applied sciences

MDPI

Article

Medical Data Transformations in Healthcare Systems with the Use of Natural Language Processing Algorithms

Aneta Poniszewska-Marańda [1,*], Elina Vynogradnyk [1] and Witold Marańda [2]

[1] Institute of Information Technology, Lodz University of Technology, 93-590 Lodz, Poland
[2] Department of Microelectronics and Computer Science, Lodz University of Technology, 93-005 Lodz, Poland
* Correspondence: aneta.poniszewska-maranda@p.lodz.pl

Abstract: Machine learning has only recently begun to see its application in medicine and is still facing quite a few challenges that prevent it from being more widely used. Problems such as high data dimensionality and the lack of a common data schema still remain relevant. It is worth examining the usage of machine learning in the context of healthcare and deploying selected machine learning algorithms on the problem of cardiovascular disease diagnosis. Cardiovascular diseases are currently the most common cause of death in the world. Many of them develop for a long time in an asymptomatic way, and when the first symptoms become visible, it is often too late to implement effective treatment. For this reason, it is important to carry out regular diagnostic tests that will allow you to detect a given disease at an early stage. It is then possible to implement appropriate treatment that will prevent the occurrence of an advanced form of the disease. While doing so, it attempts to analyse data from different sources and utilizing natural language processing to combat data heterogeneity. The paper assesses the efficiency of various approaches of machine learning (i.e., TR-SVM (Terminated Ramp–Support Vector Machine), TWNFI (Transductive Neuro-Fuzzy Inference), Naive Bayes) when applied in the healthcare field and proposes the solutions to the problem of plain text data transformation and data heterogeneity with the help of natural language processing. The algorithms used for diagnosis were implemented, tested and their performance compared, with their parameters also investigated, making it easier to choose an algorithm better suited for a specific case. Whereas TRSVM is better suited for smaller datasets with a high amount of dimensions, TWNFI performs better on larger ones and does not have the performance problems.

Keywords: machine learning; data management; natural language processing; healthcare systems

Citation: Poniszewska-Marańda, A.; Vynogradnyk, E.; Marańda, W. Medical Data Transformations in Healthcare Systems with the Use of Natural Language Processing Algorithms. *Appl. Sci.* **2023**, *13*, 682. https://doi.org/10.3390/app13020682

Academic Editor: Jan Egger

Received: 29 October 2022
Revised: 28 December 2022
Accepted: 29 December 2022
Published: 4 January 2023

1. Introduction

Despite the growing usage of machine learning in medical systems, there are multiple inconveniences that prevent its widespread usage. Different types of approaches may not perform as accurately as desired in the field of healthcare, or perform better or worse on different types of data [1–4]. In addition to that, existing solutions cannot always be applied efficiently due to the lack of common data schema for healthcare organizations to rely on [5–7]. This requires conversion between different data formats, and the rarity of solutions that automate those things make it a serious obstacle.

One of the more frequently used formats is plain text, in the form of patient notes, making it even more difficult for AI (Artificial Intelligence) and automatic health systems to make use of existing data without it being transformed manually. As a result, the task of automating the conversion of this data still remains important to this day [8]. Different possibilities of handling this task are still being explored, but the results of this research are both scary and largely remain hidden from the public eye (with larger companies and organizations preferring not to share their patents and studies). These factors cause the question of parsing patient notes into more computer-friendly formats to stay open for now.

Natural language processing might be a possible solution to the problem of heterogeneity if it could be forced to analyse column labels in the data structure itself. Assuming that familiar columns are named similarly, we could potentially detect features that are dependent on each other. As a continuation of said idea, language processing could help detect related features that are expressed in different units and help convert them [9,10].

Unit conversion within the context of heterogeneous datasets is yet another issue that requires exploration. While the extracted labels may contain clues as to what units were used for the individual dataset, this is not always the case. Due to this, this paper proposes a similarity-based approach towards unit conversion in features, where a feature conversion rate for new data is being approximated based on samples from existing data that are most similar in regard to features that we know to be equivalent [11,12].

Another problem with regard to medical data is that different diagnoses is the multicollinearity, which can distort the end result due to the classifiers receiving the same data twice. Difficulties may arise when merging datasets, with different default values being assigned to the samples in missing fields, making the features look more different than they are as a result of an assumption, which gives us a problem of finding default values that will distort the data the least [13–15].

The main contribution of the paper relates to the application of selected machine learning methods in healthcare for the transformation of medical data regarding the cardiovascular problem, with particular emphasis on the heterogeneity of medical data sources, which were addressed in [10,16–21] in a manner different from the approach described in our approach, i.e., as follows:

1. Assesses the efficiency of various approaches of machine learning when applied in healthcare field, with particular emphasis on the heterogeneity of medical data sources;
2. Uses existing Machine Learning classification algorithms with NLP for the medical data transformation on the problem of cardiovascular in healthcare systems;
3. Proposes the solutions to the problem of plain text data transformation and data heterogeneity with the help of natural language processing.

The paper is structured as follows: Section 2 presents the related work on machine learning for medical system. Section 3 gives the outline of health data formats in medical systems. Section 4 features the proposal in data transformation for tables and text to resolve incompatibilities between medical formats. Section 5 describes the created prototype of algorithms and data conversion while Section 6 deals with analysis of experiment results, and Section 7 concludes the presented studies.

2. Related Work on Machine Learning for Medical Systems

Machine learning these days is one of the many technologies employed in medicine and particularly medical data analysis. To this day, there are countless approaches of machine learning that are applied in various medical problems, whether it is hospital readmission, diagnosis or treatment plans, and even newer methods and applications are being developed as of now [22–25].

Machine learning is predicted to create processes performed by both humans and computers. In these instances, there is a need to achieve an optimal combination by leveraging human abilities of hypotheses generation, collaboration and AI systems oversight along with the AI abilities of analysis of large volumes of data, finding associations with predictive power, or optimization against a success criterion [26]. The basic used algorithms are Linear Regression, Logistic Regression, Decision Tree, Support Vector Machines (SVM), K-means, Random Forest (or Random Decision Forests), k-Nearest Neighbors algorithm (kNN), Dimensionality Reduction, Gradient Boosting and Extreme Gradient Boosting (XGBoost), variations of Neural Networks, Logistic Regression, and Naive Bayes Classifiers [27].

A support vector machine (SVM) is a supervised machine learning algorithm that classifies data into two categories. It uses a series of data already sorted into two classes and builds its model of it during training, drawing lines between established categories. The Naive Bayes classifier is a well-known approach in machine learning that has been

applied in healthcare for some time, is a modification of the normal Bayes classifier, and uses probability to solve the task of classification. Its basic principle is building probability distributions per trait for each class.

Despite the discussed benefits [28] of Fuzzy Inference Systems (FIS) and the related Adaptive-Network-based Fuzzy Inference System (ANFIS) in healthcare, in recent years, they are not as frequently mentioned in scientific literature. FIS are built upon ideas such as fuzzy sets, fuzzy control and fuzzy-logic systems and use fuzzy theory for mapping inputs to outputs. There are two known conventional types of FIS: Mamdani and Sugeno. The Mamdani fuzzy inference system was designed with the purpose of operating a steam engine and boiler combination, and it was meant to utilize sets of linguistic control rules obtained from the experienced human operators. The purpose of the Sugeno fuzzy inference system was to develop a systematic approach of fuzzy rules generation from a given input–output dataset.

Despite the evident benefits of the usage of machine learning systems in the field of healthcare, the technology is still facing many problems in practice, such as data heterogeneity and a lack of good enough hardware [29–32], as well as general human distrust towards machine assistance [24,33,34].

Data heterogeneity refers to the issue of coming from different, disparate data sources. The data collected from different resources come in a wide variety of data formats and types [35], and as a result, requires additional transformation before they can be utilized in any automation or calculations. Data heterogeneity has been a major problem in fields such as statistics, computer science, Internet of Things, and, essentially, machine learning, which includes, but is not limited to, machine learning in the context of healthcare. Another problem that is prevalent in machine learning-based health systems is the difference between various approaches in purpose, performance, and computational requirements. More computation-heavy techniques like deep learning may not be fit for organizations that do not have powerful hardware to run these programs efficiently [23,36,37].

Finally, there is a psychological problem that lies in a certain amount of mistrust towards machine learning in fields as crucial as medicine [38–41]. People outside the technological fields show reluctance to use methods that cannot be easily explained to them. With machine learning being a highly researched discipline in recent years, newer methods and approaches appear every day. Although a lot of them seem to draw from the more widely known approaches in the field, different kinds of modifications are being made to them to fit various applications of artificial intelligence. Healthcare is no exception in this regard, with different kinds of algorithms being made to tailor various problems medical professionals may be facing in their jobs [25,34]

With Support Vector Machines being a widely used technique due to them handling data with high dimensionality extremely well, newer algorithms are being proposed basing off that concept, with one of them being the Terminated Ramp–Support Vector Machine (TR–SVM) [42]. TR-SVM is a type of Support Vector Machines that is inspired by geometric considerations. The kernel is automatically built as a function of simple classifiers and generalized terminated ramp functions determined by the training data, which are obtained through a separation of oppositely labelled pairs of training points. Although not as popular in health systems as the SVMs, fuzzy inference systems seem to have been studied in the context of healthcare, with modifications being made to the conventional FIS structure specifically for their application in the field of healthcare.

The Transductive Neuro-Fuzzy Inference system (TWNFI) is a transductive neuro-fuzzy inference model with weighted data normalization, which, unlike the inductive learning and inference methods, predicts the model value for a single data sample, using the available information related to this point [28]. TWNFI seems to better fit the applications of machine learning in healthcare and medicine, where most of the focus is placed on the individual patient rather than the model. The logic can be applied to both the Mamdani and the Sugeno FIS, which makes it more adaptable.

Named Entity Recognition (NER) is one of the known approaches of semantic analysis. It is a technique that analyses text by automatically identifying named entities in it and classifying them into established categories, such as by names of people, organizations, locations, times, quantities, monetary values, percentages, etc. Two important NLP techniques are stemming and lemmatization, both of which aim to return the word to its "base" form by chopping off the endings and prefixes. Stemming methods approach this through heuristic processes that chop off both ends of the words, while lemmatization methods aim to use the vocabulary to correctly achieve that. Not so long ago, NLP and semantic analysis experienced several breakthroughs, one of them being word vectors. Word vectors aim to represent meanings of words with vectors of numbers. With the model being a set of vectors positioned in vector space, word vectors that share contexts are located closer to each other. The way the data are organized, operations with the word vectors such as subtraction and addition give us words with related meanings [25,43].

The study on ML techniques aiming to improve cardiovascular disease outcomes makes a mention of several systems that utilize SVMs, various types of neural networks, or a combination of both, with the latter currently being one of the most commonly used approaches in healthcare. The approach [44] proposes the usage of multiple fuzzy neural networks for creating models with sparse data, which switches between multiple networks and uses kernel regression to generate more training data for them. The authors of [45] focused the challenge of Combined Algorithm Selection and Hyper-parameter tuning (CASH) in relation to multiple techniques such as meta-learning methods, black box optimization and neural architecture search mentions solutions such as SmartML, ML-Plan, and Auto-Sklearn, which are designed to tackle the said problem, indicating an interest in the usage of multiple algorithms together. Another approach [43] reviews modern developments in NLP in the context of healthcare, showing that the recent medical NLP systems have been focused on techniques that rely on deeper understanding of the language and knowledge-oriented approaches, such as Metamap and Gene TUC. The paper also mentions NLP systems deploying several levels of analysis to extract information. However, current systems still face multiple challenges, such as incompatible vocabularies developing in the field of healthcare, the presence of spelling errors, and problems with parsing uncertainty and negation in text. The authors of [46] describe the usage of text mining for cases of respiratory diseases. It describes a more complex model where, alongside entity recognition, entity traits and the amount of information on them are stored. Machine learning has different approaches, applications and setbacks. However, in the context of healthcare systems, problems arise in its application due to data heterogeneity. This problem and its variations are examined more closely in the following part [47–51].

3. Health Data Formats in Medical Systems

The application of machine learning in healthcare very often greatly depends on the format in which medical data are supplied to the system. The success of a machine learning model may be determined by how relevant and how well the data are represented. As a result, different formats are used to tailor to specific purposes. The formats in which medical data are stored still remain very different and somewhat disorganized in the context of usage for machine learning. To combat heterogeneity, such data are normally integrated into existing dataset through different points, all for different types of data. In the context of health systems, the data that get used in machine learning, most health data used can be split into three major types: tables, images and plain text (patient records).

