleagues at Aachen University, they mentioned that missing values within the data were filled out using the mean of their surrounding points. This however did not cover the missing data at the beginnings and ends of the patient attribute values which we needed to fill. The method used also filled in the values with a mean value so as not to affect the signal in any significant way.

After taking care of missing values it was necessary to remove outliers which figure in our plots as spikes in the data and greatly affect the mean and the standard deviation of the signals. This was done by applying a rolling average function on the data which resulted in signal smoothing.

Finally, before attempting to cluster in any way, it was necessary to standardise the data so that the means and standard deviations are within the same ranges. This was done by applying the built-in StandardScaler function of the scikit-learn module in Python.

The code for the pre-processing that we applied is presented below followed by the plot results for one patient attribute before and after applying the filters.

Code Snippet 4.5: Initial pre-processing steps, revised below.

```
data = data.fillna(data.mean())
data = data.fillna(0)
X2 = data.iloc[:,1:].rolling(10).mean()
X2 = X2.fillna(data.mean())
X2 = X2.fillna(0)
newdata = pd.DataFrame(StandardScaler().
    ↪ fit_transform(X2))
newdata.columns = data.drop(columns=['charttime'])
    ↪ .columns
```

4.3.4. Constructing the Data

Based on the Berlin Definition, the current standard used to define ARDS onset in mechanically ventilated ICU patients, diagnosis is based on the value of the Horowitz

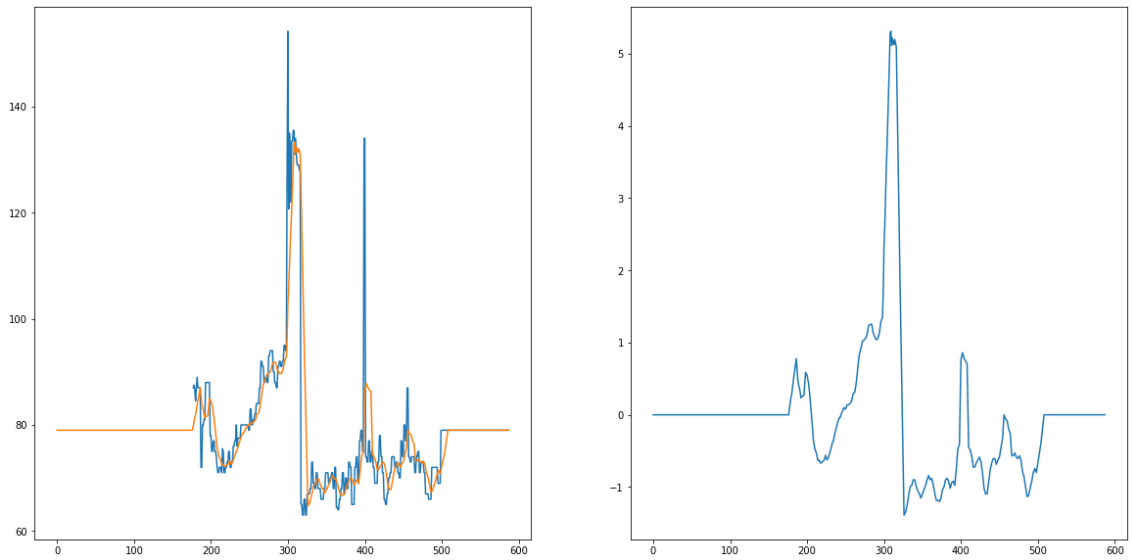

Figure 4.5: Result changes at three steps of pre-processing.

index at 7 days after onset of the injury or of reduced oxygenation as well as other parameters including PEEP and lung compliance. The Horowitz index is defined as the ratio of the partial pressure of arterial oxygen (P_aO_2) to the fraction of inspired oxygen (F_iO_2) and it defines the severity of the condition as follows: a measurement is marked as mild ARDS for a value between 300 and 200, moderate for a value between 200 and 100, and severe for a value below 100.

In order to have label generation for our patient set so as to validate the clustering results, we implement a model to calculate the Horowitz index of every patient and store it as a separate column within the patient file. The function to perform the calculation was obtained from work done by colleagues at Aachen University for the purposes of their research and implemented at the pre-processing phase of our data analysis. The Horowitz value as well as a copy of the PEEP value were preserved while the remainder of the parameters were standardised.

After performing the tasks described above, every patient's data was exported to an individual file in .csv format so that the calculations need only be done once but the data can be accessed repeatedly. All of the processes described here are presented in the code snippet below.

Code Snippet 4.6: Pre-processing the data by applying rolling average for smoothing and standardising.

```
data['horowitz'] = horowitz_sample.
    ↪ calculate_horowitz(data)
X2 = data.iloc[:,1:].rolling(10).mean()
X2 = X2.fillna(data.mean())
X2 = X2.fillna(0)
```

```
    X2['PEEP2'] = X2.PEEP
    newdata = pd.DataFrame(StandardScaler().
        ↪ fit_transform(X2.drop(columns=['horowitz',
        ↪ 'PEEP2'])))
    newdata['horowitz'] = X2.horowitz
    newdata['PEEP2'] = X2.PEEP
    newdata.columns = X2.columns
    newdata.to_csv("data-averaged/{0}.csv".format(
        ↪ patient), index=False)
```

4.4. Modelling

4.4.1. Selecting the Modelling Technique

The goal of this project is also to cluster the data in order to draw out of it some relevant features to our research on ARDS, therefore our modelling technique will be adapted so as to simplify the clustering process. The DBSCAN algorithm that will be employed to cluster the data is able to find arbitrary-shaped clusters efficiently as opposed to other available clustering techniques, as discussed in the foundations section of this report, though it is not well suited to high dimensional data such as that we have obtained thus far. For that reason we apply methods of reducing the dimensionality of the data, namely Principal Component Analysis (PCA).

In order to apply PCA to the datasets, prior analysis is required to pinpoint the number of attributes that best represent the data in question. For this, fitting of the PCA function is first performed on the data that is loaded into one file, at which point a graph can be plotted which shows roughly how many attributes are required so as to represent the data with adequate precision. After this step, when the number of components is selected, PCA is applied and the data is further reduced in such a way that it is better adapted for clustering using DBSCAN.

Additionally, though DBSCAN does not require us to know the number of clusters beforehand, it does take for input two parameters that need to be determined before it can be applied. The *epsilon* radius is determined by plotting the K-distance graph and selecting the distance at which the curve presents an "elbow." The minimum number of points *minpts* is selected as either the number of attributes +1 or its double [19]. Finally, through our analysis of the data it became clear that the potential clusters will not be globular or of any consistent shape, which means that more common clustering techniques such as K-means cannot be used. The sections that follow will present the processes by which the above-mentioned values were selected.

4.4.2. Designing the Validation Process

After completing the clustering of our data we aim to validate our results in order to determine whether the clustering was able to find any relevant patterns in our dataset. We perform this action so that at the roll-out phase we can be certain that the resulting clusters will provide relevant, non-random, information that can be used to diagnose ARDS or highlight the potential that a patient will develop it. To do this validation step, we will need to highlight which patients did develop ARDS by applying the Berlin Definition of the condition.

The calculated Horowitz index and non-standardised PEEP value are loaded from each patient file and, based on these parameters during the timeseries, we create a list of patient IDs marked as ARDS or non-ARDS. In order to populate this list, we obtain the values for both parameters at one point at the beginning of the time-series. If the value for the Horowitz index is lower than 300 (indicating lung injury) we measure the values again at the 400^{th} timestamp which would be around 8 days later. If at this point the Horowitz index is still below 300 and the PEEP is above 5 The patient is immediately labeled as ARDS. The code snippet below was developed to perform the task described in this section.

