Utrecht University

Master's thesis

Computing Science

Machine Learning for Referral of Patients with Chest Pains using Regular Care Data

---

Author
Martijn Jansen (6513328)

Supervision
Dr. Ad Feelders
Dr. Bram van Es

Second reader
Prof. dr. Arno Siebes

July 12, 2023

Abstract

This research uses machine learning models to predict the right care for patients referred to the University Medical Center in Utrecht (UMCU). The right care will be predicted using regular care data which includes blood measurements, previous patient appointments, and referral letters. These referral letters will be text mined. The goal is to classify patients early, to prevent expensive scans. Only patients who have been referred to the cardiology department with chest pain symptoms will be considered. Patients are given one or more classes, making this a multi-label classification problem. Time series data will be constructed using patients' history, and models using time series data will be compared to models that cannot. As time series models, the LSTM model will be used and compared to the T-LSTM. The addition of target replication to both models is researched. As non-time series models, the XGBoost, SVM, and Neural Network models will be used. After creating labels using available data, the number of labels is reduced from 30 to 9 using a novel label combination algorithm. Additionally, subgroup discovery is performed, which was able to find a group based on one rule with an accuracy of 73%. Overall, the time-aware LSTM does not perform better than the vanilla LSTM. The time-series model outperforms the non-time-series models. Target replication only gives a performance increase for the vanilla LSTM. The best-performing model, the LSTM with target replication, has a micro F1 score of 0.62 and an AUC of 0.84. Performance might be limited by the fact that labels had to be constructed from scratch.

Contents

Chapter 1

Introduction

In the past few years, machine learning models have been implemented by many sectors and companies [9]. Even though interest in applied machine learning is growing in the medical sector, there have not been many examples where machine learning is applied in the sector [47].

This research uses machine learning models to predict which kind of care a patient needs after being referred by a General Practitioner (GP) to the University Medical Center in Utrecht (UMCU). We restrict attention to patients who have been referred to the cardiology department with chest pain symptoms. This symptom has been chosen as there are many different possible diagnoses that can be the cause of chest pain [26].

Models that provide interesting insight into a patient's condition are called Clinical Decision Systems (CDS's) [13]. However, most CDS's focus on one specific yes or no question or one specific disease [38, 42, 49]. For more complex diseases, these models tend to use expensive diagnosis tools like an MRI. This might cause patients to be incorrectly diagnosed due to not receiving such expensive scans. There are, however, forms of data that are less expensive and routinely measured, called regular care data. This research will use regular care data to diagnose patients without having to receive expensive scans. If the diagnosis can be predicted accurately for some groups of patients, patients can be diagnosed earlier in the future. This would make healthcare cheaper as expensive tests do not have to be performed. Machine learning is specifically well-suited for analyzing regular care data with many different features. Machine learning models can automatically find patterns in data [5]. These patterns might be too subtle for humans to find.

In this research, treatment periods will be used to classify patients. A treatment period of a patient is defined as the period that starts when the patient is referred to the hospital and ends when the patient's care demand has been met. Each of these treatment periods will be assigned one or more classes based on the care received in that period. As a treatment period can be given one or more classes, this research is a multi-label classification problem. These treatment periods and their classes will be constructed from available data. Regular care data which includes blood measurements, referral letters written by the General Practitioner (GP), demographics, and appointments are used to predict the class of a treatment period. Features will be extracted from the GP letters using text mining. By also adding measurements taken before referral to the hospital, a time series will be created.

To predict which hospital department will give the best care to a patient, different

machine learning models will be used, including models that can interpret time series data and models that cannot do so. The following non-time series models will be used: XGBoost, Support Vector Machines, and a Neural Network. These models will be given a summarization of the time-series features. The following time-series models will be used: LSTM and Time-aware LSTM (T-LSTM). The T-LSTM is an improvement on the LSTM proposed by Baytas et al [8]. The performance of the LSTM and T-LSTM will be compared. Lastly, the LSTM and TLSTM models will be compared to the same models with target replication [34]. Performance is measured using the AUC and F1 scores of the model predictions. In addition to this, subgroup discovery will be performed. Subgroup discovery is the act of forming groups using simple rules based on a given set of inputs with classes. These rules are based on feature values. Subgroup discovery will be performed using a decision tree. This results in the following research questions:

RQ1 *How well can machine learning models predict the necessary care for a patient having chest pains, using regular care data?*

RQ2 *How well does the Time-aware LSTM perform compared to the LSTM?*

RQ3 *Does integrating target replication into the T-LSTM and LSTM increase performance?*

RQ4 *How well do models that handle time-series data perform compared to models that do not?*

RQ5 *Are there any classes that can be identified using simple rules retrieved from performing subgroup discovery?*

The research is structured as follows. First, a few works related to the current research will be discussed. After this, the used dataset will be described and pre-processed on a per-table level. Next, the data will be processed on a global level. After this, the used models will be discussed. The next chapter discussed in-depth tuning of all models. Following this, the merging of similar classes will be discussed. Then, the results will be presented. Finally, the research questions will be answered and discussed and recommendations for future research will be made.

Chapter 2

Related work

In this chapter, other work related to the current research will be discussed. All selected works are applications of machine learning in the medical industry. Furthermore, the works are selected based on their similarity to the current research in their input data and/or used models.

Lipton et al. [34] use a Recurrent Neural Network (RNN) to diagnose intensive care unit (ICU) patients. The data used consisted of 10,401 cases, called "episodes". One episode consists of an irregularly sampled time series of 13 variables. Every episode also has zero or more (out of 128) diagnostic labels. The RNN will try to predict these labels. As an RNN requires a fixed number of inputs, the data was transformed into a time series with intervals of one hour. The mean is taken of all measurements within an interval. Next, forward- and back-filling are used to fill any intervals with no measurements. Forward- and back-filling means that if there is an interval with no data, it will be given the data of the previous or next interval. The paper by Lipton et al. is quite similar to the current research. The most significant difference is the number of available samples. Where Lipton has data in which patients are measured hourly, the patients in this research will likely be measured a lot less frequently, as these patients are not necessarily ICU patients.

Alsheref and Gomaa [6] use blood values to classify blood diseases using 8 different machine learning models. As input, they have the blood analysis of 668 patients, which each contain 28 variables. They use these to try to predict one of four different blood diseases, using 9 different machine learning techniques. For this task, the Support Vector Machine gave the worst accuracy with only 71.20%, whereas LogitBoost gave the highest accuracy with 98.16%. The study is similar to the current study in the fact that blood values are used to predict classes. However, the current research will have many more variables, as hematology values will also be added to the blood values. Furthermore, the research will not specifically be focused on blood-related diseases. This research will also examine using the patient's history as time series data, which means different models will be tested, as opposed to the research by Alsheref and Gomaa.

Gladding et al. [25] use hematology data to predict COVID-19, pneumonia, urinary tract infection, and heart failure. The models were evaluated using the AUC metric [33]. Separate models were created for each prediction goal. A boosted decision tree was used to predict COVID-19 and pneumonia, a random forest was used to predict urinary tract infection, and a logistic regression model was used to predict heart failure. On average, these models returned an AUC score of 0.75

on their validation sets. From these, urinary tract infection was the hardest to predict with an AUC of 0.68, and COVID-19 was the easiest to predict with an AUC of 0.8. The research by Gladding et al. is similar to the current study. The biggest difference between the two is the fact that in this research, models will be developed for multi-class prediction. Also, this research will examine models which use patients' history as time series data.

AlJame et al. [4] use routine care data to predict COVID-19. First, to impute missing values, a K-nearest neighbors-based imputation method was used [7]. To detect outliers, an isolation forest (iForest) was used [35]. Lastly, as the dataset was imbalanced (with 9.9% positive and 90.1% negative COVID-19 cases) SMOTE was used to balance the data [17]. A stacking model with two levels was used for prediction [48]. The first level consists of three classifiers: an extra trees model, a random forest, and a logistic regression model. The results from these three models are used by an XGBoost model. This model achieved an AUC of 99.38%. The research by AlJame et al. differs in a few aspects from the current study. The current research discusses a multi-class classification problem, while AlJame et al. discussed a binary classification problem. Also, AlJame et al. use no time series data, while the current research aims to do so.

Chapter 3

Dataset description

This chapter describes the dataset and all its tables. The dataset is provided by the University Medical Center Utrecht (UMCU). It is a subset of the Utrecht Patient Oriented Database (UPOD), which is a large relational database with patient information [11]. The dataset contains data of patients who have been referred to the hospital by a General Practitioner (GP) through a digital GP letter. These letters are also called "ZorgDomein" letters. The UMCU started to store these digitally in 2013. Therefore, we selected all patients that have been referred by the GP in 2013 or later. Another requirement for the group of patients is that they have been referred to the cardiology department for chest complaints.

3.1 General practitioner letters

The first table consists of all retrieved General Practitioner (GP) referral letters. This is primarily unstructured data, as a GP is not required to fill in a structured form. Instead, a free text input field is provided to them.

Patient_id	Date	Sex	Age	Text
p_1	d_1	1	40	...
p_2	d_2	0	45	...
p_3	d_3	1	55	...

Table 3.1: Structure of the GP letter data

Table 3.1 shows the structure of the data. Accompanying the free text comes some structured data. The date on which the letter is sent, and the patient's age and sex, are included in the table.

3.2 Hematology analyzers

The next table consists of values taken from hematology analyzers by Abbott. Hematology analyzers are able to provide a Complete Blood Count (CBC), which is a set of features giving information about the tested blood [20, 24]. The features produced by the analyzers are sent to the GLIMS software. GLIMS adds more features and gives warnings if the software thinks a determination is invalid.