With tables being the most friendly format for machine learning out of three, it is no surprise that most systems, both in healthcare and in general, are accustomed to accepting this type of data. However, in healthcare, it is more of a "middleman" between machines and humans, and tables have to be filled with data from other (likely non-table) sources before they can be passed to an AI.

Images can present all kinds of scans and diagrams received from patient testing (CT, X-rays, ultrasound, etc.). Along with the text files, they are one of the primary sources

of information for physicians, since a lot of data about the patient's disease is usually discovered through tests. They are also the hardest to automate due to an individual solution having to be built for a specific purpose. They are mostly processed with neural networks due to Convolutional Neural Networks (CNN) excelling at image processing and their ability to learn smallest details.

Finally, plain text, aka patient records, is a format where the physician leaves most of their observations. It can be difficult to process for machine learning systems due to the fact that most of the records contain natural language. Transformation of such data from text to organized tables is a relatively new task in the machine learning field. Multiple approaches should be used and/or combined here to ensure that the medical data are processed and transformed accurately. However, each of the formats has its own problems one must be on the lookout for when designing a health system (Figure 1).

History	Date December 10, 2001
ID	Ms. XY is a 23-year-old single female college student
CC	"Headache"
HPI	Ms. XY presents with a one-month history of headaches. The headaches are located in a band like distribution around her forehead and occipital area.
Physical Examination	
Vital Signs	BP 110/68 P 72 RR 14 T 98.4F Wt 160lb Ht 68 in
CV	No jugular venous distension. No carotid bruits. Apical impulse mid-clavicular line, 5th intercostal space. Regular rhythm. Normal S1, S2 with physiological split. No S3 or S4. No murmur or rub. No femoral bruit. Pulses: dorsalis pedis, posterior tibialis, femoral and radial 2+. Abdomen Flat,

Figure 1. Example of medical record with plain text in it [52].

While the common data types used in healthcare being well-known for some time, there are numerous problems in the case of each when it comes to their usage in machine learning, due to the fact that even the data of the same format can still deal with the problem of heterogeneity.

As already mentioned, heterogeneity is the issue of data having high variability of types and formats. Ambiguity and low quality may be present due to missing values, high data redundancy, and untruthfulness. The integration of heterogeneous data remains a challenging task these days. There are the following types of data: heterogeneity [35]:

- Syntactic heterogeneity happens if data sources are expressed in different languages;
- Conceptual heterogeneity, alternatively known as semantic heterogeneity or logical mismatch, stands for differences in modelling the same domain of interest;
- Terminological heterogeneity represents differences in names and labels for the same entities when referring from different data sources;
- Semiotic heterogeneity, or pragmatic heterogeneity, denotes different interpretation of entities by people.

Tables are most likely to manifest at least one of these types of heterogeneity, due to usually being derivative from other data formats. Even similarly structured tables may be rendered unusable together due to frequent cases of terminological heterogeneity. In different sources, features may be named differently or with different units. An even bigger problem can be encountered in the form of semantic heterogeneity, which may arise as a result of different units being used to describe numerical data. Machine learning systems tend to not see the unit difference, leading to wildly inaccurate predictions.

Alternatively, semantic heterogeneity can occur when traits that are present in one table, but missing in another (such as gender, weight, or whether the patient had alcohol problems in the past), may be important to the existing model of the system with the previous trainings, but difficult to auto-assign, since a guessed value that is far from the truth could lead to further errors. Text processing may require very detailed descriptions to work properly. Since the patients may not always talk about every symptom or detail the

systems rely on, even the more accurate NLP systems may yield a result of low quality or ambiguous data.

Image-based medical data are, by their very nature, prone to conceptual heterogeneity, due to the fact that all kinds of scans and diagrams that are contained in it demand a separate algorithm for each kind of image. This makes it extremely difficult to find extra traits in images with the help of machine learning or to attempt classification of medical data images. There is no way to transform the data in this instance either, due to the differing nature of medical tests.

Another image-related problem that is worth noting is that, while images such as scans and diagrams can be mined for additional traits with solutions tailored specifically to particular problems, such a task is usually best trusted to convolutional neural networks and deep learning. Due to these approaches being extremely computation heavy, mining a lot of images for additional features can be too time-consuming to be reasonably useful. While classical image processing algorithms could be applied to them instead for faster computation, the implementation of such systems may be troublesome due to a huge amount of diversity in human anatomy and the number of different diseases that may cause the system to produce inaccurate results.

However, another complexity of image data processing in the context of medicine is that the best approaches require a lot of data samples to train on. As a result, any organization to use such a system would have to obtain countless samples from willing patients before such a system becomes efficient. As a result, a lot of potential users might decide an image-based healthcare system is not worth it due to the amount of investment required for it to become efficient.

The studies [53] apply Deterministic Sensitivity Analysis and Probabilistic Sensitivity Analysis to combat stochastic heterogeneity; however, the author notes that the complexity of the model largely depends on decisions made concerning the patient's treatment. Another approach [54] proposes an algorithm to help structure medical big data. It relies on the Apache HBase database and relies on transforming the data several times and applying text mining to it before transforming it to a common format; however, it makes very little mention of data formats that are not tables. The authors of [55] describe an approach to classify unlabelled data called inter-training that revolves around using multiple classifiers. The algorithm revolves around iteratively training multiple classifiers and adjusting them based on dataset members that have a higher amount of prediction agreement between classifiers [56–58].

All in all, regardless of the data type or format, heterogeneity remains a major problem in healthcare systems, preventing them from being used to their full efficiency. Due to different types of heterogeneity requiring different handling, automation of managing such data remains difficult to this day. In addition, data formats like images are difficult to work with in terms of universal solutions, due to images representing a wide variety of traits. To add to that, in a lot of cases, data have to be mined from text due to not all hospitals using automated or categorized solutions. These problems together make medical data difficult to operate with, especially in cases of patients who previously had differently formatted records.

4. Proposal for Data Transformation to Resolve Incompatibilities between Medical Formats

With different types of heterogeneity, it becomes increasingly difficult to manage patient data in health systems, especially in large amounts. As a result, different solutions are still being proposed for various applications, and ways to automate data transformation and normalization for usage in machine learning systems are still sought after. This section covers performance comparison of machine learning methods, as well as possible suggestions on such data transformation in the context of diagnosis with the application of NLP, as well as possible problems in data transformation requiring further research.

Many new algorithms have recently been developed, but have not seen a practical application yet. This experiment aims to put novel modifications of machine learning methods to use by evaluating them against a more "classic" approach, to see how different algorithms act in the case of data with missing values, as well as to help to test the NLP-based data transformation part. Particularly interesting in the context of heterogeneous data, however, are the novel modifications to fuzzy logic systems, with their fuzzy logic being interesting to observe within the context of incomplete or ambiguous samples. At the same time, Support Vector Machines have other draws when it comes to dealing with a large amount of data, making their performance in the context of an expanding dataset interesting.

For these reasons, the three algorithms chosen to be tested in this study are TWNFI, TR-SVM and Naive Bayes classifier, as a component to evaluate cardiovascular disease risk based on transformed data that were taken from various sources and automatically transformed [50,59,60].

TWNFI is also interesting to study in this context due to the individual quirks each algorithm possesses. TWNFI tailors its current model to the input, and it would take fewer steps to load during the feature changes too. To go with the experiment, a more conventional machine learning method—the Naive Bayes classifier—has also been chosen along with these.

The primary reason to evaluate cardiovascular disease risk is to identify those individuals who may be at an increased risk of developing this condition. Early identification of individuals at risk can lead to earlier interventions which can reduce the risk of developing cardiovascular disease, and can also help to identify those individuals who may need to make lifestyle changes in order to reduce their risk. Additionally, assessing cardiovascular disease risk can help to guide treatment decisions for those already diagnosed.

4.1. Data Transformation and Representation

Due to the aforementioned problem of data heterogeneity, the data must be transformed in different ways to combat it. However, with all the different formats available out there, it is impossible to write individual converters for absolutely every possible type and format. The proposed solution, in this case, is instead of converting formats from one another, the system could convert different types of data to a single format with the usage of NLP, all while attempting to make the method accept as many types of data as possible, which is later on used as a direct input for diagnosis, disease prediction or treatment plan assignment.

Due to a lot of machine learning algorithms dealing with vectors, the proposed schema is a vector representation of the data, with each sample of data presented as an N-dimensional vector, where N is the number of all accounted traits, which represent symptoms and patient characteristics ever stored in the system and can be either numerical or Boolean (represented as 1 and 0 in the implementation).

When the system is running, the data are processed according to its type, transformed with the help of semantic analysis (if structured differently), normalized and saved in a format accepted by processing system. Newer traits can be added by analysing different sources, text and images at the processing stage. Old and new data are then merged and updated automatically, with the missing values being replaced by either “0” (if Boolean) or the mean of all entries in the training set (if numerical). There is a problem arising from it, though, as too much analysis can result in data with a high amount of dimensions.

When a diagnosis algorithm of choice is used, distance can be evaluated between the samples with any possible distance score due to the vector representation of the data, making such a schema easy to use with most of the existing algorithms.

4.2. Feature Extraction

For formats other than tables, specific approaches have to be used to extract features. Due to patient records in the form of raw text being a common data format, NLP and

semantic analysis seem like logical choices to turn to when it comes to text analysis and feature extraction. When importing a table from a new source, it will be organized unlike our existing schema. However, due to a lot of labels often resembling natural language phrases or words, it might be possible to find a similar phrase to them and transform them into analysable phrases. In turn, these could provide hints about which features are equivalent and how data should be transformed (Figure 2).

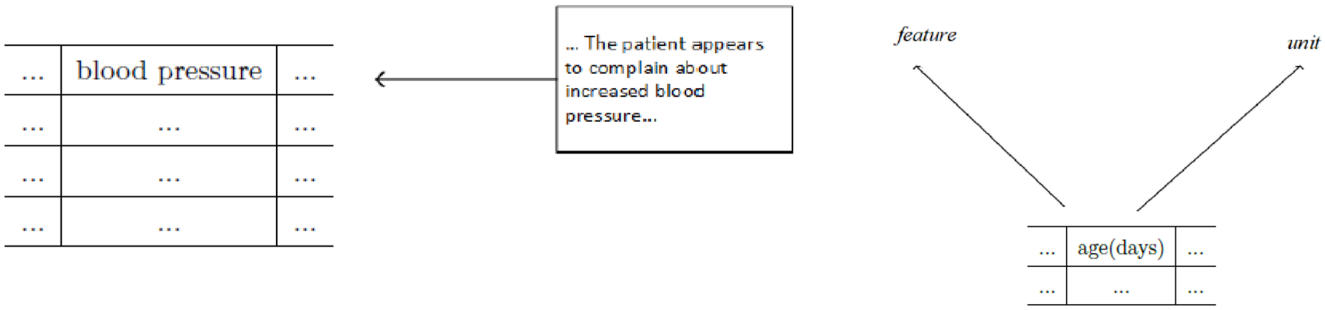

Figure 2. Transformation of text with patient notes to table with features represented as column (**left**) and application of NLP for analysis of table labels (**right**).

A similar logic could be applied to patient record-based data. As the raw text gets processed, it is interpreted by semantic analysis algorithms such as NER and word vectors, which help recognize entities and find related terms. As a possibility, the words for units measuring the same thing could be related in the model.

Feature extraction through NLP requires the application of multiple steps (Figure 3 [10,16–19,22]):

1. Find all entities in column names;
2. For each entity, check the existing labels for similarity using stemming and/or lemmatization. If they do, move on to step 3, otherwise assume there is no equivalent feature, fill the missing values with defaults and proceed to step 6;
3. If nearby words exist, perform semantic analysis. They might contain units. If the target dataset also has a unit specified in column name, utilize word vectors to establish a relation between them;
4. Assign a dependency between the two features, then proceed to step 6;
5. Repeat step 2 until all columns either have a dependency, or are marked to not have it;
6. Save the dependency list for possible future usage and merge the tables in accordance with it.

With the merge complete, the entire dataset can now be saved for later usage by the core system.

4.3. Unit Discrepancy Identification and Automatic Conversion

The most intuitive way to solve the discrepancy between different units in data would be to assign each possible unit certain limits and check at the beginning of each processing which one the first data rows fit most. However, that would be not the most reliable approach, since developers and medical professionals cannot always account for all the units that ever exist in the world and ensure their correct conversion. In addition, even then, the difference in some units may not always be clear to a machine.