Code Snippet 4.7: Diagnosing ARDS using the Horowitz index.

```
avgs = []
for file in sorted(list(glob.glob("data-averaged/*.csv
    ↪ "))):
    patient = pd.read_csv(file)
    if (patient.horowitz[150] < 300 and patient.
        ↪ horowitz[400] < 300 and patient.PEEP2[400]
        ↪ >7):
        avgs.append(int(file.replace("data-averaged/",
            ↪ '').replace('.csv','')))
avgs = pd.DataFrame(avgs)
avgs.columns = ['id']
avgs.to_csv('horowitz.csv', index=False)
```

4.4.3. Building the Model

Running PCA on our data is meant to reduce the number of dimensions so as to simplify the clustering process. By fitting the PCA function to our data we can graph to which extent the attributes represent the whole dataset and from that draw the number of components to consider so that the ultimate results would most closely resemble the data. The code used to plot the principal components and the graph obtained are presented below.

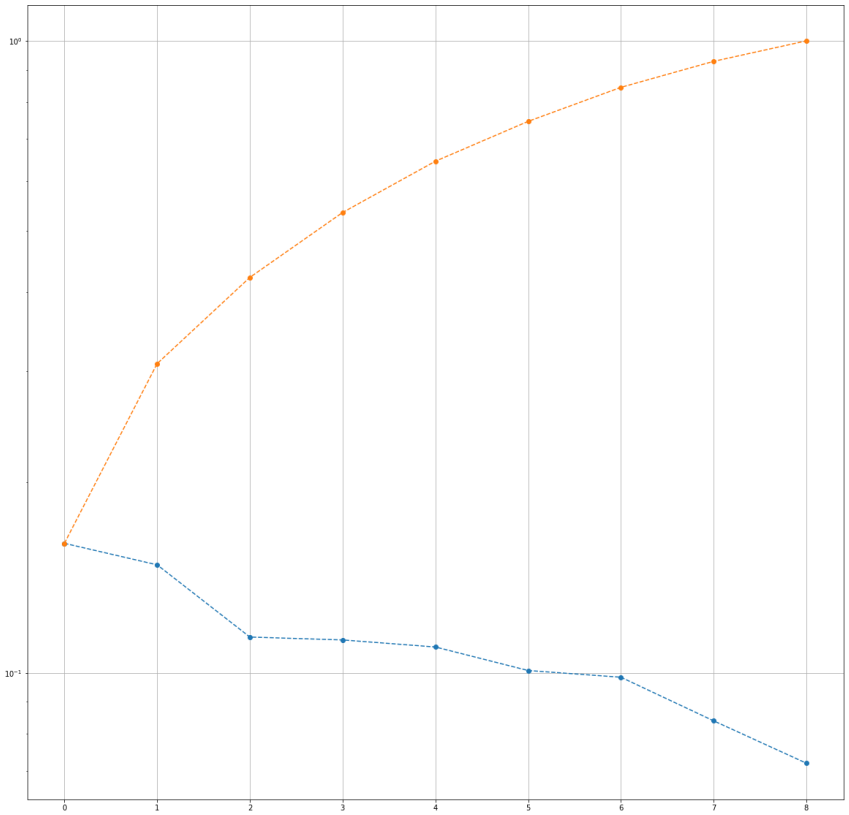

Figure 4.6: Component analysis of the data.

Code Snippet 4.8: Determining principal components of the data based on information content.

```
# Load all averaged patient data and store them in a
    ↪ dataframe
allPatients = pd.read_csv('allPatients.csv')
patients = pd.DataFrame()
for id in allPatients.PatientID:
    patients = patients.append(pd.read_csv("data-
        ↪ averaged/{0}.csv".format(id)))
# Apply PCA to all patients in order to find the
    ↪ number of
# principal components that most closely represents
    ↪ the data.
patients = patients.drop(columns=['horowitz','PEEP2'])
pca = PCA().fit(patients)
plt.figure(figsize=(20,20))
plt.grid(True)
plt.semilogy(pca.explained_variance_ratio_,'--o')
plt.semilogy(pca.explained_variance_ratio_.cumsum(),'
    ↪ --o')
plt.savefig('pca.png', bbox_inches = "tight")
```

In the plot we see that we can perform our cutoff at the 5th component since, at that level, the major part of the data is represented and the information content is not increasing much; thus we can proceed without risk of affecting our results in

a negative way. We apply PCA to the each data file and reduce it to 5 principal components then perform K-nearest neighbour calculations using the "cosine" distance metric for the components over the whole time series and store all the sorted distance results in one variable. Each row of the resulting dataframe is averaged producing a list of the average distances within the whole dataset that is then plotted and the resulting figure is presented below.

Code Snippet 4.9: Applying PCA and computing the K-distance values.

```
# Apply PCA based on the number of components that was
    ↪  decided on.
dist = pd.DataFrame(index=range(0,2000))
for id in allPatients.PatientID:
    pat = pd.read_csv("data-averaged/{0}.csv".format(
        ↪ id))
    comp = pd.DataFrame(PCA(n_components=5).
        ↪ fit_transform(pat.drop(columns = ['horowitz
        ↪ ','PEEP2'])))
# Save the data of each patient as a separate file
    ↪ after applying PCA (to reduce read-write cycles
    ↪ )
    comp.to_csv("data-pca/comp{0}.csv".format(id),
        ↪ index = False)
    nbrs = NearestNeighbors(n_neighbors=5,algorithm='
        ↪ brute',metric='cosine').fit(comp.values[
        ↪ range(0,len(comp)),:])
    distance,indices = nbrs.kneighbors(comp)
    for i in range(0,len(distance)):
        distance[i] = np.average(distance[i,:])
        distance = pd.DataFrame(sorted(distance[:,0]))
        distance = distance.iloc[::-1]
        distance = distance.reset_index()
        distance = distance.drop(columns='index')
        dist = pd.concat([dist,distance],axis = 1)
dist.columns = allPatients.PatientID
# Compute the average of the k-distances and plot the
    ↪ k-distance graph
avg = []
for i in range(0,2000):
    avg.append(np.average(dist.values[i,:]))
plt.figure(figsize=(15,15))
plt.grid(True)
plt.plot(avg)
plt.savefig('distance.png', bbox_inches = "tight")
```

Selecting the *epsilon* value for DBSCAN is not a straightforward approach but rather requires a bit tuning. From the distance graph in Figure 4.7 we select *epsilon* between 0,01 and 0,02. As for *minpts* we will perform clustering based on both recommended values (dim+1 and dim*2) [19] and study other potential values using the gridsearch method. Additionally, the clustering is done at one point for all the patient files, which is a point 4 days after initial signs of lung injury. The code below describes how DBSCAN is applied to the data.