These features are useful for practitioners when examining patients [20]. If any features change, this might be an indicator that the patient suffers from some kind of disease. Even though hematology analyzers have many applications like the aforementioned one, due to insufficient knowledge they are not used to their potential [19].

In the period from 2005 until now, two different hematology analyzers have been used: Saphire and Alinity. In 2020, the hospital started to use Alinity. The tables of both analyzers have a few groups of variables. Of these groups, the "c_b" variables are the actual features. An issue with the different hematology analyzers is that they do not measure the exact same "c_b" features.

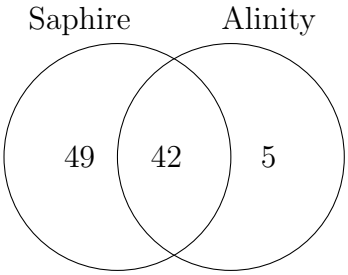


Figure 3.1: Venn diagram of how many features are measured by each analyzer

Figure 3.1 shows how many features are exclusive to each analyzer and how many features are measured by both. The figure shows that there are 49 features unique to Saphire, while there are 5 unique to Alinity. There is an overlap of 42 features.

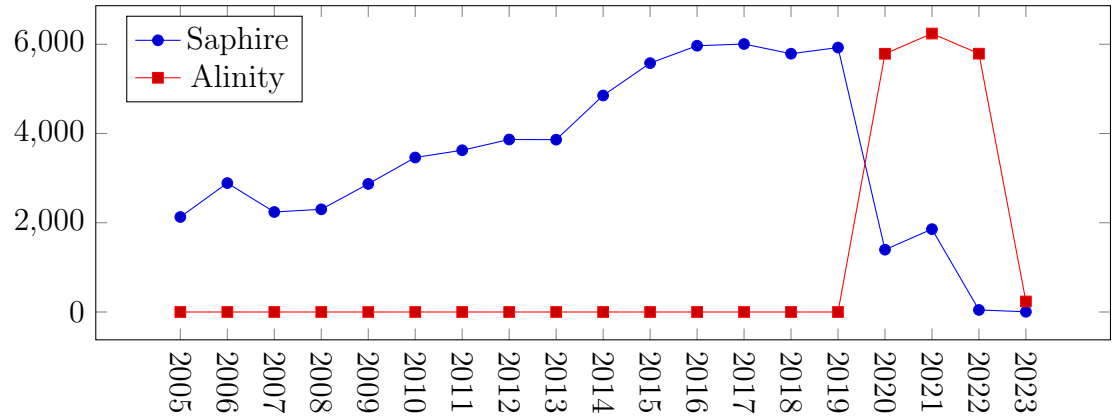


Figure 3.2: The number of determinations per hematology analyzer per year

Figure 3.2 shows how many determinations were taken using each analyzer from 2005 to 2023. The usage of Saphire is slowly being phased out. Therefore, it might not be a good idea to choose only the Saphire data, even though it has a lot more features. Hence, both the Saphire and Alinity data will be used. As 54 imputed features will probably create a lot of noise, only the 42 features which are measured by both Saphire and Alinity will be used.

Before merging the tables, rows containing errors will be removed. Another group of variables in the tables are the "c_s" variables. These are control variables that return 2 if anything has gone wrong with measuring the blood. Therefore, all rows which contain a "c_s" variable with the value 2 are dropped.

Date	Patient_id	Analyzer	Saphire ∩ Alinity: $x_1, x_2 \dots x_{42}$
d_1	p_1	0	...
d_2	p_1	0	...
d_3	p_2	1	...

Table 3.2: Structure of the hematology analyzer data

The Alinity and Saphire tables are merged vertically with only the c_b variables that are measured by both analyzers (Saphire ∩ Alinity), the date and patient_id are taken. An indicator variable is added where Saphire rows are given a 0 and Alinity rows are given a 1. The resulting dataset is structured as shown in table 3.2. A patient can have different determinations for different dates, or possibly no determinations at all. For each date and patient_id combination in the dataset, variables $(x_1, x_2, \dots, x_{42})$ are given.

Including the patient_id, date, and analyzer columns, there are 45 columns in the dataset. There are 82,479 rows in the merged hematology analyzer table. Of these rows, 3,987 are dropped as these have 41 or more missing columns. This leaves 78,492 rows. One of the advantages of data taken from a hematology analyzer is that it is very complete. This is because all variables are always tested in a single determination, whereas in laboratory determinations a doctor decides which variables will be tested [41]. This is also shown by counting the missingness in the table, which is 145,026. This means there is a missingness of only $\frac{145{,}026}{78{,}492 \cdot 45} \cdot 100 = 4.01$ percent.

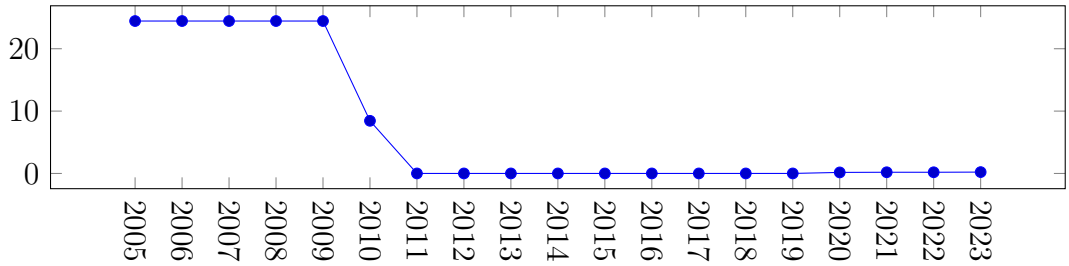


Figure 3.3: Missingness percentage in the merged hematology analyzer data from 2005 to 2023

Figure 3.3 shows the missingness percentage ($\frac{\text{missing}}{\text{missing+not missing}} \cdot 100$) per year. From 2005 till 2009 11 columns were missing in all rows, so it seems these have been added somewhere in 2010.

3.3 Laboratory determinations

The next set of features is taken from laboratory determinations. Just like the hematology analyzer data, the dataset consists of determinations for different patients on different dates. However, as can be seen in table 3.3, every variable has its own row. One determination consists of multiple features measured in a patient's blood. Which features, and how many are measured, differs per determination. In table 3.3, the first two rows have the same det_id. This means that both rows belong to

Det_id	Date	Patient_id	Variable_name	Variable_value	Lab_result_ok
1	d_1	p_1	Iron	...	1
1	d_1	p_1	Urea	...	1
2	d_3	p_2	Phosphate	...	0

Table 3.3: Structure of the laboratory determinations data

the same determination. As both rows belong to the same determination, they have the same date and belong to the same patient.

Before looking at the missingness, a few preprocessing steps must be performed on the table. Each row has the lab_result_ok variable. If this variable is 0, it means that the row is invalid. Therefore, all rows with lab_result_ok = 0 are removed. The variable_name column contains variable names in two different formats. These will be mapped to the same format by two mappings that were provided with the dataset: (format1 → uniform_format) and (format2 → uniform_format)). These mappings have been used on the variable_name column in the table. After this, all rows with NaN values were dropped. Lastly, the table is pivoted with det_id as the index, the names of variable_name as columns, and Variable_value as values for these columns. After these steps, a pivoted table with 182,791 rows and 1,590 columns is returned.

Date	Patient_id	$x_1, x_2 \dots x_{19}$
d_1	p_1	...
d_2	p_1	...
d_3	p_2	...

Table 3.4: Structure of the preprocessed lab determinations table

This means that there are 1,590 unique features in the laboratory determinations table. There are 290,637,690 missing values in this table, meaning that there is a missingness of $\frac{290,637,690}{182,791 \cdot 1,590} \cdot 100 = 99.48\%$. To reduce missingness, determinations taken on the same day from the same patient are merged. This means that if on the same day, there is a determination taken for iron and a separate one for urea, these will be merged as one determination. If there are two determinations on the same day for the same patient that measure one or more of the same features, the mean is taken. This reduced the number of rows to 115,054, the number of missing values to 181,546,483, and the missingness to $\frac{181,546,483}{115,054 \cdot 1,590} \cdot 100 = 99.24\%$. As most features rarely occur, only the ones that occur in at least 20% of the determinations are kept. This leaves 19 features. Figure 3.4 shows the structure of the table after these preprocessing steps.

3.4 Diagnosis-Treatment-Combinations (DTC)

The next table consists of Diagnosis-Treatment-Combinations (DTCs). Table 3.5 shows the structure of this table. For some interval i = (Date, End_date) and for some patient a, a diagnosis code is given together with the department in which the diagnosis code was assigned. There are 227,722 rows in the table and 30 different departments.