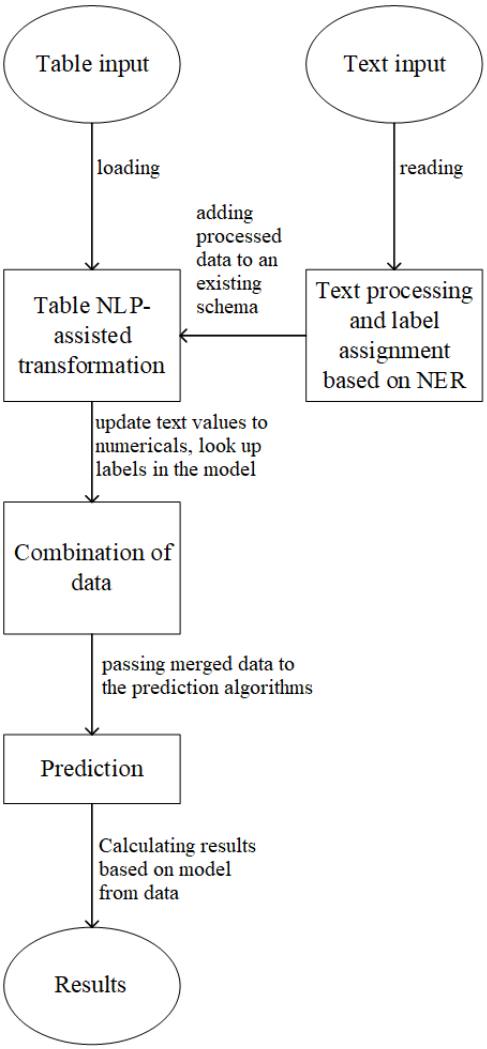

Figure 3. Scheme for an experimental unit conversion approach.

Alternatively, semantic analysis could be used on the labels to extract the possible units that may have been described in the label itself. This is not a very reliable method either, since column labels may not always indicate any units, leaving the system with no hints. In addition, still, we might have to deal with unfamiliar units.

For the biggest likelihood of catching such errors, multiple precautions, but better resolutions of this problem are still a point for future research.

4.4. Missing Values and Their Replacement

While adding entries from a differently organized table dataset to our own, it is inevitable that we are going to have to deal with missing values, due to the tables having different features. To avoid throwing away entire features (and potentially important components of our model), we have to assign the missing fields some default value instead.

As it was mentioned above, such default values are "0" for Boolean values and the mean of all entries for numerical (or otherwise represented by numerical values) features. This is based on the assumption that this should have the least effect on the variance value of a particular feature compared to other values. Unfortunately, the same cannot be achieved for Boolean values. Since not all systems may support uncertainties, one cannot make any assumptions about Boolean traits. As a result, it may be better to assume the trait is not present until there is proof that it is, which lead to "0" being the proposed value for the experiments.

The machine learning algorithms for missing data imputation, such as Mean/mode replacement, k-nearest neighbours, multiple imputation, linear regression, MICE (Multivariate Imputation by Chained Equations), decision tree regression, Bayesian network, k-means clustering, could be used here. These algorithms use statistical methods to fill in missing data in a dataset. This is achieved by using the existing data to learn the patterns and trends within the dataset, and then using those patterns and trends to estimate the missing values.

4.5. Multicollinearity and Its Detectability after Data Transformation

Data transformation and merging present problems with its detection. Let *x1*, *x2* be features that belong to datasets *D1* and *D2* respectively, and have no direct equivalents inside each other, but in reality present dependent features. As we attempt to merge them into the table *D3*, we will obtain a result similar to what is shown in Figure 4.

D_1	Past visits	...
...	3	...
...	2	...
...	0	...
...	...	...

D_2	Had cardio diseases in the past	...
...	1	...
...	0	...
...	1	...
...	...	...

⇓

D_3	Had cardio diseases in the past	Past visits	...
...	0	3	...
...	0	2	...
...	0	0	...
...	1	mean(x)	..
...	0	mean(x)	...
...	1	mean(x)	...

Figure 4. Two dependent features end up in one table, with filled values obscuring their dependency, making it a potential problem for the system.

The resulting table's filled spaces will obscure multicollinearity, especially if there are many samples to fill. It will prevent the conventional feature selection methods from discovering them and would decrease the quality of a built model if encountered. This paper does not provide a definite solution to this problem; however, it should be looked at in future research.

4.6. Experimental Approach to Unit Conversion

The proposed technique takes inspiration from similarity-based recommendation algorithm and draws from the idea that the "converted" value for the feature in a sample could be predicted by basing it off similar data in the dataset it is being merged with. The step-by-step explanation of this idea is as follows:

Let N and k are hyper-parameters, then:

1. Choose N samples from the dataset with the column that needs converting;
2. For N_i in N, choose k most similar samples in the dataset that is being merged with, according to features that either match or have already been transformed;
3. Take the values of the feature that needs conversion and calculate an average between them. This will be the predicted "converted" result;
4. Use the result from the previous step to and the value of N_i to calculate an estimated conversion rate;
5. Repeat steps 2–4 until all N samples have an estimated conversion rate;
6. Calculate an average of the estimates and assign it as a final conversion rate.

Such an approach is purely experimental. The initial suspicions are that it may give not very accurate results due to people being generally diverse. However, the idea that patients with similar disease history may share similar traits, and therefore a unit conversion rate could be accurately predicted based on similarity, was considered worth putting to a test. Another problem with such an approach would also be the computation time, due to the amount of time it would take for the program to go through a large dataset N times.

Finally, there is no guarantee that the datasets presented will contain features that share the same units. In this case, there may not be enough similar data for the method to base coefficients off, making this approach most effective when used in conjunction with the other methods. Furthermore, the units of matched columns may not always have linear dependency.

4.7. Boolean-Non-Boolean Discrepancy and Possible Fixes

It is possible that the values that need aggregating may come in both Boolean and non-Boolean formats with given data. In that case, putting them together may pose yet another difficult problem due to the system's uncertainty of how should different values should be translated from non-Boolean to Boolean and vice versa. While human medical experts may know the fitting values, they are not able to update the machine's functionality by themselves to get accustomed to the new value.

A simple fix for this may be normalizing the non-Boolean data. Unlike numeric qualities, the minimum and maximum of Boolean are always "0" and "1", respectively. As a variant, the normalized data may be assigned "0" if it is below average and "1" if it is above. This approach may, however, carry its own weakness, due to its assumption that the threshold determining the presence or absence of a certain trait may not always be the average.

4.8. Possible Application of Polynomial for Dependent Trait Calculation

While dependent variables normally reduce the usability of data in machine learning, it is possible that they could actually help establish values for the dependent variables that are not present in the data being aggregated by applying polynomial regression and calculating the exact dependency (Figure 5). This theory will be tested in the implemented prototype, to recognize dependent traits in the table labels (such as the obesity trait being calculated from BMI, and therefore related to height and width).

Unfortunately, this theory was not tested in the implemented prototype, due to a lack of available word vector models that would recognize dependent traits in the table labels (such as the obesity trait being calculated from BMI, and therefore related to height and width). This could still serve, however, as further research subject, or tested independently from these presented works.

D_1	weight(kg)	height(cm)	BMI	...
...	70	170	24.22	...
...	75	165	27.55	...
...	70	186	20.23	...
...	...	...	...	...

D_1	weight(kg)	height(cm)	...
...	80	180	...
...	75	170	...
...	90	190	...
...	...	...	...

⇓

D_1	weight(kg)	height(cm)	BMI	...
...	80	180	24.69	...
...	75	170	25.95	...
...	90	190	24.93	...
...	...	...	...	...

Figure 5. Three features end up in one table, with application of polynomials for dependent trait calculation.

4.9. NLP Parser Details

Due to the scarcity of trained medical NER models, a decision was made to move away from simply analysing the text with NER, in favour of deploying several approaches at once. The NLP implemented in the prototype in particular combines NER, word vectors and rule-based matching in an attempt to maximize the recognition of important details, with the techniques being applied in the following order:

Let there be hyperparameters S that indicate how synonyms for the program load, coefficients $k_1, k_2, \ldots, k_N$, lower and upper thresholds $l_1, l_2, \ldots, l_N$ and $u_1, u_2, \ldots, u_N$, where $i = 1, 2, \ldots, N$ is the unique number for each NLP model, then:

1. Load the program's schema stored in file and NLP models, as well as second instances of these models with NER disabled. Look through the feature labels and their units in the schema;
2. If all labels have been processed, proceed to *step 5*. Otherwise, load the next label and proceed to *step 3*;
3. Check the label name and assigned unit in the file. If a recognized feature is a collection or a Boolean, load S label synonyms from the word vector model, else check a numerical value's unit. If there is one, select the names of all the units with compatible dimensionality;
4. Create extra entities and entity recognition rules based on the findings in *step 3*, replace the missing NER component in the previously loaded model instances with the entity rulers based on our newly created rules;
5. Perform NER, group each recognized phrase by entity. Have the models that discovered respective entities compare it to the existing schema labels. If similarity reaches the given lower threshold value l_i, its values are passed on for further selection as potential entries to fill the missing values;
6. Selected values are now sorted by their similarity score to a certain feature label. The similarity score is also modified by a coefficient k_i that is unique to the model that discovered the value. Values with scores of "1" and above are automatically assigned to the features. If there is more than one value like that, the one with maximum score is always assigned;
7. If all the labels have assigned values, proceed to *step 10*. Otherwise, proceed to *step 9*;
8. Process values with the score below "1" by comparing the highest score among a label's assigned values to the upper threshold numbers. If the score is greater or equal than u_i, where i is the number of the model that discovered the value, proceed to *step 9*. Otherwise, repeat with the next label;
9. Parse the selected value depending on whether it is a numerical value, Boolean, or a connection of values. Numerical values (both with assigned units and without them)

are processed using a special function that parses the string, collections are processed by finding a collection value most similar to the discovered one, and Booleans are parsed by discovering negation in the sentences;
10. Assign the parsed result to the feature currently being processed;
11. Return to *step 8* if there are more feature labels to be analysed, otherwise finish parsing.

4.10. On the Possibility of a Medical Image Classifier

As mentioned above, medical image data are difficult to work with due to its diversity. A separate system has to be made for processing different types of these images. However, a degree of automation could be achieved if the data could be organized into categories if a classifier was created to automatically sort images and send them to the appropriate system. It is difficult to speak of the intricacies of such a task without knowing the intricacies of human anatomy and medicine in general, and the many specifics of particular human organs will likely take numerous amounts of data, but if achieved, it could help combat the problem of heterogeneity in the context of healthcare greatly. It could be a future research direction.

To conclude, various problems that arise with merging and transforming data require different types of processing for the results to be usable. The solutions described in this section aim to automate this process as much as possible by analysing table column labels and their contents, as well as assigning default values to missing data. However, generalizations made in such cases may come at the cost of prediction accuracy. Whether the end results of such automated manipulations are acceptable or not is to be determined in the following sections.

5. Prototype of Algorithms and Data Conversion

This section presents the specifics pertaining to the implementation of the prototype that makes use of the ideas described in the previous section. The prototype runs on Python 3.7. It was chosen due to the language's simple syntax and support of many scientific computation libraries. The following libraries were used in the implementation: *SciKit-learn* (for various utilities and an implementation of Naive Bayes classifier) [61], *Pandas* (for random sample selection and other various utilities [62], *NumPy* (for general-purpose calculation functions and faster two-dimensional arrays) [63], *spaCy* (for natural language processing) [64–66], *statsmodels* (to detect multicollinearity) [67], *simpy* (additional mathematical functions). The experiments were performed in the Standard, medium-size performance system.

5.1. Datasets for Testing the Methods

The datasets used in this research were two cardiovascular disease datasets. The first was a dataset with 11 features and a binary target ("Presence or absence of cardiovascular disease") by Svetlana Ulianova, and was provided in a *csv* table [68]. The features of this dataset are as follows:

- Age–Objective Feature–age–int (days);
- Height–Objective Feature–height–int (cm);
- Weight–Objective Feature–weight–float (kg);
- Gender–Objective Feature–gender–categorical code;
- Systolic blood pressure–Examination Feature–ap_hi–int;
- Diastolic blood pressure–Examination Feature–ap_lo–int;
- Cholesterol–Examination Feature–cholesterol–1: normal, 2: above normal, 3: well above normal;
- Glucose–Examination Feature–gluc–1: normal, 2: above normal, 3: well above normal;
- Smoking–Subjective Feature–smoke–binary;
- Alcohol intake–Subjective Feature–alco–binary;
- Physical activity–Subjective Feature–active–binary;
- Presence or absence of cardiovascular disease–Target Variable–cardio–binary.