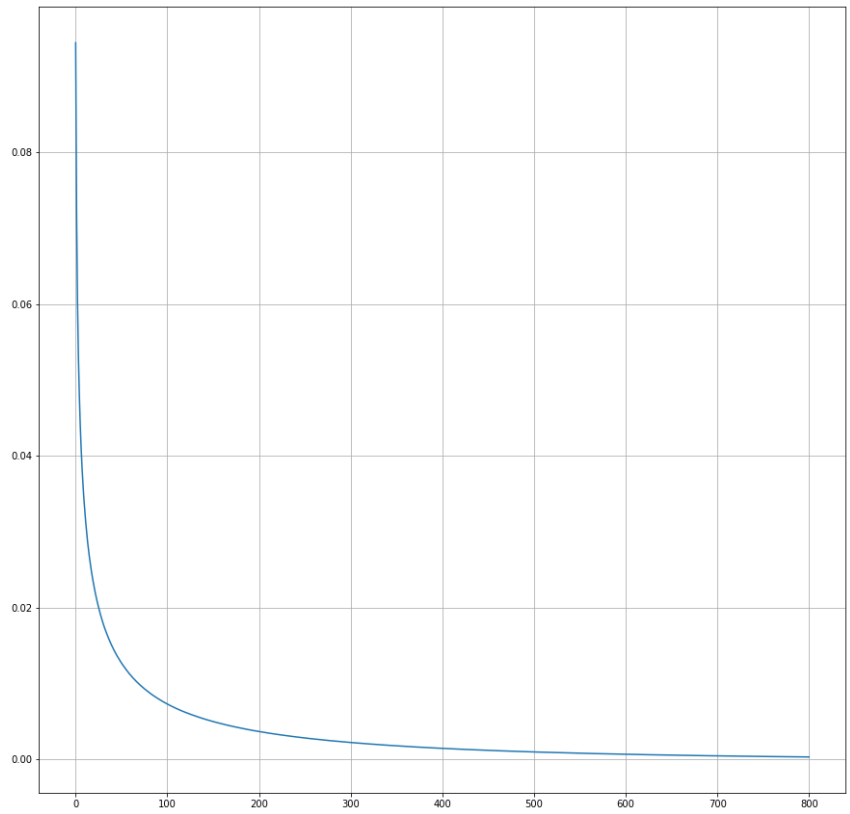

Figure 4.7: K-distance graph of the data.

Code Snippet 4.10: Applying the DBSCAN algorithm and saving the results to an output file.

```
# Select a specific timepoint for every patient and
    ↪ store in a dataframe
start = time.time()
extract = []
for file in list(glob.glob("data-pca/*.csv")):
    comp = pd.read_csv(file)
    extract.append(comp.values[300,:])
extract = pd.DataFrame(extract)
# Apply DBSCAN clustering on the selected timepoint
    ↪ for each
# patient with eps selected based on the k-distance
    ↪ graph.
clust = cluster.DBSCAN(eps=0.015, min_samples=10,
    ↪ metric='cosine')
q = clust.fit_predict(extract)
allPatients['clusters'] = q
allPatients.to_csv('results.csv')
end = time.time()
print(end-start) # Displays the total time to complete
    ↪  this task in seconds
```

Finally, given that there is a large amount of data that needs to be processed and an equally large amount that is produced in our attempt to cluster patients for

ARDS diagnosis, we proposed to run the code using parallel computing technology in order to gauge the potential speed-up that may be achieved when running such data- and compute-intensive applications on a supercomputer as opposed to running them on a local machine. Pre-processing the data and creating individual patient files, and loading the data to calculate distances are the two steps that require the most computing time and power and are therefore the parts of the data that can be parallelised so as to speed up the analysis process.

4.4.4. Assessing the Model

The final part to consider in the modelling phase of this project is the assessment of our parameter selection in terms of the clustering method and the number of cores in running the compute-intensive parts of the code.

When it comes to applying DBSCAN to our data, simply selecting values for *epsilon* and *minpts* may not mean that the best values were selected; therefore, we implement a grid search method to provide a list of the most effective combinations of the two parameters. This method, the code for which is provided below, was implemented both in a serial and a parallel environment and the speed-up was recorded. The results are also presented in a Figure 5.2 and will be further investigated in the evaluation chapter that will follow.

Code Snippet 4.11: Applying gridsearch in serial.

```
eps = list(range(5,26))
eps[:] = [x / 1000 for x in eps]
minpts = list(range(6,27))
grid = pd.DataFrame(index=minpts, columns=eps)
start = time.time()
extract = []
for file in list(glob.glob('/data-pca/*.csv')):
    comp = pd.read_csv(file)
    extract.append(comp.values[300,:])
extract = pd.DataFrame(extract)
for i in minpts:
    for j in eps:
# Apply DBSCAN clustering on the selected timepoint
    ↪ for each
# patient with eps selected based on the k-distance
    ↪ graph.
        clust = cluster.DBSCAN(eps=j, min_samples=i,
            ↪ metric='cosine')
        q = clust.fit_predict(extract)
        grid.values[i-6,int(j*1000-5)] = len(np.unique
            ↪ (q))-1
end = time.time()
print(end-start)
```

Running any snippet of code in parallel requires a different approach to solving the

problem and for that reason it is not often straightforward to select the number of cores to call on to run our code. In the case of the data preparation step, we judged that running the program on one node should be sufficient to achieve a significant speed-up, given that a node has 48 cores. On the other hand using 48 cores to run the gridsearch code below would be wasteful and counterproductive as in either dimension, the selected range is 21 steps, so we can break the first for loop and assign 21 cores with the task of running the remainder of the loop to complete the clustering implementation.

5. Evaluation

5.1. Experimental Setup

In this chapter we break down our results in order to draw our conclusions as to the effectiveness of the approaches that were implemented in our attempt to answer the research questions posed at the beginning of this report. By continuing with the CRISP-DM process with respect to model evaluation, we need a proper experimental setup over which to perform our evaluation. This section of the chapter analyses the computing and data infrastructures, as well as tools and methods that were used in the process and highlights the key aspects of the methods by which the results will be judged.

We opted to perform our benchmarking steps for speed-up related to large scale data-processing applications using a laptop computer and the JURON and DEEP-ER SDV HPC clusters. While not loosing sight of general usability elements with Jupyter, the idea is to show the speed-up that can be achieved with the proper architecture. Similarly, we needed to program certain elements of the application for running in parallel and making use of the available threads, which allowed us to gauge the speed-up of the data processing between parallel and serial programming on the same system. Tables 5.1 and 5.2 present the specifications of the two HPC clusters employed in the parallelisation approach.

Table 5.1: Technical specifications of each of the servers in the JURON cluster[2].

Processor	2 IBMPOWER8 processors (up to 4.023GHz, 2x 10 cores, 8 threads/core)
Memory	256GB DDR4 attached to CPU
GPU	4 NVIDIA Tesla P100 ("Pascal")
Memory	4x16 GByte HBM memory attached to GPU
Non-Volatile Memory	$\sim$ 1TB GST Ultrastar SN100 Series NVMe SSD
Operating System	CentOS 7.3, Linux kernel 3.10.0-514.21.1.el7

When it comes to software, the Jupyter Notebook was a major tool in writing, testing, running, and evaluating the code that would extract the data and perform

Table 5.2: Technical specifications of the DEEP-ER SDV Prototype[3].

Cluster	16 dual-socket Intel® Xeon® E5-2680v3 nodes
Cluster Memory	128GB DRAM, 400GB NVM
Booster	8 Adams Pass Intel Xeon Phi 7210 CPU
Booster Memory/Node	16GB on-package memory, 96GB DRAM, 200GB NVM
Storage	2 storage servers (spinning disks, 57TB)

the necessary pre-processing steps before clustering. For that reason, being able to use it in order to perform the same tasks that would be done locally, but on a large scale HPC cluster was extremely convenient and was instrumental in completing the research that we set out to perform. It is worth noting that, until the implementation of this tool into the parallel computing environment, all tasks to be performed on a supercomputer had to involve job scripts and required the user to run all of their code every time in one go. The benefit of using the Jupyter Notebook on HPC in this case, regardless of it still being a work in progress and only a recent endeavour, is the fact that it offers the added convenience of providing a graphical interface to visualise the code and the output at the same time and of running only part of the code as desired, thus allowing us to break down the tasks into parts and measure speed up over all of them. The Jupyter environment can be also easily extended for future studies in ARDS and is thus massively flexible for further research by medical doctors. Additionally, the research in this thesis contributed to overcoming issues with technical job scripts, ssh logins and keys, and other technical elements by focusing on a seamless approach that can also be used by non-technical-savvy medical experts.