Date	End_date	Patient_id	Diagnosis_code	Department
d_1	d_4	p_1	...	Cardiology
d_2	d_5	p_1	...	Radiology
d_3	d_6	p_2	...	Cardiology

Table 3.5: Structure of the Diagnoses-Treatment-Combination's (DTC's). For every row, Date < End_date

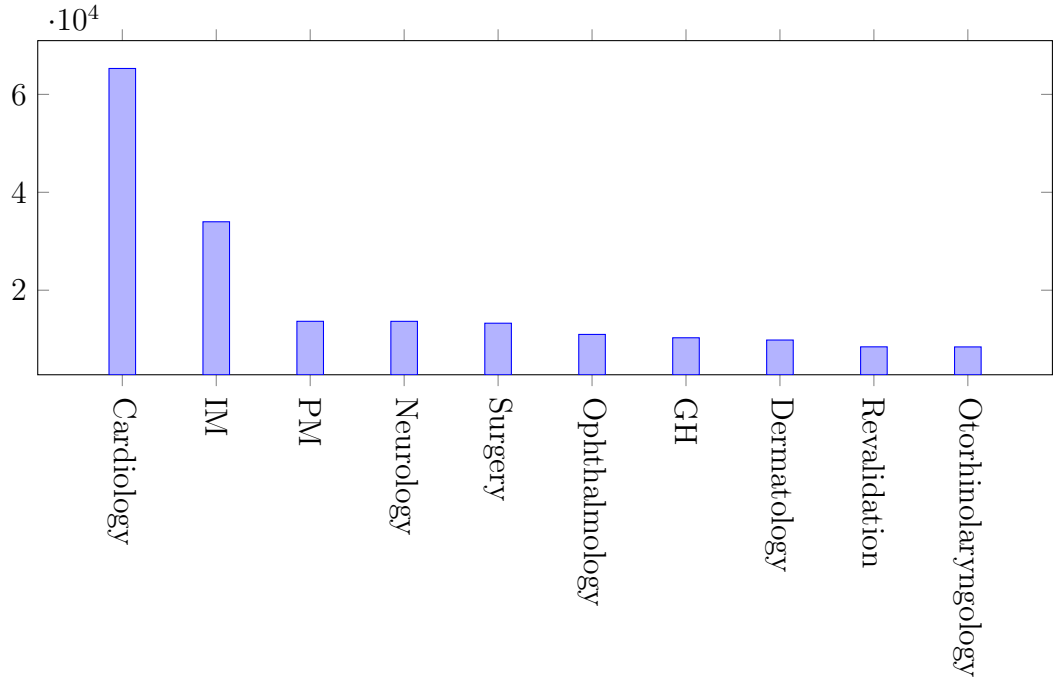


Figure 3.4: Frequency of the ten most frequent departments in the DTC table. Here, IM stands for Internal Medicine, PM stands for pulmonary Medicine, GH stands for Gastroenterology and Hepatology

Figure 3.4 shows the frequency of the ten most frequent departments in the DTC table. The cardiology department occurs the most, followed by Internal Medicine.

3.5 Actions

Patient_id	Date	Description	Type
p_1	d_1	Pacemaker	Operative
p_2	d_2	Repeat consultation	Consultation
p_3	d_3	E.C.G	E.C.G

Table 3.6: Structure of the actions table

Table 3.6 shows the structure of the actions table. It features any kind of "action" performed on a patient For example, consultations, medical tests like an ECG or MRI, and any operative actions are recorded here.

The actions table will be split up in two: one without operative actions and one with only operative actions.

3.6 Medication

Patient_id	Start_date	Stop_date	ATC code	Dose	If necessary
p_1	d_1	d_4	C01AA04	2	1
p_2	d_2	d_5	C10AA04	0	1
p_3	d_3	NaN	G10ZZ05	5	0

Table 3.7: Structure of the medications table

Table 3.7 shows the structure of the medications table. Here, there is a start and stop date for every medication taken. The ATC code tells which category a medicine belongs to [30]. It is a code consisting of a combination of seven digits and letters. Each digit or letter further specifies the category a medication is in. For example, if the code starts with C the medication is related to cardiovascular diseases. The dose column tells how much of the medication needs to be taken every day. If the dose is 0 this says that it's optional. Similarly, the if necessary column is 0 if the medication needs to be taken and 1 if it can be taken whenever the patient needs it. The medication table has 490,816 rows.

3.7 Measurements

Session_id	Date	Patient_id	Variable_name	Variable_value
1	d_1	p_1	Weight	...
1	d_1	p_1	Height	...

Table 3.8: Structure of the measurement table

The measurements table features a variety of measurements like height, weight, BMI, and heart rate. As shown in table 3.8, it is structured like the laboratory determinations table, as every variable in one measurement is given its own row.

This table will be split into two tables. The first table, measurements$_1$ will feature height, weight, and BMI. So, the rows where "variable_name" is not weight, height or BMI are dropped. After this, the table is pivoted with session_id as the index, variable_name as the columns, and variable_value as the values of these columns. This leaves 35,515 rows. Here, weight is missing in only 684 rows, a missingness of $\frac{684}{35{,}515} \cdot 100 = 1.92\%$, and height is missing in 14,279 rows, a missingness of $\frac{14{,}279}{49{,}169} = 40.20\%$. BMI is imputed whenever it is NaN and height and weight are both available using formula 3.1.

$$BMI = \frac{weight}{(\frac{height}{100})^2} \tag{3.1}$$

The second table, measurements$_2$, will feature all other features like heart rate and blood pressure. These will be kept separate from the other table as weight, height, and BMI are less likely to change over time than a feature like heart rate. This table will be preprocessed similarly. First, all rows where "variable_name" is height, weight, or BMI are dropped. After this, the table is pivoted with session_id as the index, variable_name as the columns, and variable_value as the values of these columns. Then, only the features that occur in at least 20% of the rows are kept. This leaves nine features.

Chapter 4

Data preprocessing

In this chapter, the data will be preprocessed into a set of inputs and labels.

4.1 Treatment periods

For a few purposes which will be discussed later, primarily label creation, it is useful to know the periods in which a patient is being treated. Here, a treatment period is a period in which a patient regularly visits the hospital. A treatment period begins when a patient is referred to the hospital with a health problem and ends when a patient has been treated. As these treatment periods are not recorded, estimated periods will be constructed using most of the available data. The treatment periods will be constructed by making groups of hospital activities that take place close to each other.

The process starts by creating a set of events $\mathcal{T}$. This set of events is defined as the collection of the GP letter (GPL), Hematology Analyzer (HA), Laboratory Determinations (LD), Diagnosis-Treatment-Combinations (DTC), Actions (A), and Operations (OP) tables per patient. So:

$$\mathcal{T} = \text{GPL} \cup \text{HA} \cup \text{LD} \cup \text{DTC} \cup \text{A} \cup \text{OP}$$

Set $\mathcal{T}_p$ is a subset of $\mathcal{T}$, with specifically only the events of patient p. The goal is to split each $\mathcal{T}_p$ into subsets $\tau_{p_1} \dots \tau_{p_n}$ such that each set of events τ_{p_i} consists of the events happening in one treatment period. Each item $t \in \tau_{p_i}$ has two attributes: $t.type$ tells which of the sets of events t originally came from and $t.date$ tells the date of t. The subsets are created as follows. First, $\mathcal{T}_p$ is sorted on date in ascending order. Let t_1 be the element with the earliest date in $\mathcal{T}_p$. The element is added to the first set of p: $\tau_{p_1} = \{t_1\}$. After this, all elements are looped over in order. If the distance in days between t_j and t_{j-1} is smaller or equal to $M(t_{j-1}, t_j)$, t_j is added to the set t_{j-1} is in. $M(x, y)$ is a function that takes two events and returns how many days can be between x and y to add y to the same set as x. It is defined as follows:

$$M(x,y) = \begin{cases} 180 & \text{If } x.date < y.date \wedge x.type = \text{GPL} \\ 180 & \text{If } x.date < y.date \wedge y.type = \text{OP} \\ 60 & \text{Otherwise} \end{cases}$$

Let $D(x, y)$ be a function that returns the difference between the date of item x and y in days. Also, let t_{j-1} be an element of τ_{p_i}. Then, if $D(t_{j-1}, t_j) \leq M(t_{j-1}, t_j)$: $\tau_{p_i} = t_j \cup \tau_{p_i}$. Otherwise, a new set will be defined: $\tau_{p_{i+1}} = \{t_j\}$. This has to be done for all $t_j \in \mathcal{T}_p$ and for all patients p.

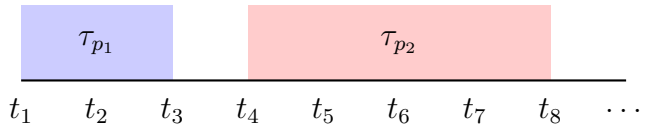


Figure 4.1: Resulting timeline

Figure 4.1 shows an example. Here, the first item of τ_{p_1} took place on t_1 and the last item took place on t_3. The distance in days between t_3 and t_4 was bigger than $M(t_2, t_3)$, which caused a new set to be defined.

Patient_id	Start date	End date
p_1	04-06-2021	04-08-2021
p_1	01-02-2022	09-03-2022
p_2	14-06-2009	04-08-2009

Table 4.1: Treatment periods structure, where Start date $\leq$ End date

Then, for each τ_{pi} the first (also earliest) and the last (also latest) elements will be taken. The first element will be the start date of the period and the last element will be the end date of the period. The periods are stored as in table 4.1.

4.2 General Practitioner letter selection

In this section, all GP letters that have at least d hematology determinations occurring before the end of their treatment period will be selected as inputs. Here, the selection is based on how much hematology data there is of a patient instead of lab data because the hematology data has less missingness.

Each GP letter G of patient p will be considered as candidate input. First, the treatment period τ_{p_i} in which G falls will be taken. Let l_i be the date of the latest element in τ_{p_i}. Table H_G contains all rows in the hematology table with patient p and before or at date l_i will be taken. If $|H_G| \geq d$, G will be an input, otherwise it is dropped.

Figure 4.2 shows the number of letters for which $|H_G| \geq d$. The higher d gets, the fewer letters satisfy the condition. To strike a balance between keeping as many letters as possible and being able to train a time series model, $d = 4$ is chosen. This leaves 2,717 letters, which will be used as input. For the time series models, $d = 20$ was chosen as an upper bound. This is the highest number for which 1000 letters still had determinations. Every letter for which $|H_G| < 20$ will be zero-padded. meaning that the remaining rows will be filled with zeroes.