The second dataset was made by Yassine Hamdaoui, had nine features and a binary target, only featured male patients and was provided in a *txt* file [69]. The features of this dataset are as follows:

- sbp–systolic blood pressure;
- tobacco–cumulative tobacco (kg);
- ldl–low density lipoprotein cholesterol;
- adiposity–a numeric vector;
- famhist–family history of heart disease, a factor with levels "Absent" and "Present";
- typea–type-A behaviour;
- obesity–a numeric vector;
- alcohol–current alcohol consumption;
- age–age at onset;
- chd–response, coronary heart disease.

Both of the sets were obtained on Kaggle. The structure of the features resembles the format described in previous section, making it easy to apply for testing, but both datasets have differently named columns and express some properties differently, making them viable for testing heterogeneity solutions. The latter set had to be converted to csv due to time constraints that prevented additional automation in this instance. For the purposes of testing, the word vector-based label parsing and giving the similarity-based algorithm more data, variants of both datasets with differently named labels were created and used alongside the originals. In addition to that, procedurally generated spiral datasets, as well as scikit-learn's breast cancer dataset, were used.

The data are normalized after import. The normalization values are stored during execution in case we ever need to return the data to its original look. During the testing, it was revealed that the first dataset contains outlier entries that contain unrealistic values of features, distorting the result of the algorithms. To solve this, a separate module was implemented specifically to fix or delete entries that are outside previously passed parameter boundaries.

The text entries are written based on the samples inside the aforementioned datasets, due to the difficulty of obtaining actual patient records connected with data confidentiality. It should be noted that, due to our inability to obtain an actual data, the results with the text parser are an approximation of how it would actually work. For the NLP part, a dataset of medical abbreviations is used to help the program create custom NER rules based on the data columns, which are discussed further.

5.2. Implementation Structure

The program implements several algorithms to test their performance in comparison with a relatively well-known method, the Naive Bayes classifier. The root directory of the implementation is organized to separate different Python files by function. The files `launcher.py` and `launcher2.py` are located in the root directory, with the former containing functions that are used to measure average execution time and a mean square error for a certain amount of experiments) and call methods implemented in all the other files. This is the file that should be used to run the program, and the latter containing functions used to generate synthetic datasets and help pick parameters for TRSVM.

Alongside `launcher.py`, there is a core subdirectory containing all the implemented methods, as well as a file with necessary utility functions. The following modules inside the `core` folder separate the implemented methods by type and purpose: `fuzz1.py` (contains the code for fuzzy inference systems, such as basic Type 1 Mamdani and Sugeno systems, as well as TWNFI), `svm.py` (implements TR-SVM), `bayes.py` (used to call Skikit-learn's implementation of Gaussian Naive Bayes classifier), `text.py` (implements the data transformation procedures related to NLP), `similarity.py` (implements the experimental method discussed in previous section), `tests.py`, `utils.py`, `multicollinearity.py` (implements multicollinearity detection), `anomalies.py`, `conversion.py` (implements additional conversion methods that utilize NLP on feature labels), `clustering.py` (implements the ECM

clustering method TWNFI uses for creating local models), `optimization.py` (implements the steepest descent method). The modules are all imported to `launcher.py`, with their functions used in there.

5.3. Implementation Details and Problems

Before proceeding to testing, a few implementation-specific things have to be addressed. Due to the general unavailability of medical patient notes to train a new model or customize an existing one, the implementation instead uses several NER models, examining the entities recognized by each of them. As a result, the text parsing part of the prototype combines multiple NLP approaches and is tested on what is an approximation of the real data.

Due to the general unavailability of medical patient notes to train a new model or customize an existing one, the implementation instead uses several NER models, examining the entities recognized by each of them. As a result, the text parsing part of the prototype is not perfect, but instead combines multiple NLP approaches and is tested on what is an approximation of the real data.

The unknown data are represented by "-1" during the merging as a way of expressing uncertainty. The value of "-1" allows the program to quickly separate the uncertain values from the rest of the data (which is normalized in the range of [0, 1]), simplifying the implementation of the methods we need to process data, particularly unit conversion, and in case unknown values remain in the sample after the conversion, they get replaced with the values mentioned in the previous section before the program proceeds to check the data for multicollinearity.

Multicollinearity in the prototype is detected using *Variance Inflation Factor (VIF)*, the function for which is provided by the *statsmodels* library. There may be more efficient ways to detect it, but as of the time of the writing of this paper, VIF seems to be a pretty popular method. Having finally checked everything, the data are passed to one of the methods.

The experiments for diagnosis methods are performed by calling the appropriate function within the `tests.py` module from `TestLauncher` function, which then splits off a subset according to one of the batch sizes mentioned in previous section, and trains the method with it. Afterwards, a sample is randomly picked from the data that was not used for training and its value is predicted. The experiments are repeated a given number of times, which can be specified as a parameter for `TestLauncher`. Meanwhile, unit conversion and NLP have their own experiment launcher within the `tests.py` module that needs to be called directly.

Due to the overall computational complexity of the TR-SVM method, an implementation of the pruning heuristic described in [42] alongside it was necessary. It is a heuristic for reducing the number of terminated ramp functions without relevant loss of information. Let $f_t = (K_t(x_i))_{i=1,\ldots,N}$ be a vector representing the feature induced by the terminated ramp function K_t and s_t is the fraction of components of f_t equal to 1 or -1:

$$S_t = \frac{Card\{i|K_t(x_i) = 1\}}{N}$$

The s_t can be used as an indicator of the relevance of feature.

The first set of the TRSVM experiments uses the hyperparameter C = 0.01, which was experimentally determined to be more optimal, and calculates S in accordance with the formula:

$$S = max\left(0, 1 - \frac{200}{n}\right)$$

to decrease the computational time on a higher amount of samples (Figure 6).

Output for the algorithm tests:

```
Dataset size: 250
S: 0.8
Fitting
Predicting
Accuracy percentage:
76.0
Fitting time:
51150.4616
Prediction time:
14569.420000000007
Mean square error:
0.24
```

Output for the parsing test:

```
Predicted sample: [23010.75, 1, 155.0, 56.0, 120.0, 80.0, 0, 1, 0, 0, -1]
Actual sample: [23191, 1, 155, 56, 120, 80, 0, 0, 0, 0, 1]
Computational time: 2857.5725000000034
Accuracy per sample: 81.81818181818183

Predicted sample: [21184.5, 2, 166.0, 101.0, 140.0, 90.0, 1, 1, 0, 1, 0]
Actual sample: [21270, 1, 166, 101, 140, 90, 1, 0, 0, 0, 1]
Computational time: 2708.042000000006
Accuracy per sample: 63.63636363636363

Predicted sample: [23010.75, 1, 164.0, 82.0, 130.0, 70.0, 1, 0, 0, 0, 0]
Actual sample: [23343, 1, 164, 82, 130, 70, 1, 0, 0, 0, 1]
Computational time: 2791.6984999999954
Accuracy per sample: 90.9090909090909
Total accuracy:
76.36363636363637
```

Output for the conversion test:

```
Coefficients for years, inches and pounds:
[367.0819398146915, 2.545209457986907, 0.4703396305808311]
Computational time
139738.50029999999
```

Figure 6. Output of implemented prototype to display results of NLP in medical data transformations.

5.4. NLP Parser Details

Due to the scarcity of trained medical NER models, a decision was made to move away from simply analysing the text with NER, in favour of deploying several approaches at once. The NLP implemented in the prototype in particular combines NER, word vectors and rule-based matching in an attempt to maximize the recognition of important details, with the techniques being applied in the following order:

Let there are hyperparameters S that indicates how synonyms for the program to load, coefficients $k_1, k_2, \ldots, k_N$, lower and upper thresholds $l_1, l_2, \ldots, l_N$ and $u_1, u_2, \ldots, u_N$, where $i = 1, 2, \ldots, N$ is the unique number for each NLP model, then:

1. Load the program's schema stored in file and NLP models, as well as second instances of these models with NER disabled. Look through the feature labels and their units in the schema.
2. If all labels have been processed, proceed to *step 5*. Otherwise, load the next label and proceed to *step 3*.
3. Check the label name and assigned unit in the file. If a recognized feature is a collection or a Boolean, load S label synonyms from the word vector model, else check a numerical value's unit. If there is one, select the names of all the units with compatible dimensionality.
4. Create extra entities and entity recognition rules based on the findings in *step 3*, replace the missing NER component in the previously loaded model instances with the entity rulers based on our newly created rules.
5. Perform NER, group each recognized phrase by entity. Have the models that discovered respective entities compare it to the existing schema labels. If similarity reaches the given lower threshold value l_i, it's values are passed on for further selection as potential entries to fill the missing values.
6. Selected values are now sorted by their similarity score to a certain feature label. The similarity score is also modified by a coefficient k_i that is unique to the model that discovered the value. Values with scores of "1" and above are automatically assigned to the features. If there is more than one value like that, the one with maximum score is always assigned.

7. If all the labels have assigned values, proceed to *step 10*. Otherwise, proceed to *step 9*.
8. Process values with the score below "1" by comparing the highest score among a label's assigned values to the upper threshold numbers. If the score is greater or equal than u_i, where i is the number of the model that discovered the value, proceed to *step 9*. Otherwise repeat with the next label.
9. Parse the selected value depending on whether it is a numerical value, Boolean, or a connection of values. Numerical values (both with assigned units and without them) are processed using a special function that parses the string, collections are processed by finding a collection value most similar to the discovered one, and Booleans are parsed by discovering negation in the sentences.
10. Assign the parsed result to the feature currently being processed.
11. Return to *step 8* if there are more feature labels to be analysed, otherwise finish parsing.

5.5. Testing Methods

To test the prototype's diagnosis methods on tables, we have used different subsets of the datasets mentioned at the beginning of this section. For these experiments, we have picked subsets of 100, 250, 500, 1000, 2500, 5000, 10,000 samples, as well as the entire set, i.e., 70,000 records of patients data for the first set and 20,000 records of patients data for the second set. The results are compared between the four approaches (i.e., TWNFI, TR-SVM and Naive Bayes classifier, Gaussian and Complement) by accuracy percentage, average fitting time, average prediction time, total time and mean square error. In TWNFI's case, additional testing was achieved to see how a number of local model samples would affect the results, since it appears to affect both the quality of the model and the computational time.

In addition to the aforementioned tests, the TRSVM was tested on a program-generated two-feature dataset representing spirals, to compare the algorithm's performance on synthetic datasets against natural ones, as well as dimensionality and its effect on prediction accuracy.

For the NLP part, the entire batch of written notes was used, due to the scarcity of such data. Instead, for testing purposes, a sample has been written by hand based on a random sample from either of the datasets, which are subjected to text parsing. The resulting data are afterwards compared to the pre-transformation data, and an accuracy percentage is calculated based on all samples. To test the unit conversion approach, a different amount of similar samples for comparison was taken to observe how their number affected the overall prediction of missing features. The same applies to the number of runs for which the similarity-based unit conversion was run.

Another experiment has an existing dataset split in two, and a random trait of it converted into new units. That way, the program is forced to approximate a conversion rate using the experimental method described in the previous section. After doing this, a random sample is picked from one of them and is subjected to this method with various numbers of similar samples as the parameter. This allows for showing how the number of similar samples may affect the performance of a similarity-based method.

Finally, we merge two tables into one while utilizing the methods mentioned in this paper, and have the machine run the implemented diagnosis methods with the merged table. The accuracy and computation time are then compared to the same values achieved by non-merged datasets. With the datasets and the program ready, it is possible to proceed to the experiments and the analysis of their results. The aforementioned resources should now let us compare the algorithms.

6. Analysis of Experiment Results

Having built and tested the prototype, we have achieved comparable results. This section covers the discussion of results obtained, as well as conclusions drawn from them, and elaborates on future research potential.

6.1. Algorithm Accuracy

Tables 1–6 present the results of TWNFI, TR-SVM and Naive Bayes classifier evaluation testing of accuracy for a different amount of samples in a subset of data on the first cardiovascular disease dataset, for 100, 250, 500, 1000, 2500, 5000, 1000 samples and the whole set, respectively.