5.2. Evaluating Relevance of Data and Code Provisioning for ARDS

Although our main goal in the data preparation and pre-processing phases was to produce samples with high relevance to the condition in question, we also aimed to reduce the datasets in size as much as possible. On the one hand this process would reduce noise from either irrelevant or inconsistently recorded attributes, while on the other hand it allowed a reduction in the density of the data manipulation job. Ultimately the selected attributes were judged as the most relevant to ARDS diagnosis based on the likelihood that a patient will have values recorded for these attributes in their record, but also based on our prior understanding of the condition and of physiology. Figure 5.1 below presents the different stages of the data, as well

as the number of attributes and individual patients at each step.

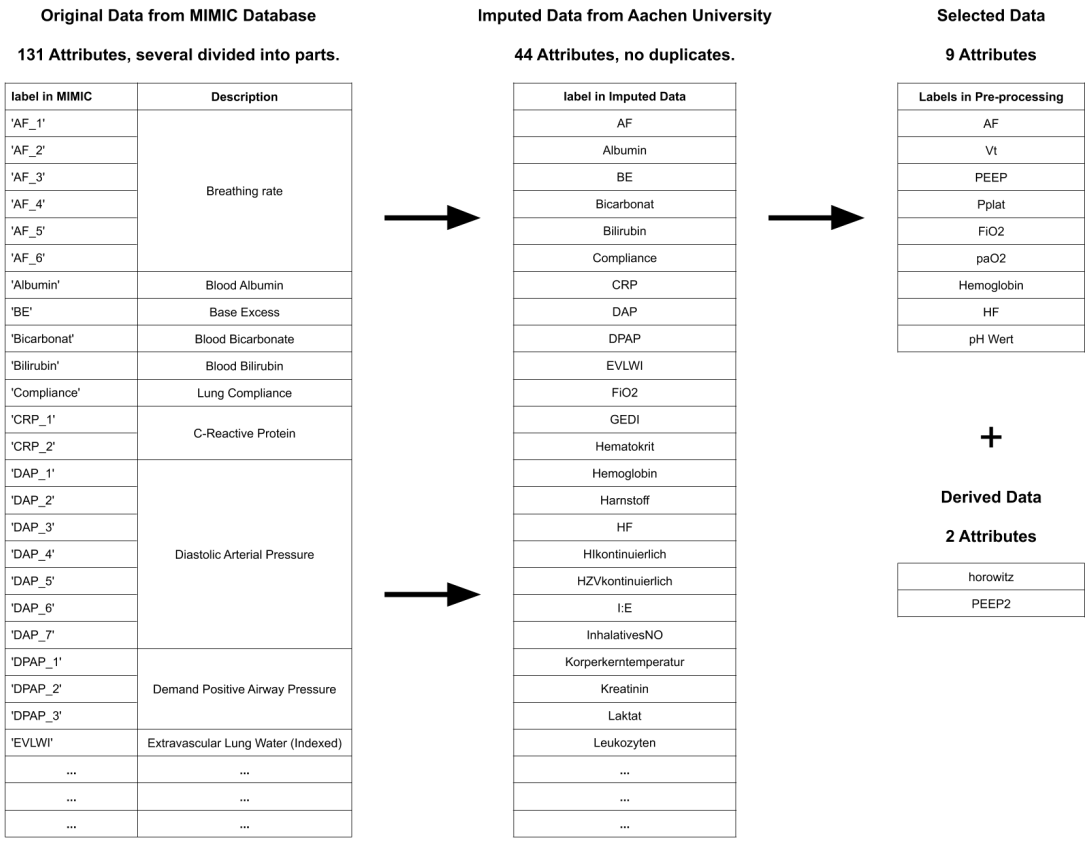

Figure 5.1: Graphical depiction of attribute selection at the major steps of the modelling process and pre-processed data provisioning pipeline for medical users.

The method used to manipulate the data and extract the relevant features is a major aspect of our research which makes our approach adaptable to the application in question. This means that clinicians and future researchers can seamlessly access it on the FZ-Juelich HPC and data infrastructures, after which they need only adapt the code to select the attributes relevant to the condition they intend to study and the software will output a number of patients with processed data and ready to undergo PCA and clustering. Access to the ARDS environment containing the data and code is given via easy-to-use Jupyter Notebooks on a simple login basis not requiring complex technical SSH keys nor deep technical knowledge of HPC systems. As a result, not only is relevance given for ARDS research, but also a usability evaluation reveals that it can be easily used by medical doctors in daily practice.

Finally, the fact that the code is portable and easily accessible enables porting

the Jupyter script to cloud infrastructures like the European Open Science Clouds (EOSC-Nordic) [28]

5.3. Pre-processing and Clustering Result Evaluation

One of the research questions of this thesis was to apply HPC to prepare for larger upcoming versions of the MIMIC database in general and to enable a speed-up and interactive clustering for users. In Section 1 of Chapter 4, our approach to modelling required comparing results obtained from a local machine with those obtained from a parallel application of the adapted code. For that purpose we attempted to run the code locally, though it became immediately obvious that the processing power and memory were not enough to complete the tasks required. For that reason, and even though our main focus is on the potential speed-up that parallelising may offer, we will also make note of the effect that running such data- and compute-intensive applications on a local machine can have.

In the data preparation and pre-processing phases, the most time and energy consuming part of the code involves reading from the imputed data files, extracting the relevant information and then storing in separate files in csv format. The speed-up for this process is presented in Table 5.3 below.

Table 5.3: Time required to complete the pre-processing section of the code on each type of architecture.

Local	JURON Serial	Parallel (48 cores)
1099,99 seconds	2370,72 seconds	68,32 seconds

It is worth mentioning that though the process took longer as a serial application on the JURON cluster than on the laptop, the latter was clearly beginning to exhibit performance issues including overheating and lag.

After applying PCA to the dataset and reducing it to five principal components, clustering was attempted and we immediately realised that the output would be greatly dependent on the selected parameters of *epsilon* and *minpts* as the results would vary greatly. Additionally our approach required comparison of the clustering results to the diagnosis obtained by using the Horowitz index as per the Berlin definition of ARDS. This process proved difficult and thus we decided to rely on the gridsearch method to find the best combination of the parameters. The steps that followed are discussed below where we discuss the parallel and serial implementations of gridsearch as well as the ultimate results.

5.4. Gridsearch Result Evaluation

Performing the gridsearch to find the most effective range of values for *minpts* and *epsilon* on a local machine required a total of 215,15 seconds (3,58 minutes) while in parallel it completed in 40,17 seconds. Another iteration of the code was prepared that allowed for scaling up the number of processors and increasing the range of *minpts* and *epsilon* which ran the code in 64,42 seconds. Finally, one early version of this code was inefficient and required a total of 3 hours to complete in serial on JURON while in parallel completed in 1093,04 seconds (18 minutes). The results of the output of the gridsearch code is presented below in Figure 5.2 with the most viable parameter highlighted in green, and acceptable values in yellow.