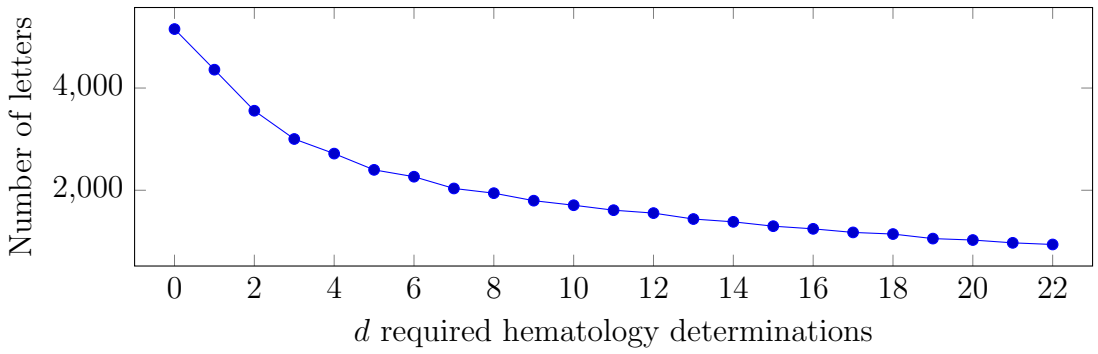


Figure 4.2: Number of letters that have d hematology determinations occurring before the end of the treatment period

4.2.1 Selected letters descriptives

This section shows a few descriptive statistics of the patients of the chosen 2,717 letters.

	Male	Female	Both
Mean age	65.66	64.74	65.19
Number of patients	1,341	1,376	2,717

Table 4.2: Mean age and number of patients by gender

Table 4.2 shows the mean age of all patients separated by gender. It seems males are a bit older than females. There is almost a fifty-fifty split in gender, there are only 35 more females.

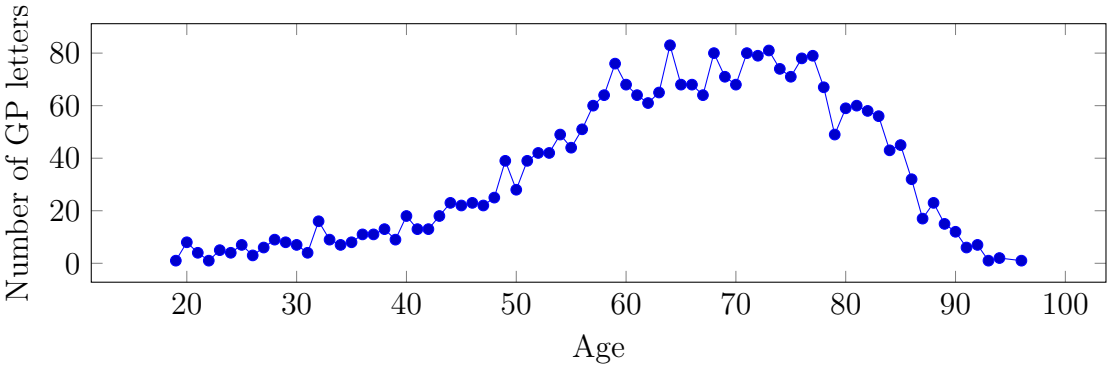


Figure 4.3: Number of included GP letters per age of patients at the time the letter was sent

Figure 4.3 shows how many of the selected patients are a certain age. The figure shows that most patients are about 60 to 80 years old. It makes sense that there are a relatively more younger people, given that the older someone is, the more chance that person has to have had enough hematology analyzer values.

Figure 4.4 shows the number of selected GP letters per year. As the hospital started to use ZorgDomein in 2013, it makes sense that in the following years the number of GP letters keep rising. This stabilizes starting with 2016.

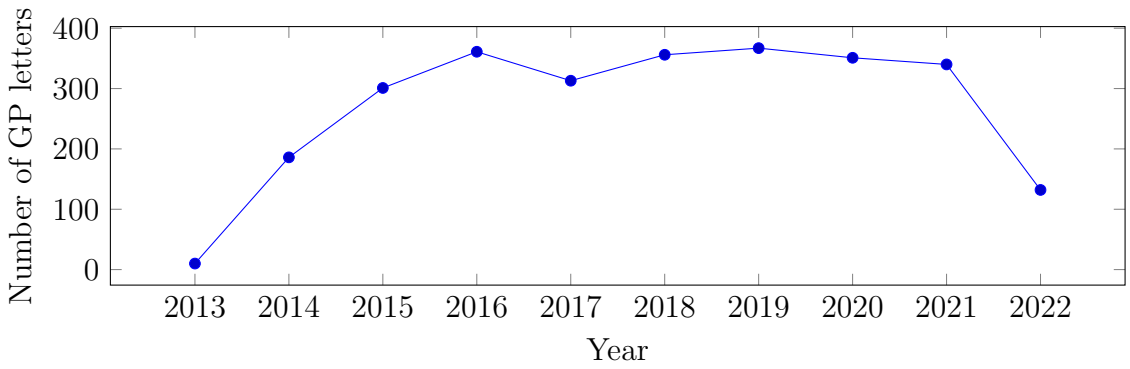


Figure 4.4: Number of included GP letters per year

4.3 Text mining GP letters

As section 3.1 shows, the GP letters mostly consist of unstructured data. As unstructured data cannot be put into machine learning models, it has to be transformed into structured data first. To do this, the Medical Concept Annotation Toolkit (MedCAT) will be used [31]. MedCAT is a machine learning algorithm that is able to recognize concepts in unstructured texts. It is chosen as it is specifically developed for the medical field and as there is support for the Dutch language [22]. To recognize those concepts, the algorithm needs a concept database. In this case, such a database consists of Concept Unique Identifiers (CUI) and words that describe them [36]. A CUI is simply an identifier that starts with a C and is followed by seven digits. This identifier then refers to a specific disease or another kind of medical concept. The MedCAT algorithm uses the words that describe CUI's to annotate texts with said CUI's. The algorithm can annotate these texts without having any predefined labels, meaning that it is unsupervised. As the provided GP letters are all in Dutch, a model using MedCAT, trained on a Dutch corpus will be used [22]. This model extends MedCAT by including negation detection. Whereas MedCAT only returns CUI's that the algorithm thinks occur in the text, the negation detection model provides a score between -1 and 1 for each CUI. Here, -1 means the model is very certain that some CUI does not apply to the patient and 1 means the opposite.

The negation detection model was trained on the 2,717 letters selected in 4.2. After this, all CUIs with their score (ranging from -1 to 1) were extracted from each text. On all letters together, 52 unique CUIs were found. A table was constructed with 2,717 rows (one for each letter) and 52 columns (one for each CUI). For each letter, the CUI columns are given values as returned by the negation detection model. If the model did not return any value for that specific letter and CUI, a score of 0 was given.

As some CUIs might be similar in meaning, a technique similar to word2vec, cui2vec, was used [10]. The technique is able to learn word embeddings of medical concepts, specifically CUI's. With word embeddings, the meaning of a word is distributed over multiple components. In this research, the pre-trained embeddings provided by Beam et al. [10] are used. These pre-trained embeddings contain 108,477 medical concepts, expressed in vectors of 500 components.

For every CUI, their embedding will be taken. Then, per GP letter, the mean of the embeddings will be taken, weighted by score. However, of the 52 unique CUIs,

there are 9 that did not have pre-trained embeddings. These will be left as is. Let E be the set of all pre-trained embeddings and M be the set of all CUIs mined by the MedCAT negation detection algorithm. Then $C = E \cap M$ is the set of all CUIs, mined by the algorithm, that have an embedding. Let S_{lc} be the score given at letter l to CUI c and let B_c be the embedding vector of CUI c. Then the embedding D_l based on all CUI's in C of GP letter l is defined as:

$$D_l = \left(\sum_{c=1}^{|C|} S_{lc} \right)^{-1} \sum_{c=1}^{|C|} S_{lc} B_c$$

Then D_l is concatenated with all scores of the CUI's that do not have an embedding (set $E \setminus M$), making for a vector embedding with size 509.

Patient_id	Date	Sex	Age	$D_{l_1}, \ldots D_{l_{500}}$	$CUI_{l_1}, \ldots CUI_{l_9}$
p_1	d_1	1	40	...	...
p_2	d_2	0	45	...	...
p_3	d_3	1	55	...	...

Table 4.3: Structure of the GP letters table after text mining. Here, l stands for which GP letter and differs in every row

This vector embedding is merged with the GP letter table, which structure is shown in table 3.1, returning the table as shown in table 4.3. In this structure, every row has a weighted embedding of the CUI's in C, denoted as $D_{l_1}, \ldots D_{l_{500}}$, in addition to the scores returned by the MedCAT negation algorithm of the CUI's that do not have a pre-trained embedding, denoted as $CUI_{l_1}, \ldots CUI_{l_9}$.

4.4 Concatenation to create model inputs

In this section, the final inputs will be created by merging most of the previously explained tables. The process starts by taking the GP letter table after preprocessing, as defined in section 4.3. Afterward, the data is split into a training set and a testing set with a 0.75/0.25 split.