For 100 samples, the TRSVM performs with the best accuracy (at 72%), followed by the Complement Naive Bayes classifier, followed by TWNFI, followed by the Gaussian Naive Bayes classifier. While the TRSVM is shown to be the most accurate, it also takes the most time to both fit and train its model. The Complement NB classifier is second in accuracy, and computes a lot faster, but loses to TRSVM by an entire 12%. TWNFI and Gaussian NB classifier both lose in this scenario, with the latter's accuracy going below 50%, and its benefits of a faster computation time being rendered irrelevant (Table 1).

Table 1. Comparison of chosen machine learning algorithms for 100 samples.

	TR-SVM	TWNFI	NB Classifier (Gaussian)	NB Classifier (Complement)
Accuracy percentage	75%	50%	42%	60%
Avg. fitting time (ms)	106,353.96	27,047.08	0.6966	0.9453
Avg. prediction time (ms)	53,244.02	1.4250	0.1470	0.1038
Mean square error (for 50 experiments)	0.28	0.5	0.58	0.4

For 500 and 1000 samples, we observe a situation, with both the TRSVM and the TWNFI coming out with better results, with the former's computational times increasing further, and the latter's computational times decreasing bit by bit. The NB classifiers, however, in addition to computational time increase, perform considerably worse here, with their accuracy decreasing by 6–14% and 2–8%, respectively (Tables 2 and 3).

Table 2. Comparison of chosen machine learning algorithms for 500 samples.

	TR-SVM	TWNFI	NB Classifier (Gaussian)	NB Classifier (Complement)
Accuracy percentage	70%	66%	54%	52%
Avg. fitting time (ms)	4,629,342.37	20,456.25	1.2287	1.2396
Avg. prediction time (ms)	425,968.14	0.5530	0.1445	0.07388
Mean square error (for 50 experiments)	0.3	0.34	0.46	0.48

Table 3. Comparison of chosen machine learning algorithms for 1000 samples.

	TR-SVM	TWNFI	NB Classifier (Gaussian)	NB Classifier (Complement)
Accuracy percentage	70%	64%	46%	58%
Avg. fitting time (ms)	12,611,525.95	5604.68	1.9520	2.070
Avg. prediction time (ms)	543,569.71	0.4675	0.1426	0.0760
Mean square error (for 50 experiments)	0.3	0.36	0.54	0.42

For 2500 samples, we see that the TRSVM begins to perform worse, with it coming second to the TWNFI (itself at 76%) with 68%. Its training time also reaches approximately 9 h at this point, making further deployment impractical. On the other hand, the TWNFI both decreases its computational time and increases its accuracy. Both NB classifiers, meanwhile, despite remaining relatively fast, do not give satisfactory accuracy, with 54% and 40% for Gaussian NB and Complement NB classifiers, respectively (Table 4).

Table 4. Comparison of chosen machine learning algorithms for 2500 samples.

	TR-SVM	TWNFI	NB Classifier (Gaussian)	NB Classifier (Complement)
Accuracy percentage	68%	76%	54%	40%
Avg. fitting time (ms)	33,368,058.12	13,610.57	4.0899	4.1547
Avg. prediction time (ms)	388,989.52	0.4662	0.1471	0.0814
Mean square error (for 50 experiments)	0.32	0.24	0.46	0.6

For 5000 samples, we observe a similar situation to the one with 2500 samples, with the TWNFI becoming even faster and more accurate on a larger subset, outputting 82% correct predictions. The quality of TRSVM, meanwhile, continues to decrease, dropping to around the same level as the NB classifiers, and further becoming slower compared to the other algorithms, with its training time now reaching almost 2 days.

For 10,000 samples, the TRSVM run was terminated due to extremely long execution times (the program did not compute after several days) and Python *RuntimeError*. In the context of the implementation, it indicates that the program has found two vectors with the same input values that belong to different classes, causing a division by zero in one of the formulas used to calculate the terminated ramp functions the TRSVM uses. This causes further questions regarding the dataset itself. Interestingly, TWNFI's accuracy dropped by 18% compared to the 5000-sample run, for reasons unknown. Despite this, it still comes out better than the Naive Bayes classifiers in terms of accuracy, all while maintaining around the same computational times. Meanwhile, the NB classifiers see a greater increase in computational times, while their accuracy remains relatively low (Table 5).

Table 5. Comparison of chosen machine learning algorithms for 10,000 samples.

	TR-SVM	TWNFI	NB Classifier (Gaussian)	NB Classifier (Complement)
Accuracy percentage	N/A	64%	58%	48%
Avg. fitting time (ms)	N/A	10,901.85	15.105	14.413
Avg. prediction time (ms)	N/A	0.4151	0.1628	0.0863
Mean square error (for 50 experiments)	N/A	0.36	0.42	0.52

For the whole set, we are also unable to compute TRSVM's performance due to lacking the RAM to store the entire Gram Matrix alone (the experiments were conducted on a laptop with 16 GB RAM, whereas the entire Gram Matrix of the set would require approximately 36 GB RAM). TWNFI, meanwhile, sees an increase in accuracy once again, while retaining a close computational time to the previous runs. The NB classifiers, on the other hand, mostly repeat the situation with the 10,000-sample run, remaining low accuracy-wise and having their computational times further increase (Table 6).

Table 6. Comparison of chosen machine learning algorithms for the whole set.

	TR-SVM	TWNFI	NB Classifier (Gaussian)	NB Classifier (Complement)
Accuracy percentage	N/A	78%	56%	48%
Avg. fitting time (ms)	N/A	12,122.58	110.80	102.51
Avg. prediction time (ms)	N/A	0.4547	0.2075	0.1328
Mean square error (for 50 experiments)	N/A	0.22	0.44	0.52

Figure 7 demonstrates the difference between the performance of the chosen algorithms: accuracy, fitting time, prediction time and mean square error.

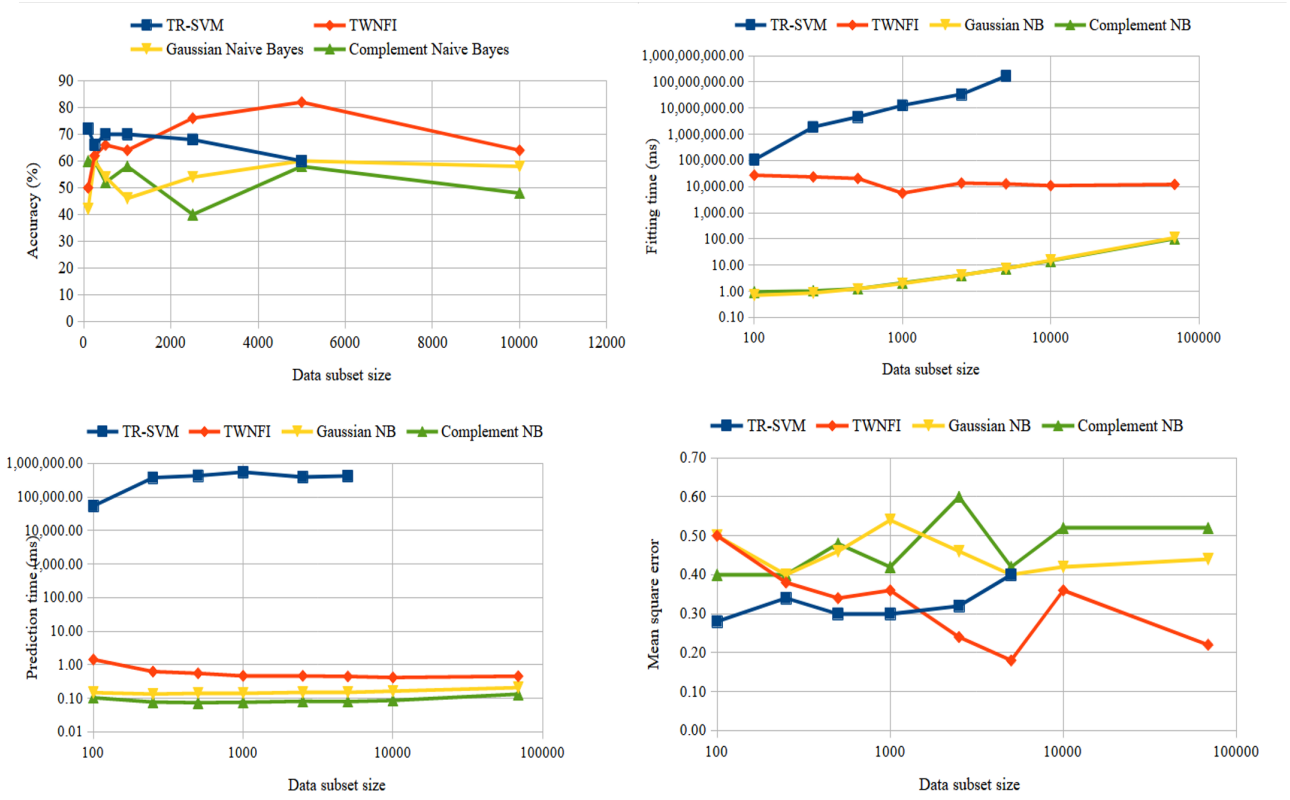

Figure 7. Comparison of accuracy, fitting time, prediction time and mean square error between the chosen algorithm.

While the Naive Bayes classifier takes less time to compute than the others, it returns pretty bad results, with accuracy fluctuating at 40–60% regardless of subset size. This is likely due to the features being dependent on each other, since a known weakness of the Naive Bayes classifier is that it assumes all features are independent from each other. Although the current literature mentions Naive Bayes classifiers being used in the medical field, it is likely that better multicollinearity detection methods are required for optimal results.

TRSVM does not work on the whole set and a 10,000 sample subset, returning *MemoryError* instead, which is caused by hardware limitations, and returns divide by zero *RuntimeErrors*, which is caused by the normalization methods creating two samples with identical feature values that belong to different classes (therefore causing a division by zero in one of the formulae). While it does achieve better accuracy on smaller datasets, it does take a dip on 250 samples for reasons unknown, and the accuracy is shown to decrease at 2500 and more samples due to a higher hyperparameter value in the pruning heuristic.

TWNFI's computational time stays relatively the same regardless of the dataset size, which can be attributed to it building smaller local models for each data sample. However, its performance on smaller subsets appears to be worse than that of TRSVM, which could be explained by SVMs generally working better on smaller datasets. Its dependency on correctly chosen parameters may also complicate its deployment. However, it starts showing better results on larger subsets, likely due to more relevant samples being available for the construction of its local models.

Overall, the relatively low performance of all algorithms (including the ones provided by *scikit-learn*) may indicate further undetected anomalies in the dataset, where further testing of both TRSVM and TWNFI on different datasets has them show much better results. Despite this, the performed calculations are enough to observe several patterns related to the algorithm. While TRSVM performs pretty well on smaller datasets, its computational time becomes impractical on larger ones, and the proposed solution by the original author

trades off accuracy for speed, in ways that are not always acceptable. Meanwhile, TWNFI can perform better in certain conditions, but its extreme dependence on external parameters can make it more difficult to deploy, due to the difficulties of parameter picking in machine learning algorithms.

6.2. TWNFI Hyperparameters and Their Effect on Performance

The quality and computation time of TWNFI may greatly depend on the amount of samples chosen to build the local model. Several parameters specific to the algorithm affect its performance, and therefore require further examination. One of such parameters is the number of samples TWNFI uses to build a local model, the influence of which can be seen in Table 7.

Table 7. TWNFI evaluation with a different number of local model samples, with *Dthr* = 0.2, on a dataset of 250 samples.

Number of Similar Samples	Accuracy Percentage (ms)	Avg. Fitting Time (ms)	Avg. Prediction Time (ms)	Mean Square Error (for 50 Experiments)
15	64%	5845.22	0.3516	0.36
20	76%	8022.30	0.3315	0.24
25	78%	9807.30	0.3514	0.22
30	66%	11,973.35	0.3356	0.34
35	72%	15,024.77	0.3408	0.28

The results show that a larger amount of samples may increase accuracy but also is guaranteed to increase computation time. The accuracy, however, does not necessarily increase with a larger number of samples, which could be due to the fact that a larger amount of samples allows less relevant data vectors to be selected. As an example, from the table, we can see that the accuracy percentage peaks at 25 similar samples and decreases at higher values. Meanwhile, the time steadily increases from 5845.22 ms to 15,024.77 ms.