Epsilon

Minpts	0.005	0.006	0.007	0.008	0.009	0.01	0.011	0.012	0.013	0.014	0.015	0.016	0.017	0.018	0.019	0.02	0.021	0.022	0.023	0.024	0.025
6	4	8	16	16	18	20	27	28	35	44	52	58	60	69	66	63	64	68	78	67	72
7	1	5	8	8	11	15	14	13	16	22	22	20	28	36	42	35	38	36	37	44	46
8	0	1	4	5	5	9	8	12	17	11	12	15	16	14	21	28	28	29	24	26	33
9	0	0	2	3	5	5	5	8	11	9	7	6	8	9	9	11	12	18	22	18	19
10	0	0	1	3	4	3	3	4	5	7	5	5	7	6	8	12	13	14	14	15	14
11	0	0	0	1	2	3	3	4	6	6	5	4	3	4	4	6	7	11	11	11	9
12	0	0	0	0	1	3	2	3	4	4	3	5	5	4	5	4	5	5	7	9	8
13	0	0	0	0	0	2	3	2	3	3	4	3	3	4	3	5	4	2	2	6	8
14	0	0	0	0	0	2	2	2	2	2	2	3	4	4	3	3	4	4	3	3	4
15	0	0	0	0	0	0	2	2	1	2	3	3	3	3	4	3	3	4	3	2	3
16	0	0	0	0	0	0	0	2	1	2	2	4	3	4	4	3	3	3	3	3	2
17	0	0	0	0	0	0	0	1	1	2	2	3	4	3	4	3	2	2	2	3	2
18	0	0	0	0	0	0	0	0	1	1	2	2	3	4	3	3	2	2	3	2	3
19	0	0	0	0	0	0	0	0	0	1	1	2	2	3	4	3	2	2	2	2	2
20	0	0	0	0	0	0	0	0	0	1	1	2	2	2	4	3	3	2	2	2	2
21	0	0	0	0	0	0	0	0	0	1	1	1	1	2	2	2	2	2	2	2	2
22	0	0	0	0	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	2	2
23	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2	2
24	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	2	2	2	2
25	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2	2	2
26	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2	2

Figure 5.2: Different possible combinations of minpts and epsilon and the number of clusters produced

5.5. Review of Applied Processes

Using the gridsearch method described in Section 4, we select a few values for the DBSCAN parameters to attempt clustering the patients. In order to validate the effectiveness of clustering we implemented the Berlin definition of diagnosing ARDS and were able to highlight some patients as fulfilling the criteria for the condition. We implemented the code below in order to cross match patient IDs that appear in

both the main patient list and the Horowitz list. Of course, this evaluation is only for showing technical feasibility of the selected approach while more research from medical experts is needed to derive medically relevant results from clustering such as detecting early stage ARDS for example.

Code Snippet 5.1: Writing diagnosis results in the output file.

```
allPatients = pd.read_csv('allPatients.csv')
allPatients['ards'] = 0
allPatients = allPatients.set_index('PatientID')
ards = pd.read_csv('horowitz.csv')
for i in ards.id:
    allPatients.loc[i].ards = 1
allPatients.to_csv('allPatients.csv')
```

Similarly, when clustering is performed based on the chosen parameter values, the results are also written into the main patients list. Thus we were able to compare the results of clustering for the four parameter values highlighted in Figure 5.2 with those of the diagnosis process. Figure 5.3 presented below highlights the differences that small changes in the parameters would cause in how the data is understood. We underline the fact that in this case the clustering results for only 2 of the 5 principal components are presented. On the other hand, Figure 5.4 shows the diagnosis results based on the Horowitz index and highlights the fact that further work needs to be done from medical experts that take advantage of the results of this master thesis in order to improve the clustering results.

We note that, in general, an early detection of the ARDS is a hot topic in research concerning this condition, especially given the challenging situation that the time series data recordings of ICU patients are usually only often short or inconsistent. This in turn makes it difficult to detect patterns and highlights the potential benefit that unsupervised methods may to have in ARDS research today.

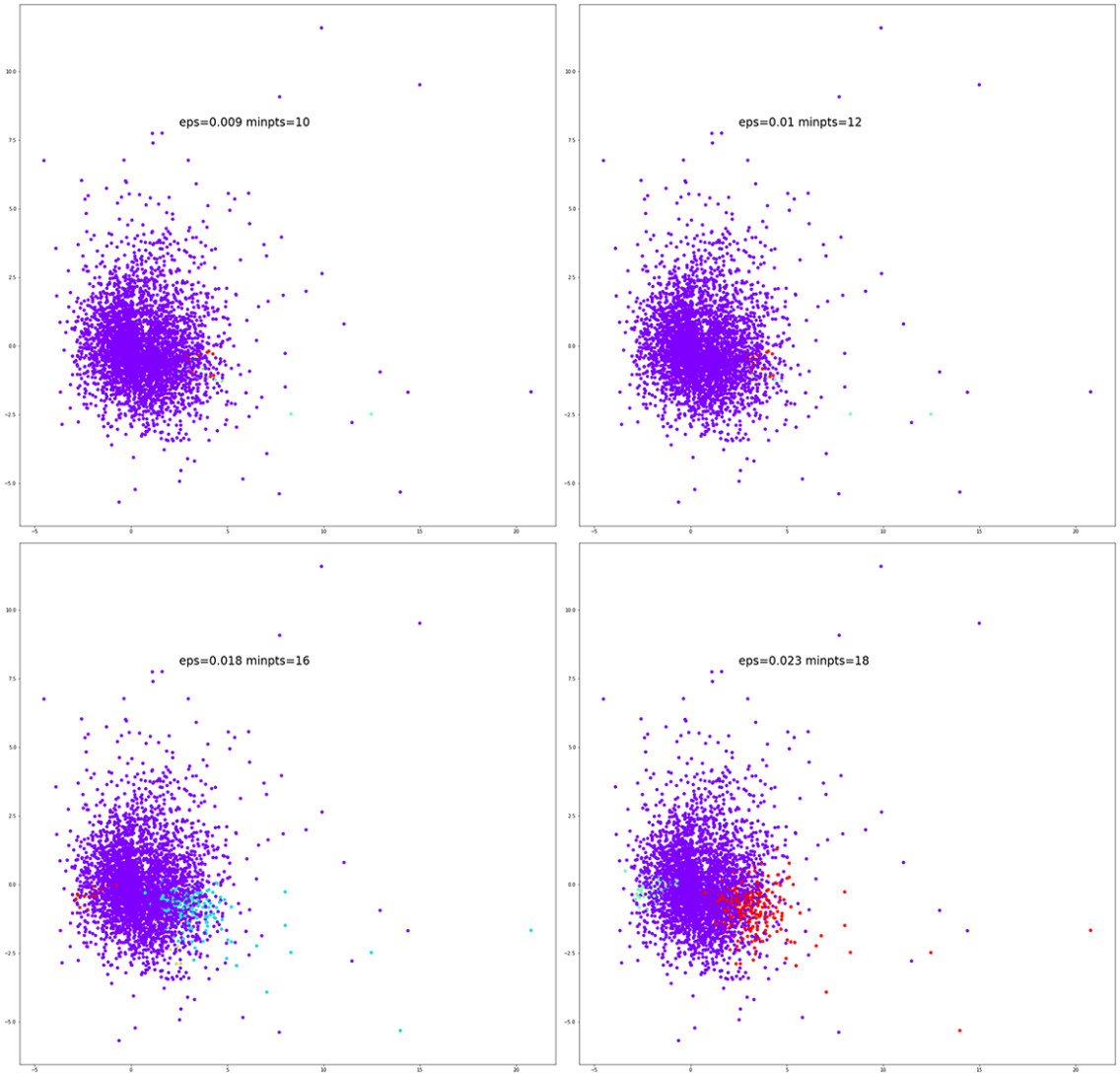

Figure 5.3: Four different clustering results with their respective parameters.