4.4.1 Dimensionality reduction of GP letter embeddings

The pre-trained embeddings used in section 4.3 to define the GP embeddings are supposed to be able to differentiate between 108,477 CUI's. As the MedCAT algorithm only found 43 CUI's, it seems reasonable that a large number of the 500 components in their embedding vectors are unnecessary. Thus, it seems reasonable to assume that dimensionality reduction could work quite well. So, Principal Component Analysis (PCA) [15] will be applied to the 509-dimensional vector embeddings. First, to scale the data, the `sklearn.preprocessing.StandardScaler` function in Python is used. This function is first fitted on the training set. Denote the resulting matrix by X. The function calculates the mean $\bar{X}_f$ and standard deviation σ_f for each feature f of X separately [43]. After this, each value of a dataset X is transformed as follows:

$$X'_{if} = \frac{X_{if} - \bar{X}_f}{\sigma_f} \; \forall i \in \{1, \ldots, N\} \; \forall f \in \{1, \ldots, F\}$$

Here, N is the number of inputs in X, and F is the number of features in X. After the scaling function is fitted on the training set, the train and test set are both transformed.

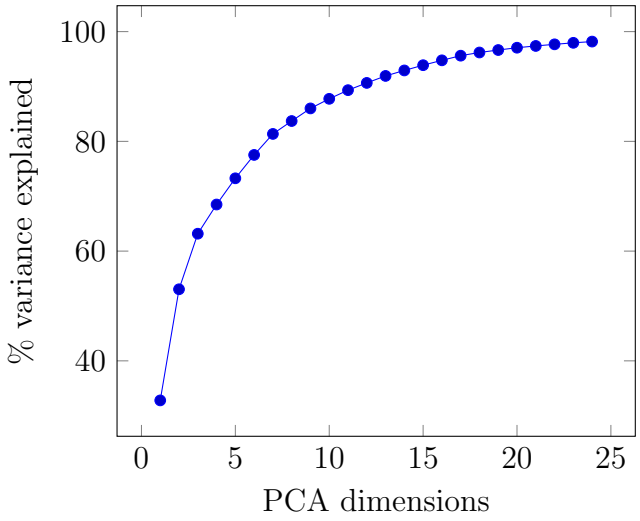


Figure 4.5: Percentage variance explained plotted against dimensions used for PCA for the GP embeddings

Next, the `sklearn.decomposition.PCA` function is used to perform the Principal Component Analysis (PCA). The PCA function is fitted on the training set after which both the training and testing set are transformed. Figure 4.5 shows how much variance is explained for each number of dimensions used in PCA. The variance explained is calculated by summing the explained variance of all components. From this, 17 dimensions are chosen as this is the smallest number of dimensions that has more than 95% variance explained, with 95.61% explained variance.

4.4.2 Combining inputs

In this section, most of the previously described tables will be combined into input sets. The steps described in this section will be performed on both the training and testing sets. First, each GP letter will be matched with the treatment period it is in. Then, the table and the hematology analyzer table are joined on patient_id. All rows for which the date from the hematology table is later than the end date of the treatment period will be dropped. For each of the rows, rows from the laboratory determinations table and the measurements table for the same patient will be searched. The rows which are the nearest in date to the date on which the row's hematology determination has been taken will be selected. Note that, as defined in section 3.7, the measurements table is split into measurments$_1$, with height, weight, and BMI, and measurements$_2$, with the other variables. The laboratory determinations and measurements$_2$ rows have to be within 60 days of the date of the hematology measurement of the current row, while the measurements$_1$ rows have to be within 5 years, as height, weight, and BMI do not change that much.

4.5 Determination of classes

To be able to predict the treatment a patient will receive, each input will need one or more classes assigned to them. These classes have to be created from available data and will be based on what treatment a patient has received. To create these, the treatment periods as described in section 4.1 will be used. Each treatment period will get its own labels, based on what happens in a period. For this research, a necessary treatment period is defined as one in which a patient has received medication and/ or an operation. To only keep medication that is given as treatment, the medication table will be filtered. Rows that abide by the rules defined in formula 4.1 and 4.2 will be kept.

$$(D(M_{\text{End\_date}}, M_{\text{Start\_date}}) > 15) \vee M_{\text{End\_date}} = \text{NaN} \tag{4.1}$$

$$M_{\text{Dose}} > 0 \wedge M_{\text{If necessary}} \neq 1 \tag{4.2}$$

Here, $D(x, y)$ returns the distance in days between x and y. So the first rule, shown in formula 4.1, requires a medication to have been taken for at least 16 days or that the End_date is not available. If the End_date is not available, it is assumed that the patient is still taking the medicine. All rows also have to abide by the rule shown in formula 4.2. This rule requires each medication row to have at least a dose of 1, and it requires If necessary to not be 1.

If a patient has not received any medication/ or an operation, they will be given the label "none". In essence, the "none" label means that someone has been examined at the hospital but did not receive any treatment. The patients that have received medication or an operation will be given the label of the department they were treated at. As the hospital letters and DTC tables are the only tables that directly mention a department, the department that occurs most in the rows that occur in the treatment period will be selected as the label.

So, this would mean that every treatment period gets the department they were treated at as a label or "None". All patients in this research are initially referred to the cardiology department. As it makes sense that if patients come into a certain department, they are treated at that department, it is likely that most of the patients have been treated at the cardiology department. To provide more detail on the necessary treatment of the patients in the cardiology department, there will be more than one class related to the cardiology department. This will be done by looking at what kind of operations and medication patients receive in their treatment period at the cardiology department. The operations table has a lot of row descriptions that point to the same operation. A mapping was provided which maps 467 of these descriptions to only 15 cardiology operations. This mapping was performed on the operations table. For the medications table, it is possible to create medication groups by looking at the ATC codes. For this research, an assumption is made that only medication for which their ATC code starts with "C" or "B01" is given to treat diseases related to cardiology. The "C" medications are further put into groups by looking at the first two digits after the "C". These range from 01, 02,. . . ,09,10, meaning that there are 10 groups of "C" medication, and one group of B medication, B01. Together, this makes 11 medication groups.

The process is summarized in figure 4.6. If a patient is treated at the cardiology department, that treatment period gets one or more labels, so, it becomes a multi-

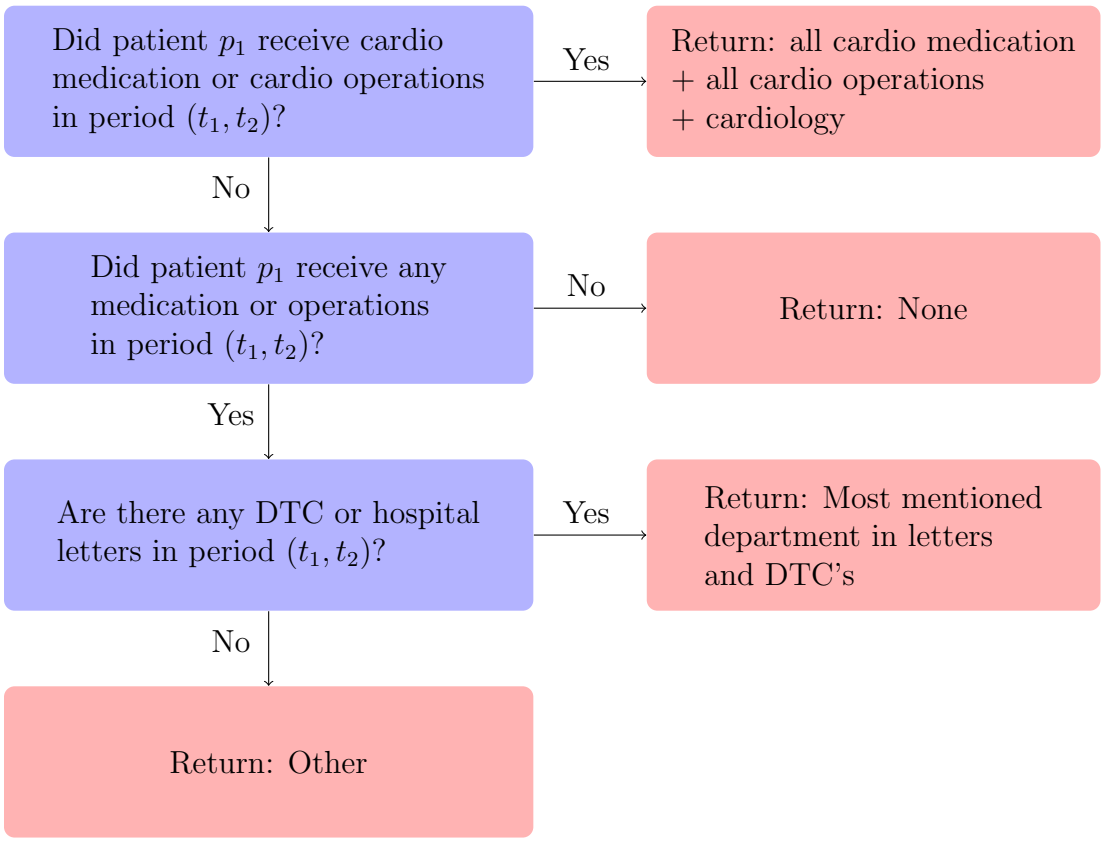


Figure 4.6: How labels are decided for treatment period (t_1, t_2) of patient p_1

label problem. Now every treatment period has one or more labels. To each period, the labels of the previous periods of the same patient will be added as input. This will be done cumulatively. This loses the order of events but gains the fact that at every time step the machine learning models will be able to use the patient's history. For example, if a patient has received a pacemaker in the past they are unlikely to receive one again. The cumulative is defined as in formula 4.3. Here, let label(τ_{p_i}) be the set of labels of the i'th treatment period of patient p.

$$\text{input}(\tau_{pi}) = \bigcup_{j=0}^{i-1} \text{label}(\tau_{pj}) \tag{4.3}$$

Formula 4.3, which will be added to each time steps input, is the union of all previous labels of patient p. The labels are converted to one hot encoding, to be able to use it as input for the models.