Another important parameter is the *Dthr* parameter in the ECM algorithm, which plays an instrumental part in clustering according to radius length by serving as a comparison. Due to TWNFI relying on ECM for clusterization and creation of rules, this parameter can greatly affect the amount of rules created, which in turn can affect the accuracy. Table 8 demonstrates how it affects the overall performance.

One thing to note is that increasing the *Dthr* parameter decreases both prediction and fitting time, with a larger *Dthr* parameter reducing the fitting time by more than two times (from 20,404.17 ms to 9877.32 ms in this case), and the prediction time by more than 20 times (from 6.1129 ms to 0.3771 ms here). This is likely caused due to a smaller amount of clusters created by ECM with a larger *Dthr*, and, by extension, a smaller amount of rules, reducing the amount of extra variables in the program. The same, however, cannot be said about the accuracy percentage, which appears to fluctuate through different parameter numbers. The reason for such behavior remains unknown for now.

Table 8. TWNFI evaluation depending on the ECM radius parameter, on a dataset of 250 samples.

Dthr	Accuracy Percentage	Avg. Fitting Time (ms)	Avg. Prediction Time (ms)	Mean Square Error (for 50 Experiments)
0.15	62%	20,404.17	6.1129	0.38
0.20	76%	9,877.32	0.3771	0.24
0.25	84%	9601	0.3363	0.16
0.30	78%	9464.78	0.3582	0.22
0.35	70%	9691.50	0.3607	0.3
0.40	80%	9561.18	0.3308	0.2
0.45	56%	9698.76	0.3391	0.44
0.50	66%	9483.88	0.3377	0.34
0.55	76%	9712.92	0.3506	0.24
0.60	78%	9490.59	0.3625	0.22
0.65	64%	9626.10	0.3368	0.36
0.70	80%	9321.10	0.3298	0.2
0.75	74%	9425.70	0.3219	0.26
0.80	82%	9703.71	0.3397	0.18
0.85	74%	9561.92	0.3265	0.26
0.90	74%	9360.46	0.3209	0.26
0.95	68%	9613.46	0.3274	0.32
1.0	68%	9658.99	0.3435	0.32

6.3. NLP Transformation Accuracy

Table 9 demonstrates the results obtained vian NLP-assisted transformation. It is worth noting several assumptions that were made while taking this table into account.

It is visible that mostly numerical qualities are parsed correctly but seems to make frequent mistakes in recognizing features represented by discrete numerical values, such as gender. This is likely caused by *sense2vec* model's general purpose, rather than it being geared towards scientific language, which was evident during debug when it assigned informal terms as synonyms for column labels. While the presence of these patterns in the program does not necessarily reduce parsing quality, it greatly slows initialization time and forces us to take more samples to have actually helpful words among them. Another problem it appears to face is the detection of negation and uncertainty in the case of boolean values, which is one of the challenges in contemporary medical NLP.

There is also a slight discrepancy in days between the parsed results and actual results, caused by the values in text being expressed in years, rather than days. This discrepancy is ignored when calculating accuracy, with the number being considered correctly parsed if it is not far from the original value.

With the highest accuracy per sample at 100%, and lowest accuracy per sample at 45%, we can see that the approach demonstrates variable results from sample to sample, performing on some better than the others. A likely cause of this is the approach recognizing specific words better than the others, which once again comes back to the limited availability of models.

Overall, the approach shows somewhat positive, although not accurate enough results for the method to be applied on real data. There is a chance, however, that these results are not indicative of the real life environment due to the data samples being created specifically for testing purposes. Real patient note records may show results that are drastically different.

Table 9. Results of NLP-assisted data transformation from patient notes.

Sample Number	Real Sample	Actual Sample	Comp. Time (ms)	Accuracy per Sample
1	[18,262.5, 2, 168.0, 62.0, 110.0, 80.0, 0, 0, 0, 0, 1]	[18,393, 2, 168, 62, 110, 80, 0, 0, 0, 0, 1]	2929.77	100%
2	[14,610.0, 2, 165.0, 60.0, 120.0, 80.0, 1, 1, 0, 0, 0]	[14,791, 2, 165, 60, 120, 80, 0, 0, 0, 0, 0]	3030.48	82%
3	[21,184.5, 1, 170.0, 75.0, 130.0, 70.0, 0, 0, 0, 0, 0]	[21,296, 1, 170, 75, 130, 70, 0, 0, 0, 0, 0]	2992.86	100%
4	[23,010.75, 2, 151.0, 92.0, 130.0, 90.0, 0, 1, 0, 0, 0]	[23,204, 1, 151, 92, 130, 90, 0, 0, 0, 0, 0]	2992.74	82%
5	[15,705.75, 2, 185.0, 88.0, 133.0, 89.0, 1, 1, 0, 0, 0]	[15,946, 2, 185, 88, 133, 89, 1, 1, 0, 0, 1]	2630.13	91%
6	[20,454.0, 2, 100.0, 78.0, 140.0, 90.0, 1, 1, 1, 0, 0]	[20,627, 2, 168, 78, 140, 90, 1, 1, 1, 0, 1]	2844.75	82%
7	[21,915.0, 2, 176.0, 74.0, 120.0, 80.0, 0, 1, 0, 1, 0]	[22,111, 1, 176, 74, 120, 80, 0, 0, 0, 0, 1]	2955.90	64%
8	[14,244.75, 2, 167.0, 66.0, 110.0, 70.0, 0, 0, 0, 0, 1]	[14,493, 1, 167, 66, 110, 70, 0, 0, 0, 0, 1]	2875.22	91%
9	[23,376.0, 2, 169.0, 73.0, 140.0, 90.0, 0, 1, 0, 0, −1]	[23,376, 2, 169, 73, 140, 90, 0, 0, 0, 0, 1]	2620.89	82%
10	[18,993.0, 2, 175.0, 53.0, 140.0, −1, 1, 1, 0, 0, 1]	[19,081, 2, 175, 53, 140, 80, 0, 0, 1, 0, 1]	2492.70	64%
11	[21,549.75, 2, 174.0, 82.0, 120.0, 80.0, 1, 1, 0, 0, 1]	[21,665, 2, 174, 82, 120, 80, 0, 0, 0, 0, 1]	2809.15	82%
12	[16,436.25, 2, 170.0, 68.0, 150.0, 90.0, 1, 1, 0, 0, 0]	[16,608, 1, 170, 68, 150, 90, 1, 0, 0, 0, 1]	3074.92	73%
13	[−1, 1, 157.0, −1, 1, 130.0, 1, 1, 0, 0, 1]	[22,608, 1, 157, 70, 130, 90, 0, 0, 0, 0, 1]	3003.69	45%
14	[23,376.0, 2, 1, 1, −1, −1, 1, 1, 0, 0, −1]	[23,389, 1, 163, 63, 120, 80, 1, 1, 0, 0, 0]	2118.30	45%
15	[19,358.25, 2, 171.0, 79.0, 80.0, −1, 0, 1, 0, 0, 0]	[19,668, 2, 171, 79, 120, 80, 0, 0, 0, 0, 1]	2475.30	64%
16	[20,454.0, 1, 180.0, 75.0, 1, −1, 0, 1, 0, 0, 1]	[20,554, 2, 180, 75, 120, 80, 0, 0, 0, 0, 1]	2632.36	64%
17	[14,610.0, 2, 170.0, 68.0, 120.0, −1, 0, 1, 0, 0, 0]	[14,798, 2, 170, 68, 120, 80, 0, 0, 0, 0, 0]	2524.41	82%
18	[23,010.75, 1, 155.0, 56.0, 120.0, 80.0, 0, 1, 0, 0, −1]	[23,191, 1, 155, 56, 120, 80, 0, 0, 0, 0, 1]	3003.80	82%
19	[21,184.5, 2, 166.0, 101.0, 140.0, 90.0, 1, 1, 0, 1, 0]	[21,270, 1, 166, 101, 140, 90, 1, 0, 0, 0, 1]	2910.77	64%
20	[23,010.75, 1, 164.0, 82.0, 130.0, 70.0, 1, 0, 0, 0, 0]	[23,343, 1, 164, 82, 130, 70, 1, 0, 0, 0, 1]	2980.47	91%
Initialization time (ms):		8030.26		
Total accuracy percentage:		76%		

6.4. Unit Conversion Evaluation

Tables 10 and 11 display the results for similarity-based conversion evaluation for sets with N = 100, 8 and 2 known features, respectively, with the *age*, *height* and *weight* in the second dataset being in *years*, *feet* and *pounds*, as opposed to *days*, *centimetres* and *kilogram*. The predicted values display the program-guessed conversion coefficients, as opposed to the actual, known values of those units.

Table 10. Results for experimental unit conversion approach with 8 known traits and 100 experiments depending on the number of similar samples.

Number of Similar Samples	Predicted Values	Actual Values
50	374.54, 2.55, 0.47	365.25, 2.54, 0.45
100	368.52, 2.56, 0.46	365.25, 2.54, 0.45
250	367.6, 2.54, 0.46	365.25, 2.54, 0.45
500	373.15, 2.54, 0.46	365.25, 2.54, 0.45
1000	375.35, 2.53, 0.47	365.25, 2.54, 0.45
2500	369.11, 2.55, 0.48	365.25, 2.54, 0.45

Table 11. Results for experimental unit conversion approach with 2 known traits and 100 experiments depending on the number of similar samples.

Number of Similar Samples	Predicted Values	Actual Values
50	370.20, 2.54, 0.48	365.25, 2.54, 0.45
100	370.82, 2.53, 0.46	365.25, 2.54, 0.45
250	371.17, 2.54, 0.47	365.25, 2.54, 0.45
500	372.60, 2.54, 0.44	365.25, 2.54, 0.45
1000	367.74, 2.55, 0.47	365.25, 2.54, 0.45
2500	370.81, 2.58, 0.47	365.25, 2.54, 0.45

In an environment without any prior manipulations to the data, with the formats similar to each other, the approach yields pretty good results, with best values (367.6, 2.54, and 0.46 for the first experiment, and 367.74, 2.54, 0.46 for the second one) achieved being not far away from the expected coefficients. However, the optimal numbers of similar samples (250 and 1000 in the cases of these experiments) appear to differ from case to case, with a smaller amount of known traits possibly requiring larger samples.

While the similarity-based conversion method has proven to be relatively accurate when tested on data with a high number of known traits without discrepancies, both the simulated environment with split data and when used to merge two datasets alone, it has proven to be more unstable and less effective. It is also clearly visible that, in the case of the first table, amounts of samples greater than 250 distort the results due to less similar samples influencing the calculation.

Upon a closer look at Table 12, where the datasets have been split and the program was forced to reconstruct the values we know, we can see how certain data samples look when extended, which reveals a few things. Depending on the parameters, the restored sample may either get distorted by a large number of outliers as similar samples with a smaller number, or will be too generalized under a larger number of similar samples, which plays into the widely known problem of overfitting.

Table 12. Individual sample transformation, the experimental unit conversion approach.

Similar Samples	Before Transformation	After Transformation
50	[23,143.48, 1.0, 163.92, 73.96, 135.0, 80.0, 1.0, 2.0, 0.0, 0.0, 0.0]	[22,431.0, 1.0, 163.0, 72.0, 135.0, 80.0, 1.0, 2.0, 0.0, 0.0, 0.0]
100	[23,058.68, 1.0, 158.69, 128.300, 140.0, 90.0, 2.0, 2.0, 0.0, 0.0, 1.0]	[22,601.0, 1.0, 158.0, 126.0, 140.0, 90.0, 2.0, 2.0, 0.0, 0.0, 1.0]
250	[20,540.0, 1.0, 170.0, 72.0, 120.0, 80.0, 2.0, 1.0, 0.0, 0.0, 1.0]	[20740.02, 1.0, 169.60, 72.238, 120.0, 80.0, 2.0, 1.0, 0.0, 0.0, 1.0]
500	[19,401.56, 2.0, 182.63, 106.37, 180.0, 90.0, 3.0, 1.0, 0.0, 1.0, 0.0]	[19,066.0, 2.0, 183.0, 105.0, 180.0, 90.0, 3.0, 1.0, 0.0, 1.0, 0.0]
1000	[21,300.94, 1.0, 170.77, 71.54, 120.0, 80.0, 2.0, 1.0, 0.0, 0.0, 1.0]	[20,540.0, 1.0, 170.0, 72.0, 120.0, 80.0, 2.0, 1.0, 0.0, 0.0, 1.0]
2500	[20,737.45, 1.0, 171.37, 75.71, 120.0, 80.0, 2.0, 1.0, 0.0, 0.0, 1.0]	[20,540.0, 1.0, 170.0, 72.0, 120.0, 80.0, 2.0, 1.0, 0.0, 0.0, 1.0]

A similar trend can be observed in Table 13, which demonstrates how the number of steps N impacts the overall performance. While the dependency between number of steps and result quality does not appear to be linear, it shows that the best results are also produced by the amount of steps around 50–100, while reducing in accuracy at larger parameters. The amount of steps also seems to greatly increase computational time.