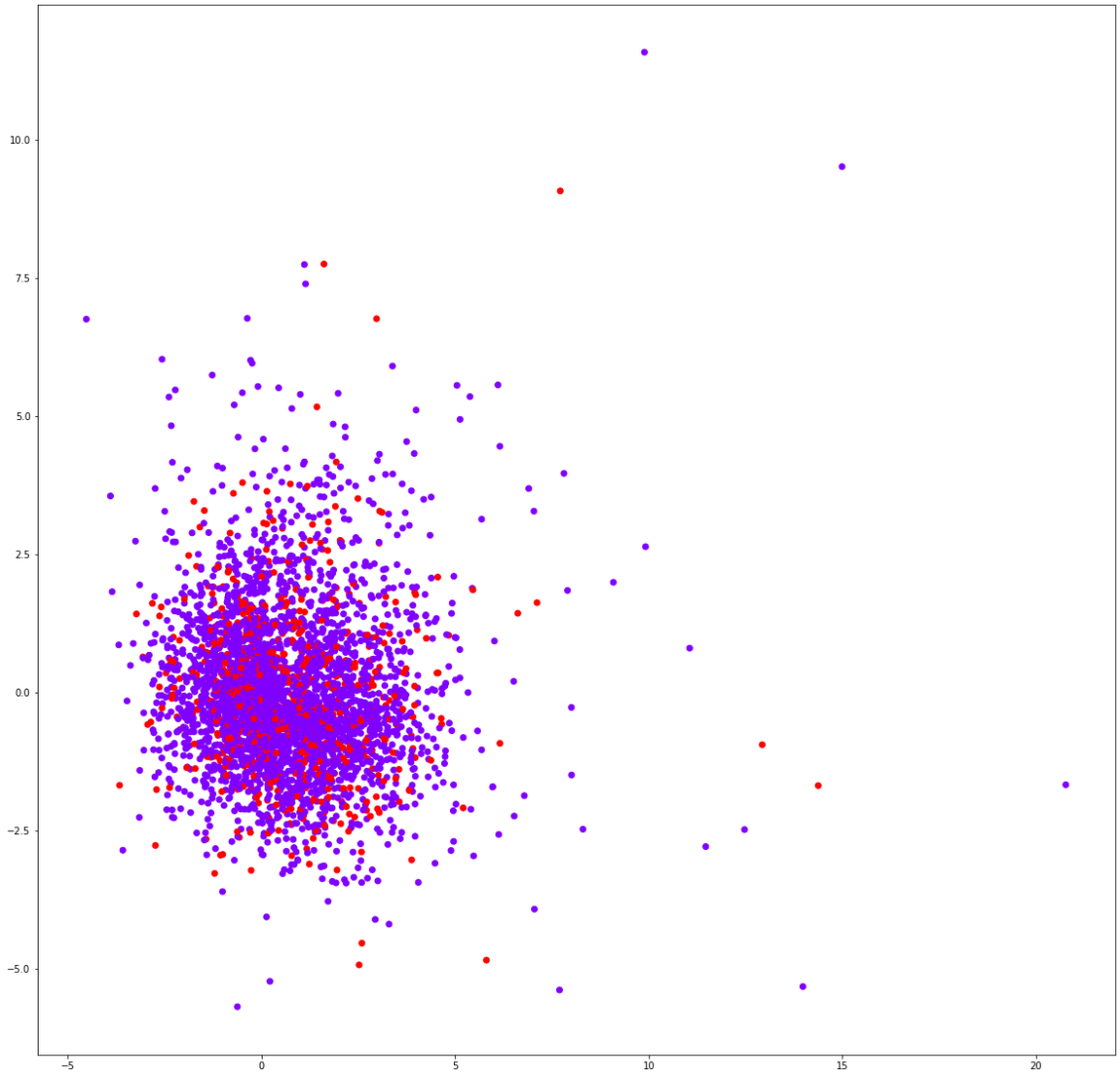

Figure 5.4: ARDS patients as diagnosed using the Horowitz index.

6. Conclusion

In this thesis we presented our work of preparing and exploring data by performing the analysis of large scale data from healthcare with a particular focus on ARDS patients, and the attempt of using state-of-the-art technology in HPC, unsupervised learning, and parallel programming in order to highlight features within the analysed data. In our approach, we developed a Python code that reads patient data from clinical database entries, selects a set of the most well-represented parameters related to the condition in question, in this case ARDS, and performs the necessary pre-processing steps to prepare this data for clustering.

Additionally, our aim was to highlight the potential benefits of using HPC technology and our knowledge in parallel programming techniques to accelerate the work and reduce the workload which ultimate proved successful given the speed-up that was observed in our applications. This was done using an HPC- and parallel computing-ready version of the Jupyter Notebook which recently became available on the JURON cluster of FZ-Juelich, and is the product of years of research towards making parallel computing more accessible despite of the fact that it is usually rather batch-driven instead of interactive. Given the fact that our approach takes advantage of the availability of Jupyter in a parallel environment, we believe our work opens up the path for clinicians to begin using this technology in their work, a step which would have been improbable with traditional HPC approaches.

Finally, we applied DBSCAN clustering in order to highlight structures in the data that could be used in the diagnosis of ARDS. Afterwards, we implemented a grid search algorithm that was able to output the most effective combinations of parameters for clustering. In parallel, and using the literature on ARDS diagnosis, we coded a program that listed the IDs of all patients that fit the criteria of the Berlin definition and attempted to use its result to evaluate our clustering technique. To conclude, the thesis results pave the way with a sound technical solution of making ARDS research faster yet flexible and usable for clinicians today.

As a product developed under the scope of several institutions, the code described in this project will be available to all participants in the SMITH project through access to the FZ-Juelich HPC and data infrastructure. One of the next steps of this thesis is to explore the possibility to make the code and data available in the Research Data Alliance (RDA) Interest Group (IG) on Big Data that is co-chaired

by FZ-Juelich. This is useful to expose the thesis findings to a broader international audience and goes far beyond the SMITH consortium.

The code is going to be edited, adapted, and used on a somewhat regular basis as part of the ASIC use case of the SMITH project as well as other potential uses where other researchers might see fit, therefore it will receive maintenance in a consistent manner. In light of the close cooperation of the University of Iceland and FZ-Juelich in several medical projects, the idea is to work together with Lanspítalinn in the context of ARDS in general and with the developed thesis approaches in particular.

Concerning future work in this field, our future research will most certainly revolve around discovering patterns of how the clustered patients are similar, what their identifying features are, and how this understanding can help improve future clustering results. Additionally, it would be interesting to focus further on the sizes of the created clusters, instead of only looking at the number, and finding ways to maximise that size. Similarly, given the easy access to HPC technology, it would be feasible to cluster at several points of the timeseries data in order to visualise how the patients are evolving towards or away from being diagnosed as ARDS.

On the other hand, we might also consider developing methods to parallelise different parts of the pre-processing phase, PCA implementation, and K-distance calculation. The code will constantly be revised so that it can be easily implemented where needed and adapted to the research questions posed in this thesis as the outcomes of this research with respect to pre-processing are also suitable for research in other conditions in the ICU. Furthermore, we see potential in adapting our processes for more advanced clinical applications and use cases with large scale datasets where traditional approaches have failed.