4.6 Merging model inputs with treatment periods

As a last step, the inputs created in section 4.4 will be merged with the treatment period input and labels as created in section 4.5. Each row in the input table as created in section 4.4 has a date based on a lab measurement. As these dates were used to create the treatment periods, each row will belong to exactly one treatment period. Each row will be merged with the input and assigned the labels of the treatment group it belongs to. Then, all missing values are imputed using

the `sklearn.IterativeImputer` function. This function is based on Multivariate Imputation by Chained Equations (MICE) [46]. All columns that have some missing values give rise to an additional column that tells if the variable was imputed (1) or not (0).

4.7 Preprocessing for time series models

For the T-LSTM model, for each sample, a list is created with the time passed between timestamps. So at timestamp d, the distance in days between d and $d-1$ is added. This is also added as a feature to the model.

d	S	A	D	M(12)	HA(43)	GPL(17)	LD(19)	PL(30)	MS(47)
1	...	...	...	...	...	...	...	...	...
2	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
d_p	...	...	...	...	...	...	...	...	...

Table 4.4: Structure of the input for the time series models. $x(n)$ means that x has n features. The following abbreviations are used: S means sex, A means age, D means time passed since previous timestep, M means measurements, HA means hematology analyzer, GP means GP letter, LD means lab determination, PL refers to previous labels (the cumulative of all labels given to each time before it), and MS refers to the missingness indicators

Table 4.4 shows the inputs created in section 4.6 for each patient p. In total, there are 171 features for each time step. The final preprocessing step depends on d_p, which is the number of hematology determinations p has. If $d_p = 20$, the input is left as is. If $d_p > 20$, only the most recent 20 inputs are taken, and the inputs for which $d > 20$ are dropped. If $d_p < 20$, the input is zero-padded, so $d_p - 20$ arrays of zeroes are added after d_p.

4.8 Preprocessing for non-time-series models

For the non-time-series models, first, the data of the final time step is taken. Then, over the other time steps, the mean, max, and min are calculated for each feature and added. This creates an input of $171 \cdot 4 = 684$ features. As there might be some features that do not add any information like the mean over age or minimum and maximum over missingness, a novel method was developed to reduce features without losing information. The idea is to take the mean of highly correlated features, as highly correlated features should be able to construct each other. This will be done using a graph.

Let G be a graph with a node for every feature. For every combination f_1, f_2 of features, the Pearson correlation on the training set is calculated. If this correlation is higher than 0.99, an edge between node f_1 and node f_2 is added to graph G. Let M be a set with the largest maximal clique in G. Then, all features inside M will be merged into a new feature, by taking the mean. After that, every feature in M is removed from G. This continues until there is no more clique in G.

This method reduces the number of features from 684 to 600. This method is similar to PCA, but its components will generally not consist of many different columns, which is good for interpretability.

Chapter 5

Models

This chapter discusses all models used in this research.

5.1 XGBoost

XGBoost [18] is a gradient tree-boosting algorithm. It is based on the idea of gradient boosting, which was proposed by Friedman et al. [23]. The idea of gradient boosting is to combine simple "base learners", into one complex model. Here, a simple function is one with few operations. Each base learner will be added to the model one by one. At each time step $1,\ldots,M$, a new base learner is added. This base learner will try to correct the mistakes made by the ensemble of the previous base learners. In the case of XGBoost, these base learners are Classification And Regression Trees (CART). These are similar to decision trees, with the difference that each tree returns a likelihood of a class rather than a specific class. A likelihood is assigned to all leaves of the tree. In XGBoost, the model prediction is defined as the sum of all likelihoods returned by each CART, as shown in formula 5.1.

$$F_M(x_i) = \sum_{m=1}^{M} f_m(x_i) \tag{5.1}$$

Here, F_M is the XGBoost model, $F_M(x_i)$ is the prediction made by the model of input x_i and each f_m for all $m \in \{0, 1, \ldots, M\}$ is a CART. The regularized objective function is defined as in formula 5.2

$$\sum_{i=1}^{I} L(y_i, F_M(x_i)) + \sum_{m=1}^{M} R(f_m) \tag{5.2}$$

Here, I is the number of samples, y_i is the label of the i'th sample, L is a loss function and R is a regularization function giving punishment for model complexity. In this formulation, the model has an ensemble of M base learners and has made M iterations. In each iteration, a new base learner is added. At step $M+1$, the current model is F_M. A new base learner will be added which minimizes the objective function 5.2. The minimization objective in step $M+1$ becomes formula 5.3.

$$\sum_{i=1}^{I} L(y_i, F_M(x_i) + F_{M+1}(x_i)) + \sum_{m=1}^{M} R(f_m) + R(F_{M+1}) \tag{5.3}$$

This optimization is solved by using the gradient for each input i of the loss function $L(y_i, F_M(x_i))$ to calculate the gain when splitting a leaf of the new base learner CART. The split with the highest possible gain is chosen. When all splits have a negative gain due to the regularization objective being larger splitting stops, the base learner is added to the model and the algorithm continues on to step $M+2$.

5.2 Support Vector Machines

A Support Vector Machine views all inputs as points in high-dimensional space [39]. The model tries to separate all classes in this space. In two-dimensional space and with binary classification, this would simply be a line separating the two. In three-dimensional space, this is a plane. In multidimensional space, however, classes will be separated using a hyperplane. Do note that in two-dimensional space, a line is considered a hyperplane and in three-dimensional space, a plane is considered a hyperplane. Hence, a hyperplane is a higher-dimension generalization of a line and a plane.

To explain SVMs, one first needs to understand how a linear classification model works. In the binary case, a linear model tries to find a hyperplane that separates inputs from class -1 from inputs from class 1. This hyperplane is defined as follows. Let w be the weight vector that will be trained, let x be an input vector, and let b be the bias. Then the linear separating hyperplane is defined as:

$$H : w^\intercal x + b = 0$$

Here, w and b are weights learned by the model. The model algorithm tries to optimize these weights such that for all vectors x which are class -1, $w^\intercal x + b < 0$, and for all vectors x which are class 1, $w^\intercal x + b \geq 0$. SVMs differ from linear classification in the following two aspects:

1. How weights w and b are optimized.

2. The separating hyperplane does not have to be linear.

The difference between the weight optimization of the linear classification models and SVMs comes from the difference in the loss function. In the linear classification model, the loss function is based on all classified points. SVMs, however, base their loss function on only a few input vectors, called support vectors. The idea is to maximize the distance between the decision boundary (the separating hyperplane) and these support vectors.

To this end, two hyperplanes H_- and H_+ parallel to the decision boundary are defined:

$$H_- : w^\intercal x + b = -1$$
$$H_+ : w^\intercal x + b = 1$$

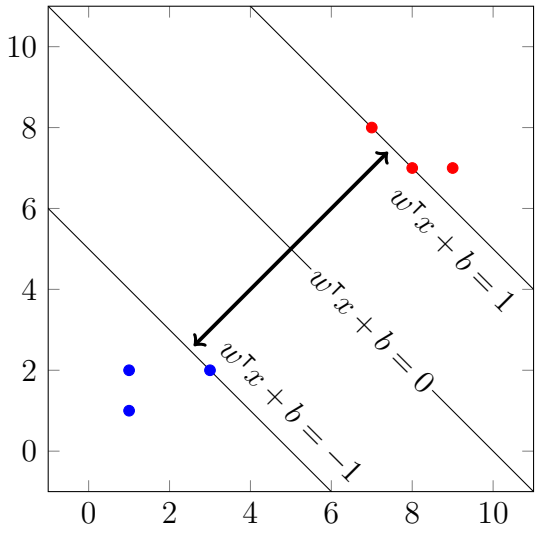


Figure 5.1: SVM example

The smallest possible distance from H_- to H_+ is called the margin. An example of this is shown in figure 5.1. SVMs try to define w and b, such that the separating hyperplane $w^\intercal x + b = 0$ is placed in a way such that the margin is as large as possible. The size of this margin is $\frac{2}{||w||}$. This can be rewritten into a minimization problem: $\min_{w,b} \frac{1}{2}||w||^2$. In the hard margin SVM case, all points have to be either on H_- or H_+ or outside the margin. This can be written in the following constraint: $y_i(w^\intercal x_i + b) \geq 1$ for all $i = 1, \ldots, n$. The points that are on H_- or H_+ are called support vectors.

In the soft margin case, the $y_i(w^\intercal x + b) \geq 1$ hard constraint will be relaxed into a soft constraint. To this end, a slack variable ξ_i is introduced. The formulation becomes as follows:

$$\min_{w,b} \frac{1}{2}||w||^2 + C\sum_{i}^{n} \xi_i$$
$$\text{S.T. } y_i(w^\intercal x_i + b) \geq 1 - \xi_i \; \forall i = 1, \ldots, n$$
$$\xi_i \geq 0 \; \forall i = 1, \ldots, n$$

In this new formulation, one is punished for putting a point inside the margin, instead of strictly disallowing it. The C parameter decides how much the model is punished for wrongly classifying inputs. As the C parameter increases, the margin will decrease and the number of wrongly classified inputs in the set of inputs you are training on will decrease. If these inputs that are wrongly classified with a smaller C are noise, it would be best for generalization to keep this C smaller.

Another difference with standard linear classification is the fact that the separating hyperplane does not have to be linear. This opens up a plethora of options for datasets featuring non-linear patterns.