Table 13. Evaluation of similarity-based conversion approach and effect of the experiment number N, with 250 samples (k = 250) per experiment.

Number of Steps N	Predicted Values	Actual Values	Computation Time
10	361.79, 2.57, 0.51	365.25, 2.54, 0.45	14,794.72
20	377.86, 2.56, 0.43	365.25, 2.54, 0.45	27,955.9273
30	380.79, 2.53, 0.47	365.25, 2.54, 0.45	40,537.82
40	373.96, 2.60, 0.47	365.25, 2.54, 0.45	57,885.87
50	370.09, 2.55, 0.45	365.25, 2.54, 0.45	67,754.48
100	368.08, 2.58, 0.46	365.25, 2.54, 0.45	143,455.42
250	373.50, 2.55, 0.46	365.25, 2.54, 0.45	328,075.21

As expected, the number of steps makes the computation time grow, but a larger number of steps does not necessarily equal higher accuracy. Here, we see that the best coefficient for years achieved in only 10 steps, whereas the better ones for inches and pounds are achieved in 250 and 50 steps, respectively. A possible future improvement to the method may be modifying it to use different amounts of steps for different features (Table 12).

Depending on the parameters, the converted values suffer various degrees of distortion, sometimes missing their mark by nearly two years. The distortion becomes even more evident on the dataset merging test that did not include label interpretation (which was switched off due to it automatically converting the example trait), where only two traits used to for the method are achieved by converting the other sets' numericals into booleans, therefore adding a degree of inaccuracy, as shown in Table 14.

With the best result being a coefficient of 471.80, the worst one being that of 542.29, and both being far from the intended coefficient of 365.25, it is clear that the approach is a lot less reliable when preceded by multiple other transformations to the data, which have already added a degree of distortion.

Table 14. Results for experimental unit conversion approach with two common traits matched from previous methods and 100 steps depending on the number of similar samples.

Number of Similar Samples	Predicted Values	Actual Values
50	523.12	365.25
100	494.48	365.25
200	498.29	365.25
250	542.89	365.25
300	532.68	365.25
500	471.80	365.25
1000	509.07	365.25
2500	503.09	365.25

6.5. Combined Methods for Data Merging

Table 15 demonstrates the performance of algorithms on a dataset that has been combined, on subsets of 250 samples, with average values from 50 experiments each. The accuracy percentage computation time with merged and converted values may be lower than with those that did not undergo this, they are still not very far off. As it seems, the approach is usable. Interestingly, the Complement Naive Bayes classifier shows better results than during tests on unmerged datasets.

Table 15. Results of different models with experimental unit conversion approach.

	TR-SVM	TWNFI	NB Classifier (Gaussian)	NB Classifier (Complement)
Accuracy percentage	65%	58%	59%	64%
Avg. fitting time (ms)	1,644,799.63	19,001.99	0.9101	0.9827
Avg. prediction time (ms)	625,832.98	0.7940	0.1361	0.0728
Mean square error	0.35	0.42	0.41	0.36

6.6. Results Discussion

As was obtained, Naive Bayes classifiers performed rather poorly on medical data, with the results for both of classifiers fluctuating around 40–60%, which may be linked to the issue of multicollinearity mentioned in the previous section, and the system's inability to detect it in all of the cases. Though the results appear to be better with feature selection, the classifier as it is not enough to produce a reliably accurate diagnosis for usage in real-life cases.

In comparison, TWNFI performs poorly on smaller datasets, only reaching an accuracy of 50% on a 100-sample subset, but seems to perform relatively well on larger datasets with proper parameters, achieving the highest accuracy in the series of experiments at 82% for 5000 samples and further confirming the ideas mentioned in [28]. TWNFI's fitting time also decreases with a larger dataset, going from 20,456.25 ms to 12,122.58 ms, which could be attributed to a larger amount of samples allowing the algorithm to pick a more precise set that takes less time to tune, all local sub-models in testing being of relatively the same size, and clustering being only applied to a local area. However, to adjust TWNFI to a specific problem, one would have to pick suitable parameters (like number of samples) for it to both make accurate predictions and perform within a reasonable time limit.

TR-SVM's showcases better results on smaller datasets, achieving the highest accuracy of 72% among all the algorithms on a subset of 100 samples, but it is impractical to use on larger ones due to large computation times, which reached several days on a subset of 5000 samples, which can take days even on more powerful machines. While the pruning heuristic does offer to decrease some of this time, it sacrifices a certain amount of accuracy in favor of faster execution, with larger values of parameter S decreasing accuracy from around 90% to only 60%, and is still not enough to make a great difference in the case of larger datasets. As a result, it is best suited for the same cases as other SVM: small datasets of high-dimensional data.

The NLP note parser appears to give tolerable results as of now, but still far from being viable due to the way it is currently implemented, with accuracy per sample varying from 100% to 45%, as well as the lack of available datasets containing patient notes. Its inaccuracies are also partially due to a lack of publicly available data that features medical notes, resulting in a worse recognition quality. In addition to that, while parsing itself only takes approximately 2000–3000 ms, the same cannot be said about the initialization time, which involves loading all the models. As a result, it may be more effective to parse several notes simultaneously.

The experimental unit conversion works in most cases; however, its effectiveness greatly depends on the parameters such as the number of similar neighbours and the amount of steps. More experiments on different kinds of datasets are required to develop it further. The quality of the conversion also appears to greatly decrease when applied together with other methods, resulting in coefficients almost 1.5 times larger than the target results, such as the amount of days in year being erroneously assumed to be closer to 500 instead of the intended 365.25.

6.7. Research Potential and Future Possibilities

Despite the aforementioned conclusions drawn in our research, new questions arise both out of the theoretical implications and the results. Due to TWNFI's transductive

modelling logic, the algorithm can be further perfected by adding feature selection to one of its steps, making it possible to eliminate the columns with fields that were assigned placeholder values pre-training data mergers, further improving the performance of the algorithm. In addition, the automation of parameter selection should be further looked into, with methods such as cross-validation being of potential help in this case. With its speed being TRSVM's main drawback, the further research must be directed into its optimization, as in [42].

With the application of data merging, the problem of multicollinearity still has not been solved in this paper. Although feature selection may be applied to the tables, it may not always detect multicollinear features if the tables have been merged. With no apparent means to solve this problem as of now, further research in this direction is necessary to make the diagnosis algorithms more efficient in cases of merged data. In addition to that, there is a possibility that polynomial regression could be applied for merging variables that do not have a linear dependency.

The NLP transformation could use further development and testing on real patient data and research in collaboration with medical organizations. While it has been tested in our research with tolerable results, the data used are still an approximation that may not be completely reflective of real life data, which is why it would require further work later on. It includes the threshold and coefficient parameters for different models involved in the program, which were obtained experimentally for this research, but could be obtained automatically instead if they are somehow translated into an optimization problem.

The idea of an image classifier mentioned back in Section 5 could be a topic for an entirely different, much more comprehensive and in-depth research. The problems regarding its implementation poses are multiple, and could not be covered by this paper, making it a potential future research target.

7. Conclusions

The paper presented several solutions towards automating medical data transformation and tests on both natural and artificially created datasets. Selected algorithms used for diagnosis were implemented, tested and had their performance compared in this paper, with their parameters also investigated, making it easier to choose an algorithm better suited for a specific case. Whereas TRSVM may be better suited for smaller datasets with a high amount of dimensions, TWNFI performs better on larger ones, and does not have the performance problem TRSVM does.

Since it appears that different datasets yield different accuracy rates across all methods, unanswered questions remain about transforming them to improve algorithm accuracy. While the subsets used in our research are all balanced with the exception of instances where the entire datasets are being used, it does very little to improve the performance of the methods. To improve overall algorithm performance, additional factors that influence accuracy need to be identified.

Despite the scarcity of medical notes, an NLP parser was created, enabling transformation from text to table format, in an attempt to combat the problem of heterogeneity. The parser combines several NLP techniques, such as NER, rule-based recognition and word vectors to extract the necessary values from the text and fill an entry with them in accordance with a schema specified in a file. Although its results are not perfect, it still leaves room for improvement on natural datasets and actual patient notes. Another possible improvement would be modifying the negation detection in the method to make the recognition of Boolean values more effective.

While the experimental unit conversion method has been shown to work in cases with a lot of known columns that are alike in both datasets being merged, it became a lot less reliable when such columns were fewer, or when it was combined with other data transformation methods. One question regarding the unit conversion method would be its modification to allow different parameters per individual column, due to testing results showing different parameters resulting in optimal values for different columns.

This modification, however, would also involve solving the problem of picking optimal parameters for such approach.

The research performed in turn puts more questions up for discussion and further research, such as multicollinearity and ways to detect it, as well as the possibility of an image classifier, and possible applications of polynomial regression to transform data. In addition to that, the diagnosis methods could benefit from cross-validation, since it would help even more to determine optimal parameters, These problems, as well as many other related ones, should be further explored and solved, to continue where this research has left off. Though the ideas described in the paper have been proven to work on the presented datasets, they have quite a few drawbacks, leaving space for future research.

Author Contributions: Conceptualization, E.V. and A.P.-M.; methodology, E.V. and A.P.-M.; validation, A.P.-M. and W.M.; formal analysis, E.V., A.P.-M. and W.M.; investigation, E.V.; resources, E.V.; data curation, E.V.; writing—original draft preparation, E.V., A.P.-M. and W.M.; writing—review and editing, W.M.; visualization, A.P.-M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhang, L.; Chen, X.; Chen, T.; Wang, Z.; Mortazavi, B.J. DynEHR: Dynamic adaptation of models with data heterogeneity in electronic health records. In Proceedings of the 2021 IEEE EMBS International Conference on Biomedical and Health Informatics (BHI), Athens, Greece, 27–30 July 2021.
2. Benito, P.J.F. Healthcare Data Heterogeneity and Its Contribution to Machine Learning Performance. Ph.D. Thesis, Universitat Politècnica de València, Valencia, Spain, 2020.
3. He, J. Learning from Data Heterogeneity: Algorithms and Applications. In Proceedings of the 26th International Joint Conference on Artificial Intelligence Early Career, Melbourne Australia, 19–25 August 2017; pp. 5126–5130.
4. Satti, F.A.; Ali, T.; Hussain, J.; Wajahat, A.K.; Asad, M.K.; Sungyoung, L. Ubiquitous Health Profile (UHPr): a big data curation platform for supporting health data interoperability. *Computing* **2020**, *102*, 2409–2444. [CrossRef]
5. Dhayne, H.; Haque, R.; Kilany, R.; Taher, Y. In Search of Big Medical Data Integration Solutions—A Comprehensive Survey. *IEEE Access* **2019**, *7*, 91265–91290. [CrossRef]
6. Khnaisser, C.; Lavoie, L.; Fraikin, B.; Barton, A.; Dussault, S.; Burgun, A.; Ethier, J.F. Using an ontology to derive a sharable and interoperable relational data model for heterogeneous healthcare data and various applications. *Methods Inf. Med. AAM* **2022**, *61*, e73–e88. [CrossRef] [PubMed]
7. Kiourtis, A.; Mavrogiorgou, A.; Kyriazis, D. Gaining the Semantic Knowledge of Healthcare Data through Syntactic Models Transformations. In Proceedings of the 2017 International Symposium on Computer Science and Intelligent Controls (ISCSIC), Budapest, Hungary, 20–22 October 2017; pp. 102–107.
8. Litman, D.J. Automating the Conversion of Data: A Review of Recent Progress. *IEEE Trans. Knowl. Data Eng.* **2017**, *29*, 912–925.
9. Barr, R.H. Natural language processing in healthcare data integration. In Proceedings of the AMIA Annual Symposium, Chicago, IL, USA, 10–14 November 2017; American Medical Informatics Association: Bethesda, MD, USA, 2007.
10. Jurafsky, D.; Martin, J.H. Martin, J.H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*; ; Pearson Education: London, UK, 2019.
11. Haverkort, J.W.M.; Reinders, M.J.T.; Houben, G.J.; Schadd, M.J.A.; Akkermans, H.A. Integrating heterogeneous datasets: Challenges and solutions. *J. Database Manag.* **2005**, *16*, 1–26.
12. Zaki, M.M. Unit conversion in heterogeneous databases and data warehouses. *IEEE Trans. Knowl. Data Eng.* **2004**, *16*, 578–589.
13. Ojha, P.; Sahu, S.K.; Chaturvedi, S. Multicollinearity: issues, detection, and remedies. *J. Big Data* **2019**, *6*, 1–17.
14. Schuemie, M.J.; Mallon, W.G.; Ryan, P.B. A review of multicollinearity in medical research. *J. Clin. Epidemiol.* **2011**, *64*, 945–953.
15. Chaudhry, M.M.; Chawla, S. A Review of Multicollinearity Diagnosis and Remedial Measures in Multiple Regression Analysis. *Res. J. Appl. Sci. Eng. Technol.* **2016**, *11*, 650–657.
16. Sniegula, A.; Poniszewska-Maranda, A.; Chomatek, L. *Towards the Named Entity Recognition Methods in Biomedical Field*; Chatzigeorgiou, A., Ed.; Springer International Publishing: Berlin/Heidelberg, Germany, 2020; pp. 375–387, ISBN 978-3-30-38918-5. [CrossRef]
17. Adelakun, O.F.; Ismail, H. Natural language processing for medical applications: A review. *Int. J. Med. Inform.* **2019**, *122*, 103398.
18. Joty, S.H.; Goecke, R. Natural language processing in healthcare applications: A survey. *IEEE Access* **2020**, *8*, 55984–56007.