Bibliography

[1] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. Anthony Celi, and R. G. Mark, "MIMIC-III, a freely accessible critical care database," *Scientific Data*, vol. 3, no. 160035, May 2016. [Online]. Available: https://doi.org/10.1038/sdata.2016.35

[2] A. Eekhoff, B. Tweddell, and D. Pleiter, "BeeGFS Benchmarks on JURON," oct 2017. [Online]. Available: https://www.beegfs.io/docs/whitepapers/JURON_OpenPOWER_NVMe_by_ThinkParQ_FZ-Juelich.pdf

[3] "DEEPprojects prototypes," 2017. [Online]. Available: https://www.deep-projects.eu/hardware/prototypes.html

[4] A. Winter, S. Stäubert, D. Ammon, S. Aiche, O. Beyan, V. Bischoff, P. Daumke, S. Decker, G. Funkat, J. E. Gewehr, A. de Greiff, S. Haferkamp, U. Hahn, A. Henkel, T. Kirsten, T. Klöss, J. Lippert, M. Löbe, V. Lowitsch, O. Maassen, J. Maschmann, S. Meister, R. Mikolajczyk, M. Nüchter, M. W. Pletz, E. Rahm, M. Riedel, K. Saleh, A. Schuppert, S. Smers, A. Stollenwerk, S. Uhlig, T. Wendt, S. Zenker, W. Fleig, G. Marx, A. Scherag, and M. Löffler, "Smart medical information technology for healthcare (SMITH)," *Methods Inf Med*, vol. 27, no. 1, 2018. [Online]. Available: https://doi.org/10.3414/ME18-02-0004

[5] B. K. Beaulieu-Jones, P. Orzechowski, and J. H. Moore, "Mapping patient trajectories using longitudinal extraction and deep learning in the MIMIC-III critical care database," *bioRxiv*, aug 2017. [Online]. Available: https://doi.org/10.1101/177428

[6] "Report on the workshop on "High Performance Computing in Health Research"," 2014. [Online]. Available: https://ec.europa.eu/digital-single-market/en/news/workshop-high-performance-computing-hpc-health-research

[7] P. F. Ghalati, S. S. Samal, J. S. Bhat, R. Deisz, G. Marx, and A. Schuppert, "Critical transitions in intensive care units: A sepsis case study," *arXiv e-prints*, 02 2019. [Online]. Available: https://arxiv.org/pdf/1902.05764.pdf

[8] M. Komorowski, L. A. Celi, O. Badawi, A. Gordon, and A. Faisal, "The artificial intelligence clinician learns optimal treatment strategies for sepsis

in intensive care," *Nature Medicine*, vol. 24, 10 2018. [Online]. Available: https://doi.org/10.1038/s41591-018-0213-5

[9] V. Huddar, K. Desiraju, V. Rajan, S. Bhattacharya, S. Roy, and C. Reddy, "Predicting complications in critical care using heterogeneous clinical data," *IEEE Access*, vol. 4, pp. 1–1, 01 2016. [Online]. Available: https://doi.org/10.1109/ACCESS.2016.2618775

[10] M. Afshar, C. Joyce, A. Oakey, P. Formanek, P. Yang, M. Churpek, R. Cooper, S. Zelisko, R. Price, and D. Dligach, "A computable phenotype for acute respiratory distress syndrome using natural language processing and machine learning," *AMIA ... Annual Symposium proceedings. AMIA Symposium*, vol. 2018, pp. 157–165, 12 2018.

[11] A. Das, M. Haque, M. Chikhani, W. Wang, J. G. Hardman, and D. G. Bates, "Creating virtual ARDS patients," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Aug 2016, pp. 2729–2732. [Online]. Available: https://doi.org/10.1109/EMBC.2016.7591294

[12] D. G. Ashbaugh, D. B. Bigelow, T. L. Petty, and B. E. Levine, "Acute respiratory distress in adults," *The Lancet*, vol. 290, no. 7511, pp. 319 – 323, 1967, originally published as Volume 2, Issue 7511. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140673667901687

[13] P. Chapman, "The CRISP-DM user guide," 1999. [Online]. Available: https://s2.smu.edu/~mhd/8331f03/crisp.pdf

[14] The ARDS Definition Task Force, "Acute Respiratory Distress Syndrome: The Berlin Definition of ARDS," *JAMA*, vol. 307, no. 23, pp. 2526–2533, Jun 2012. [Online]. Available: https://doi.org/10.1001/jama.2012.5669

[15] J. Villar, J. Blanco, J. M. Añón, A. Santos-Bouza, L. Blanch, A. Ambrós, F. Gandía, D. Carriedo, F. Mosteiro, S. Basaldúa, R. L. Fernández, R. M. Kacmarek, and the ALIEN Network, "The ALIEN study: incidence and outcome of acute respiratory distress syndrome in the era of lung protective ventilation," *Intensive Care Medicine*, vol. 37, no. 12, pp. 1932–1941, Dec 2011. [Online]. Available: https://doi.org/10.1007/s00134-011-2380-4

[16] L. D. Bos, L. R. Schouten, L. A. van Vught, M. A. Wiewel, D. S. Y. Ong, O. Cremer, A. Artigas, I. Martin-Loeches, A. J. Hoogendijk, T. van der Poll, J. Horn, N. Juffermans, C. S. Calfee, and M. J. Schultz, "Identification and validation of distinct biological phenotypes in patients with acute respiratory distress syndrome by cluster analysis," *Thorax*, vol. 72, no. 10, pp. 876–883, 2017. [Online]. Available: https://thorax.bmj.com/content/72/10/876

[17] C. S. Calfee, K. Delucchi, P. E. Parsons, B. T. Thompson, L. B. Ware, M. A. Matthay, and the NHLBI ARDS Network, "Subphenotypes in acute respiratory distress syndrome: latent class analysis of data from two randomised controlled trials," *Respiratory Medicine*, vol. 2, no. 8, May 2015. [Online]. Available: https://doi.org/10.1016/S2213-2600(14)70097-9

[18] S. Shalev-Shwartz, S. Ben-David, and Cambridge University Press, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2015. [Online]. Available: https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf

[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, pp. 226–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=3001460.3001507

[20] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, *Introduction to Data Mining*. Pearson, 2005.

[21] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*. Chapman & Hall/CRC Computational Science, 2010.

[22] M. Götz, C. Bodenstein, and M. Riedel, "HPDBSCAN: Highly parallel DBSCAN," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15. New York, NY, USA: ACM, 2015, pp. 2:1–2:10. [Online]. Available: https://doi.org/10.1145/2834892.2834894

[23] "Project jupyter." [Online]. Available: https://jupyter.org/

[24] "Intellispace critical care and anesthesia." [Online]. Available: https://www.philips.co.uk/healthcare/product/HCNOCTN332/intellispace-critical-care-and-anesthesia

[25] M. Götz, M. Kononets, C. Bodenstein, M. Riedel, M. Book, and O. P. Palsson, "Automatic water mixing event identification in the koljö fjord observatory data," *International Journal of Data Science and Analytics*, vol. 7, no. 1, pp. 67–79, Feb 2019. [Online]. Available: https://doi.org/10.1007/s41060-018-0132-z

[26] "Python data analysis library." [Online]. Available: https://pandas.pydata.org/

[27] "feather-format." [Online]. Available: https://pypi.org/project/feather-format/

[28] "European open science cloud." [Online]. Available: https://www.eosc-portal.eu/

A. Appendix

Code Snippet A.1: The code that was developed for this thesis. This code is only meant for serial implementation.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import glob
import time
import horowitz_sample
from sklearn.neighbors import NearestNeighbors
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn import cluster
from scipy import signal
from scipy import stats