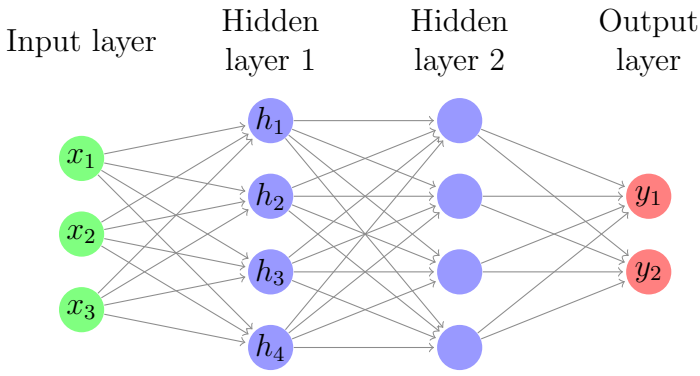


Figure 5.2: Fully connected neural network

5.3 Neural Networks

A standard Neural Network consists of sets of nodes. These sets are also called layers. The first layer is the input layer. Each feature of the input is given one node in the input layer. So for example in figure 5.2, the input has 3 features x_1, x_2 and x_3. These features are fed into the network at their corresponding node. Then, the next layer is formed by taking different weighted combinations of x_1, x_2,x_3, and a bias term. Each connection between the input and hidden layer is given a different weight. So, for example h_1 is defined as $h_1 = x_1w_1 + x_2w_2 + x_3w_3 + b_{h_1}$, where each w_i is a weight and b_{h_1} is the bias term. This can be generalized as follows. Let p be a layer preceding layer c. Let $w_{p_ic_j}$ be the weight between the neuron i of layer p and neuron j of layer c and let p be the layer before c. Then the value of neuron c_j is defined as in formula 5.4.

$$c_j = f\left(b_{c_j} + \sum_{i=0}^{|p|} w_{p_ic_j}p_i\right) \tag{5.4}$$

$$\text{ReLU}(x) = \max(0, x) \tag{5.5}$$

$$\sigma(\hat{y}_i) = \frac{e^{\hat{y}_i}}{\sum_j^{|\hat{y}|} e^{\hat{y}_j}} \tag{5.6}$$

Note that every neuron is given its own bias term. Formula 5.4 also features f, which is a non-linear activation function. Using f on the weighted combination allows non-linear relationships to be modeled. An example of such a function is ReLU, as shown in formula 5.5. All neuron values of the layers after the input layer are defined as shown in formula 5.4. An exception to this is the output layer, on which another function is performed after calculating their activation values using formula 5.4. This function is often softmax, as defined in formula 5.6. This function transforms the outputs into a probability distribution, where all output neurons add up to 1. Each output neuron then gets a fraction of 1 of how likely the model thinks it is the class assigned to that neuron. Therefore, the size of the output layer is defined by the number of output classes. The example shown in figure 5.2 shows only two output classes.

$$\sum_{i=0}^{I} L(y_i, \sigma(\hat{y}_i)) \tag{5.7}$$

$$\frac{\partial \sum_{i=0}^{I} L(y_i, \sigma(\hat{y}_i))}{\partial w_k} \tag{5.8}$$

$$w_k = w_k - \eta \frac{\partial \sum_{i=0}^{I} L(y_i, \sigma(\hat{y}_i))}{\partial w_k} \tag{5.9}$$

Neural network weights are randomly initialized but updated using gradient descent, which works as follows. The loss of a neural network with I inputs is defined as shown in formula 5.7. Here y_i is defined as the label for input $i \in \{0, 1, \ldots, I\}$, and L is a loss function. Let the weights of layer k be w_k. Formula 5.8 shows the impact of w_k on the loss function, by taking the partial derivative of the loss function with respect to w_k. To minimize the loss function, the weights are updated in the negative direction of the gradient. This is done with a rate of η, which is also called the learning rate. Updating the weights of layer k is shown in 5.9. Each time the weights are updated for all inputs is called an epoch.

5.4 Recurrent Neural Networks

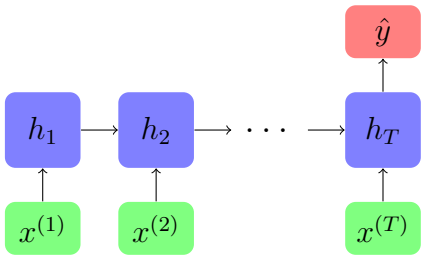


Figure 5.3: Recurrent Neural Network

A Recurrent Neural Network (RNN) is a model designed to deal with time-series data [45]. An RNN consists of layers, which are connected via weights. As input, you have the same n variables but sampled at T different points in time. More formally, you have T vectors $(x^{(1)}, x^{(2)}, \ldots, x^{(T)})$, and each vector $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$ has n features. An example of this is shown in figure 5.3. The input vectors are put one by one into the hidden vector. Similar to neural networks, inside the hidden layer, the inputs are multiplied by weights, and an activation function is used on the resulting values. An RNN differs, however, in the fact that starting with h_2 the output values are determined by the input and the previous layer. The values calculated by the last layer are the values used by the output layer. The values from the previous layer are also called the hidden state. Similar to neural networks, the weights are also randomly initialized and optimized, as shown in formula 5.9.

5.4.1 LSTM

As an RNN suffers from the vanishing gradient problem, different RNN variants have been developed to combat this [45]. An RNN variant changes what calculations are

being performed in the hidden layer. The Long Short-Term Memory (LSTM) model is one of the most popular ones [27], hence LSTMs are used in this research.

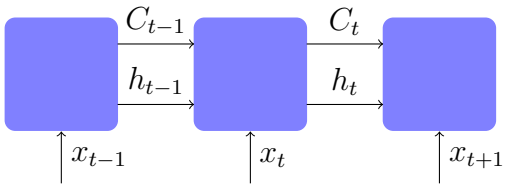


Figure 5.4: LSTM

The LSTM model adds a cell state C_t, separate from the hidden state h_t, which keeps track of long-term information. Figure 5.4 shows an example. In the LSTM cell, the blue boxes in the figure, multiplications, and additions using the model weights are performed to calculate the next cell state and hidden state. The cell state first goes through the forget gate, which decides what the model should forget and remember through weights. After that, information is added to the cell state from the input and previous hidden state. Lastly, the cell state is transformed using the previous hidden state and an activation function into the next hidden state. The hidden state is what is used to predict the output label at each time step. So the hidden state can be seen as what was predicted in the previous step, while the cell state can be seen as what is known about the previous time steps.

5.4.2 Time-Aware LSTM

Baytas et al. [8] proposed an alternative to the LSTM model: the Time-Aware LSTM model (T-LSTM). A regular LSTM (and RNN) assumes that the same time has passed between $x^{(t)}$ and $x^{(t+1)}$ as between $x^{(t+1)}$ and $x^{(t+2)}$. The T-LSTM model does not assume this and incorporates the elapsed time between time steps in the model. For each time step, in addition to an input vector $x^{(t)}$, the model requires Δ_t, which is the time that has passed from $x^{(t-1)}$ to $x^{(t)}$.

When compared to the LSTM, the differences occur in handling the previous memory cell. The T-LSTM model splits the incoming memory values into short- and long-term memory. The short-term memory is discounted using the value of Δ_t. This retains a patient's global profile while removing temporary aspects from the profile. The adjustment is defined as follows. At step t, memory cell C_{t-1} is received by the T-LSTM unit. Memory cell C_{t-1} is then decomposed into C^S_{t-1}, a short-term memory cell, and C^T_{t-1}, a long-term memory cell.

$$C^S_{t-1} = a(W_d C_{t-1} + b_d) \tag{5.10}$$

$$C^T_{t-1} = C_{t-1} - C^S_{t-1} \tag{5.11}$$

The short-term memory cell is defined as in formula 5.10. Here, a is an activation function, and W_d and b_d are learnable weights, where W_d is a vector of the size C_{t-1} and b_d is the bias term. The model tries to learn which dimensions of the vector belong to short-term memory and gives these higher weights. Formula 5.11 defines

the long-term memory, which is the difference between the previous memory cell and the decomposed short-term memory.

$$\hat{C}^S_{t-1} = C^S_{t-1} \cdot g(\Delta_t) \tag{5.12}$$

$$C^*_{t-1} = C^T_{t-1} + \hat{C}^S_{t-1} \tag{5.13}$$

The short term memory cell, C^S_{t-1}, is discounted by $g(\Delta_t)$, where g is a monotonically decreasing function. This means that the larger Δ_t becomes, the smaller $g(\Delta_t)$ becomes. The discounted short-term memory $\hat{C}^S_{t-1}$ is defined as the short-term memory times $g(\Delta_t)$, as defined in 5.12. Finally, the short-term memory and long-term memory are added together, forming the adjusted memory cell C^*_{t-1} as shown in formula 5.13. The rest of the model is the same as the LSTM model, only C^*_{t-1} is used instead of C_{t-1}. Compared to the LSTM model, the T-LSTM model has to learn weights W_d and b_d, which decompose the previous memory state, in addition to the other weights.

To conclude, the time-aware LSTM model becomes aware of time by discounting short-term effects, which accentuates long-term effects.