19. Zhang, Q.; Wang, X. Natural language processing in healthcare: A survey of applications and challenges. *IEEE Access* **2020**, *8*, 151576–151594.
20. Demeester, J.C.; Chapman, L.S.; Zhou, X. Natural language processing applications in the medical field. *Artif. Intell. Med.* **2015**, *64*, 123–139.
21. Tsoukatos, T.K. Natural Language Processing Techniques in the Medical Field. *Int. J. Comput. Linguist. Nat. Lang. Process.* **2014**, *1*, 11–22.
22. Krzeszewska, U.; Poniszewska-Mar, A.; Ochelska-Mierzejewska, J. Systematic comparison of vectorization methods in classification context. *Appl. Sci.* **2022**, *12*, 5119. [CrossRef]
23. Aldahiri, A.; Alrashed, B.; Hussain, W. Trends in using IoT with machine learning in health prediction system. *Forecasting* **2021**, *3*, 181–206. [CrossRef]
24. Ak, M.F. A comparative analysis of breast cancer detection and diagnosis using data visualization and machine learning applications. *Healthcare* **2020**, *8*, 111. [CrossRef]
25. Garg, A.; Mago, V. Role of machine learning in medical research: A survey. *Comput. Sci. Rev.* **2021**, *40*, 100370. [CrossRef]
26. Panch, T.; Szolovits, P.; Atun, R. Artificial intelligence, machine learning and health systems. *J. Glob. Health* **2018**, *8*, 020303. [CrossRef]
27. Sciforce. Top AI algorithms for Healthcare. Available online: https://medium.com/sciforce/top-ai-algorithms-for-healthcare-aa5007ffa330 (accessed on 9 July 2020).
28. Song, Q.; Kasabov, N. TWNFI—A transductive neuro-fuzzy inference system with weighted data normalization for personalized modelling. *Neural Netw.* **2006**, *19*, 1591–1596. [CrossRef]
29. Kiourtis, A.; Mavrogiorgou, A.; Kyriazis, D. Aggregating Heterogeneous Health Data through an Ontological Common Health Language. In Proceedings of the 10th International Conference on Developments in eSystems Engineering (DeSE), Paris, France, 14–16 June 2017; pp. 175–181.
30. Ganie, S.M.; Majid, B.M.; Tasleem, A. Machine Learning Techniques for Big Data Analytics in Healthcare: Current Scenario and Future Prospects. In *Telemedicine: The Computer Transformation of Healthcare*; Springer: Cham, Switzerland, 2022; pp. 103–123.
31. Pfaff, E.R.; Champion, J.; Bradford, R.L.; Clark, M.; Xu, H.; Fecho, K.; Krishnamurthy, A.; Cox, S.; Chute, C.G.; Overby, T.C.; et al. Fast Healthcare Interoperability Resources (FHIR) as a Meta Model to Integrate Common Data Models: Development of a Tool and Quantitative Validation Study. *JMIR Med. Inform.* **2019**, *16*, e15199. [CrossRef] [PubMed]
32. Mavrogiorgou, A.; Kiourtis, A.; Touloupou, M.; Kapassa, E.; Kyriazis, D.; Themistocleous, M. *The Road to the Future of Healthcare: Transmitting Interoperable Healthcare Data through a 5G Based Communication Platform. Information Systems, EMCIS 2018*; Lecture Notes in Business Information Processing; Themistocleous, M., Rupino da Cunha, P., Eds.; Springer: Cham, Switzerland, 2019; Volume 341.
33. Punia, S.K.; Kumar, M.; Stephan, T.; Deverajan, G.G.; Patan, R. Performance analysis of machine learning algorithms for big data classification: Ml and ai-based algorithms for big data analysis. *Int. J. Health Med. Commun. (IJEHMC)* **2021**, *12*, 60–75. [CrossRef]
34. Mohan, S.; Thirumalai, C.; Srivastava, G. Effective heart disease prediction using hybrid machine learning techniques. *IEEE Access* **2019**, *7*, 81542–81554. [CrossRef]
35. Wang, L. Heterogeneous Data and Big Data Analytics. *Autom. Control. Inf. Sci.* **2017**, *3*, 8–15. [CrossRef]
36. Sarker, I.H. Machine learning: Algorithms, real-world applications and research directions. *SN Comput. Sci.* **2021**, *2*, 1–21. [CrossRef]
37. Mehbodniya, A.; Lazar, A.J.P.; Webber, J.; Sharma, D.K.; Jayagopalan, S.; Singh, P.; Rajan, R.; Pandya, S.; Sengan, S. Fetal health classification from cardiotocographic data using machine learning. *Expert Syst.* **2021**, *39*, e12899. [CrossRef]
38. Halpern, J.Y.; Okonkowski, D.C. The Challenges of Machine Learning in Medicine. *N. Engl. J. Med.* **2018**, *379*, 1814–1816.
39. Rayan, K.; Rajpurkar, P.; Topol, E.J. Self-supervised learning in medicine and healthcare. *Nat. Biomed. Eng.* **2022**, *6*, 1–7.
40. Fei, W.; Casalino, L.P.; Khullar, D. Deep learning in medicine—Promise, progress, and challenges. *JAMA Intern. Med.* **2019**, *179*, 293–294.
41. Razzak, M.I.; Naz, S.; Zaib, A. Deep learning for medical image processing: Overview, challenges and the future. In *Classification in BioApp*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 323–350.
42. Merler, S.; Jurman, G. Terminated Ramp–Support Vector Machines: A nonparametric data dependent kernel. *Neural Netw.* **2006**, *19*, 1597–1611. [CrossRef]
43. Iroju, O.G.; Olaleke, J.O. A Systematic Review of Natural Language Processing in Healthcare. *Int. J. Inf. Technol. Comput. Sci.* **2015**, *7*, 44–50. [CrossRef]
44. Israel, C.V.; Yu, W.; Cordova, J.J. Multiple fuzzy neural networks modeling with sparse data. In Proceedings of the International Conference on Fuzzy Systems, Barcelona, Spain, 18–23 July 2010; pp. 1–7. [CrossRef]
45. Elshawi, R.; Maher, M.; Sakr, S. Automated Machine Learning: State-of-The-Art and Open Challenges. *arXiv* **2019**, arXiv:1906.02287.
46. Piedra, D.; Ferrer, A.; Gea, J. Text Mining and Medicine: Usefulness in Respiratory Diseases. *Arch. Bronconeumol.* **2014**, *50*, 113–119. [CrossRef] [PubMed]
47. Mikhailidis, D.P.; Papamichael, C.M.; Banach, M. Machine learning techniques aiming to improve cardiovascular disease prevention and treatment: A review. *Heart* **2017**, *103*, 1733–1740.
48. Fumera, G.; Roli, F. Machine learning techniques for cardiovascular disease prediction. *Artif. Intell. Med.* **2016**, *71*, 3–19.

49. Malek, M.H.; Kavousi, M. Machine learning techniques in cardiovascular disease diagnosis and prognosis. *BMC Med. Inform. Decis. Mak.* **2016**, *16*, 1–11.
50. Xu, Y.; Li, Y. Machine learning techniques for cardiovascular disease risk prediction: progress and perspectives. *Bioinformatics* **2017**, *33*, 2044–2052.
51. Xu, J.; Yao, X. Training-resampling based SVM for imbalanced classification. *IEEE Trans. Neural Netw. Learn. Syst.* **2017**, *28*, 1094–1105.
52. Ingvaldsen, J.; Veres, C. Using the WordNet Ontology for Interpreting Medical Records. In Proceedings of the CAiSE, Riga, Latvia, 7–11 June 2004; pp. 355–358.
53. Groot, Koerkamp, B.; Weinstein, M.C.; Stijnen, T.; Heijenbrok-Kal, M.H.; ; Hunink, M.G. Uncertainty and patient heterogeneity in medical decision models. Medical decision-making. *Int. J. Soc. Med. Decis. Making* **2010**, *30*, 194–205. [CrossRef]
54. Sindhu, C.S.; Hegde, N.P. A framework to handle data heterogeneity contextual to medical big data. In Proceedings of the 2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Madurai, India, 10–12 December 2015; pp. 1–7. [CrossRef]
55. Jiang, Z.; Zeng, J.; Zhang, S. Inter-training: Exploiting unlabelled data in multi-classifier systems. *Knowl.-Based Syst.* **2013**, *45*, 8–19. [CrossRef]
56. Saltelli, A.; Ratto, M.; Andres, T.; Campolongo, F.; Cariboni, J.; Gatelli, D.; Saisana, M.; Tarantola, S. *Global Sensitivity Analysis: The Primer*; John Wiley & Sons: Chichester, UK, 2008.
57. Ferson, S.; Ginzburg, L. Deterministic and probabilistic sensitivity analysis. *Reliab. Eng. Syst. Saf.* **2004**, *83*, 1–17.
58. Saltelli, A.; Annoni, P.; Azzini, I.; Campolongo, F.; Ratto, M.; Tarantola, S. Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. *Comput. Phys. Commun.* **2010**, *81*, 259–270. [CrossRef]
59. Chawla, S.; Raghavan, V. TWNFI: Training with noisy feature injection for enhanced deep learning on imbalanced data. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 3730–3738.
60. Mitchell, T. *Machine Learning*; McGraw Hill: New York, NY, USA, 1997.
61. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
62. Pandas. The Pandas Development Team. pandas-dev/pandas. 2020. Available online: https://github.com/pandas-dev/pandas (accessed on 15 August 2020).
63. Harris, C.R.; Millman, K.J.; van der Walt, S.J. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef]
64. Honnibal, M.; Montani, I.; Van Lan-deghem, S.; Boyd, A. spaCy:Industrial-strength Natural Language Processing inPython. Documentation, 2020. Available online: https://zenodo.org/record/7445599#.Y7UVLBVBxPY (accessed on 15 August 2020).
65. Neumann, M.; King, D.; Beltagy, I.; Ammar, W. ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. In Proceedings of the 18th BioNLP Workshop and Shared Task (BioNLP@ACL 2019), Florence, Italy, 1 August 2019; pp. 319–327. [CrossRef]
66. Trask, A.; Michalak, P.; Liu, J. sense2vec—A Fast and Accurate Method for Word Sense Disambiguation in Neural Word Embeddings. *arXiv* **2015**, arXiv:1511.06388.
67. Seabold, S.; Perktold, J. Statsmodels: Econometric and statistical modeling with python. In Proceedings of the 9th Python in Science Conference (SCIPY'2010), Austin, TX, USA, 28 June–3 July 2010; pp. 92–96.
68. Cardiovascular Disease Dataset. Available online: https://kaggle.com/sulianova/cardiovascular-disease-dataset (accessed on 9 April 2020).
69. Cardiovascular Disease. Available online: https://kaggle.com/yassinehamdaoui1/cardiovascular-disease (accessed on 9 August 2020).