# Query the database and print Figure \ref{fig:mimic}
import helpers
cnx = helpers.getCnx("mimic_oracle")

data = pd.read_excel('mimic_factor.xlsx')
attids = data.itemid
labels = data.label

params_intersection = pd.DataFrame()
for i,attr in enumerate(attids):
    query = """(select distinct(subject_id) from labevents where
        ↪ itemid={0}) union (select distinct(subject_id) from
        ↪ chartevents where itemid={0})""".format(attr)
    query = query.replace('[', '(').replace(']', ')')
    data = pd.read_sql(query, cnx)
    params_intersection = pd.concat([params_intersection,data],
        ↪ ignore_index=True, axis=1)
    print(i)
params_intersection.columns = attids
params_intersection.columns = labels
# for each attribute ID list the number of admissions
width = 18
height = 14
plt.figure()
params_intersection.count().sort_values(ascending = False).plot.
    ↪ bar(figsize=(width,height), color=(0.2,0.4,0.6,0.6))
ax = plt.gca()
ax.set_ylabel("Number of admissions with values for an attributeid
    ↪ ", size = 20)
ax.set_title('Number of admissions for an attributeid, mimic',
    ↪ size = 24)
ax.set_xlabel('Attributeid in mimic', size = 24)
ax.grid(True)
ax.tick_params(axis='both', which='major', labelsize=16)
plt.tight_layout()
```

```
plt.show()

# Query imputed data and print Figure \ref{fig:imputed}
for file in list(glob.glob("mimic/*.feather")):
    ds = feather.read_dataframe(file)
    abc = ds.isna().all()
    abc ^= True
    abc = abc.rename(file.replace('mimic/data','').replace('.
        ↪ feather',''))
    out = out.append(abc.drop('charttime'))
out.sum(axis=0).sort_values(ascending = False).plot.bar(figsize=(
    ↪ width,height), color=(0.2,0.4,0.6,0.6))

# Prepare a list of patients fitting within the set window of
    ↪ samples
samples = 200
times = []
for file in sorted(list(glob.glob("mimic/*.feather"))):
    patient = feather.read_dataframe(file)
    if (4*samples < len(patient.charttime) < 10*samples):
        times.append([id])
times = pd.DataFrame(times)
times.columns = ['PatientID']
times.to_csv('allPatients.csv', index=False)

# Perform Pre-processing
number = pd.read_csv('allPatients.csv')
for patient in number.PatientID:
    dataarray = []
    ds = feather.read_dataframe("mimic/data{0}.feather".format(
        ↪ patient))
    data = ds[['charttime','AF','Vt','PEEP','Pplat','FiO2','paO2',
        ↪ 'Hämoglobin','HF','pH Wert']]
    data['horowitz'] = horowitz_sample.calculate_horowitz(data)
    X2 = data.iloc[:,1:].rolling(10).mean()
    X2 = X2.fillna(data.mean())
    X2 = X2.fillna(0)
    X2['PEEP2'] = X2.PEEP
    newdata = pd.DataFrame(StandardScaler().fit_transform(X2.drop(
        ↪ columns=['horowitz', 'PEEP2'])))
    newdata['horowitz'] = X2.horowitz
    newdata['PEEP2'] = X2.PEEP
    newdata.columns = X2.columns
    newdata.to_csv("data-averaged/{0}.csv".format(patient), index=
        ↪ False)

# Diagnose ARDS based on Horowitz index
avgs = []
for file in sorted(list(glob.glob("data-averaged/*.csv"))):
    patient = pd.read_csv(file)
    if (patient.horowitz[150] < 300 and patient.horowitz[400] <
        ↪ 300 and patient.PEEP2[400] >7):
        avgs.append(int(file.replace("data-averaged/",'').replace(
            ↪ '.csv','')))
avgs = pd.DataFrame(avgs)
avgs.columns = ['id']
avgs.to_csv('horowitz.csv', index=False)

# Load all averaged patient data and store them in a dataframe
allPatients = pd.read_csv('allPatients.csv')
patients = pd.DataFrame()
for id in allPatients.PatientID:
    patients = patients.append(pd.read_csv("data-averaged/{0}.csv"
        ↪ .format(id)))
# Apply PCA to all patients in order to find the number of
# principal components that most closely represents the data.
```

```
patients = patients.drop(columns=['horowitz','PEEP2'])
pca = PCA().fit(patients)
plt.figure(figsize=(20,20))
plt.grid(True)
plt.semilogy(pca.explained_variance_ratio_,'--o')
plt.semilogy(pca.explained_variance_ratio_.cumsum(),'--o')
plt.savefig('pca.png', bbox_inches = "tight")

# Apply PCA based on the number of components that was decided on
    ↪ from Figure \ref{fig:pca}.
dist = pd.DataFrame(index=range(0,2000))
for id in allPatients.PatientID:
    pat = pd.read_csv("data-averaged/{0}.csv".format(id))
    comp = pd.DataFrame(PCA(n_components=5).fit_transform(pat.drop
        ↪ (columns = ['horowitz','PEEP2'])))
# Save the data of each patient as a separate file after applying
    ↪ PCA (to reduce read-write cycles)
    comp.to_csv("data-pca/comp{0}.csv".format(id), index = False)
    nbrs = NearestNeighbors(n_neighbors=5,algorithm='brute',metric
        ↪ ='cosine').fit(comp.values[range(0,len(comp)),:])
    distance,indices = nbrs.kneighbors(comp)
    for i in range(0,len(distance)):
        distance[i] = np.average(distance[i,:])
        distance = pd.DataFrame(sorted(distance[:,0]))
        distance = distance.iloc[::-1]
        distance = distance.reset_index()
        distance = distance.drop(columns='index')
        dist = pd.concat([dist,distance],axis = 1)
dist.columns = allPatients.PatientID
# Compute the average of the k-distances and plot the k-distance
    ↪ graph from Figure \ref{fig:distance}
avg = []
for i in range(0,2000):
    avg.append(np.average(dist.values[i,:]))
plt.figure(figsize=(15,15))
plt.grid(True)
plt.plot(avg)
plt.savefig('distance.png', bbox_inches = "tight")

# Select a specific timepoint for every patient and store in a
    ↪ dataframe
start = time.time()
extract = []
for file in list(glob.glob("data-pca/*.csv")):
    comp = pd.read_csv(file)
    extract.append(comp.values[300,:])
extract = pd.DataFrame(extract)
# Apply DBSCAN clustering on the selected timepoint for each
# patient with eps selected based on the k-distance graph.
clust = cluster.DBSCAN(eps=0.015, min_samples=10, metric='cosine')
q = clust.fit_predict(extract)
allPatients['clusters'] = q
allPatients.to_csv('results.csv')
end = time.time()
print(end-start) # Displays the total time to complete this task
    ↪ in seconds

# Save ARDS Diagnosis results into main patient file
allPatients = pd.read_csv('allPatients.csv')
allPatients['ards'] = 0
allPatients = allPatients.set_index('PatientID')
ards = pd.read_csv('horowitz.csv')
for i in ards.id:
    allPatients.loc[i].ards = 1
allPatients.to_csv('allPatients.csv')
```

```
# Perform Gridsearch
eps = list(range(5,26))
eps[:] = [x / 1000 for x in eps]
minpts = list(range(6,27))
grid = pd.DataFrame(index=minpts, columns=eps)
start = time.time()
extract = []
for file in list(glob.glob('/data-pca/*.csv')):
    comp = pd.read_csv(file)
    extract.append(comp.values[300,:])
extract = pd.DataFrame(extract)
for i in minpts:
    for j in eps:
# Apply DBSCAN clustering on the selected timepoint for each
# patient with eps selected based on the k-distance graph.
        clust = cluster.DBSCAN(eps=j, min_samples=i, metric='
            ↪ cosine')
        q = clust.fit_predict(extract)
        grid.values[i-6,int(j*1000-5)] = len(np.unique(q))-1
end = time.time()
print(end-start)
```