5.4.3 Target replication

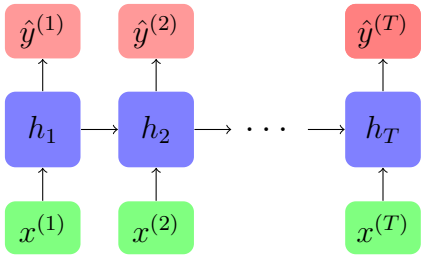


Figure 5.5: Many to many Recurrent Neural Network

Target replication is a technique proposed to improve the performance of RNN models [34]. It is made for many-to-one problems, where only a prediction is made at the end. The technique requires transforming the RNN into a many-to-many network. This means that at every timestamp, a prediction is made. An example of such a network is shown in figure 5.3. In a normal many-to-many network, all predictions are weighed equally. The idea of target replication is, however, to weigh the prediction that one wants to know, $\hat{y}^{(T)}$, compared to all the other predictions $\hat{y}^{(t)}$ where $t = 1, 2, \ldots, T-1$. Let $y^{(t)}$ be the actual labels for timestamp t and let $\text{loss}(\hat{y}^{(t)}, y^{(t)})$ be a loss function comparing the predicted values to the actual labels. Then, the loss function using target replication is defined as shown in formula 5.14.

$$\alpha \left(\frac{1}{T} \sum_{t=1}^{T-1} \text{loss}(\hat{y}^{(t)}, y^{(t)}) \right) + (1-\alpha) \left(\text{loss}(\hat{y}^{(T)}, y^{(T)}) \right) \tag{5.14}$$

Here, $\alpha \in [0, 1]$ is a hyperparameter that determines how much the predictions before the final predictions weigh. If $\alpha = 0$, the network is one without target replication. As α grows, the weight of the intermediate predictions increases.

Chapter 6

Hypertuning models

To provide the best possible performance, the models will be hyper-tuned. This means finding the values for which the models perform best. This chapter will describe which implementation of each model was used and the hyper-tuning process for each model.

6.1 XGBoost

The XGBoost implementation of the original authors has been used [18]. The parameters as proposed by Putatunda et al. [44] will be tuned. The n_estimators parameter defines the number of trees used in the model [21]. This parameter also defines the number of iterations of the model. The max_depth parameter limits the maximum depth of each tree. The colsample_bytree parameter is the fraction of features used when constructing a tree. The features that can be used are chosen randomly at each iteration of the model. It is a number between 0 and 1. The reg_lambda parameter determines the amount of L2 regularization on the trees. Lastly, the subsample parameter is defined as the fraction of training samples that will be used to train the model. Similar to colsample_bytree, in every iteration a fraction of training samples are randomly chosen. The parameter value is a number between 0 and 1. To hyper-tune the model, `hyperopt` is used [12]. Hyperopt has been chosen due to its performance when hyper parameterizing XGBoost [44].

Parameter	Values
n_estimators	{50,60,. . . ,200}
max_depth	{4,5,. . . , 16}
colsample_bytree	[0.5, 1.0]
reg_lambda	[0.0, 1.0]
subsample	[0.8, 1.0]

Table 6.1: XGB hyperparameters. Here all interval values are drawn from uniform distributions over these intervals

Table 6.1 shows the parameters which will be optimized by Hyperopt. These parameter ranges have been adopted from Putatunda et al. [44]. The n_estimators parameter will be between 50 and 200 in steps of 10, max_depth is an integer between

4 and 16, colsample_bytree is a fraction between 0.5 and 1, reg_lambda is between 0 and 1 and subsample is between 0.8 and 1. The Hyperopt implementation by Putatunda et al. [44] is taken as a base. The model will be optimized on the training data. The performance is validated using cross-validation with five folds. The optimization objective is changed to the weighted F1 score. Also, probabilities are predicted by the XGB classifier instead of directly predicting classes. In order to speed up the process, "gpu_hist" is used as tree_method. The Hyperopt optimizer is run for 30 trials.

6.2 SVM

For the Support Vector Machine (SVM) the `sklearn` implementation was used. More specifically, the `sklearn.svm.SVC` function. As a SVM does not support multi-label classification, the `sklearn.multioutput.MultiOutputClassifier` function will be used to extend the SVM classifier to a multi-label classifier. The `MultiOutputClassifier` function trains a separate model for each possible class. To select the optimal hyperparameters `hyperopt` was used. The kernel parameter, the C parameter, and kernel-specific parameters will be tuned. As discussed in section 5.2, the C parameter is a regularization parameter that tells the model how conservative it should be. A higher C value results in less misclassified training samples, but possibly a less generalized model which performs worse on the test set. The kernel parameter decides what kind of functions can be used to separate the hyperplane. The linear kernel means that the hyperplane is strictly linear. The poly kernel makes the hyperplane any kind of polynomial function. The degree of that polynomial function is a hyperparameter. The rbf and sigmoid kernels have no exclusive hyperparameters. Every kernel, except for the linear kernel, also has the γ hyperparameter. The gamma hyperparameter influences how many samples should be included in defining the separating hyperplane, depending on their distance to that hyperplane. With a large γ, only the support vectors are included. When γ grows first the samples close to the hyperplane will be included. The larger it becomes, the more samples will be included.

Which bound	Equation	Solve for x
γ lower bound	$e^x = 2^{-15}$	≈ -10.397
γ upper bound	$e^x = 2^3$	≈ 2.079
C lower bound	$e^x = 2^{-5}$	≈ -3.465
C upper bound	$e^x = 2^{15}$	≈ 10.397

Table 6.2: SVM ranges translated

The tested hyperparameter values are mostly based on the guide by Hsu et al. [29]. Hsu et al. recommend using ranges with exponents of 2. As Hyperopt has an exponent range based on e instead, the upper and lower bounds of the exponent ranges will be translated. This translation is shown in table 6.2. The ranges are shown in 6.3. The model will be optimized on the training data. The performance is validated using cross-validation with five folds. The optimization objective is changed to the weighted F1 score. Also, probabilities are predicted by the SVM

Parameter	Values
Kernel	{linear, rbf, sigmoid, poly}
C	$[e^{-3.465}, e^{10.397}]$
γ	$[e^{-10.397}, e^{2.079}] \vee \{\text{auto} : \frac{1}{f}, \text{scale} : \frac{1}{f \cdot var(x)}\}$
Degree	$\{2, 3, 4, 5\}$

Table 6.3: SVM hyperparameters. Here, f is the number of features, x is the table of all inputs, and var(x) flattens x and computes the variance of the flattened vector

classifier instead of directly predicting classes. The Hyperopt optimizer is run for 30 trials.

6.3 Neural Network

For the standard neural network, the `TensorFlow` library is used. To hyper-tune the model, `keras-tuner` is used [40]. A few parts of the structure are fixed. The output layer will have the sigmoid activation function applied to it, as this is a multi-label classification problem. Sigmoid clips all values between 0 and 1, and it is a more sensible choice to use with multi-label classification compared to softmax. Using softmax, a probability distribution is returned which adds up to 1. With multi-label classification, however, it is also possible to predict more than one class or zero classes. Then softmax does not make sense as, for example, it needs to be possible for two classes to have a probability of 1 or for all classes to have a probability of 0. This is possible with sigmoid, as it clips all values independently. Adam is chosen as the model optimizer, as this is one of the most used optimizers [14].

The model will have an input and output layer, and there will be at least one dense layer plus dropout layer combination. However, the number of dense plus dropout layers will be hyper-tuned. A dense layer is also called a fully connected layer, meaning that every neuron in the layer has a connection and weight to the neurons in the layer before it. The size of the hidden layer is a hyperparameter and can be set to any positive integer. A dropout layer sits between layers and randomly sets a fraction of neurons of the previous layer to 0. This fraction is a hyperparameter and can be between 0 (no dropout) and 1 (all neurons are set to 0). The learning rate is the rate at which weights are adjusted as discussed in section 5.3. The batch size is the number of examples taken at once to perform gradient descent on. The loss function will also be hyper parameterized. The first option is binary cross-entropy, as defined in formula 6.1.

$$-\frac{1}{I}\sum_{i=1}^{I}\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right) \tag{6.1}$$

Here, y_i is the label of input i while $\hat{y}_i$ is the prediction of y_i by the model. The function can be split up into two parts, if $y_i = 1$, $\log(\hat{y}_i))$ is returned. As the prediction is put into a log, the closer it is to 1 the larger $\log(\hat{y}_i))$ becomes. The other part, if $y_i = 0$, is $\log(1 - \hat{y}_i)$. Because of the one minus, this has the same effect but for 0, the closer it gets to 0 the larger it becomes. As a minus is added

in front of the formula, it becomes a minimization objective. The second option for the loss function is the binary focal cross-entropy loss [32]. This loss function was introduced to combat class imbalance. Formula 6.2 shows the definition.

$$-\frac{1}{I}\sum_{i=1}^{I}(y_i(\alpha_f(1-y_i)^\gamma \log(\hat{y}_i)) + (1-y_i)((1-\alpha_f)(y_i)^\gamma \log(1-\hat{y}_i))) \tag{6.2}$$

It is similar to binary cross-entropy, only differing in the addition of α_f and γ. Parameter α_f, is a number between 0 and 1 and it weighs class 0 to class 1. The larger α_f is, the more labels of class 1 weigh and the fewer labels of class 0 weigh. Parameter γ lets predictions that are close to their label weigh less. The larger γ becomes, the more pronounced this effect is. As Lin et al. [32] suggest using $\gamma = 2$, hyper parameterization will be done in ranges around that number.

Parameter	Values
Number of dense/dropout layers	{1, 2, 3, 4, 5}
Units for each dense layer	{32, 64, 128, 256, 512, 1024}
Dropout for each dropout layer	{0.2, 0.3,. . . , 0.8}
Loss function	{Binary, Focal}
Alpha focal	{0.1, 0.2,. . . , 1}
Gamma focal	{2, 2.2,. . . , 3.4}
Learning rate	[0.01, 0.0001]
Batch size	{32,64,128,256}

Table 6.4: Neural network hyperparameters

The hyperparameters which will be tuned are shown in table 6.4. The number of dense plus dropout layers will vary between 1 and 5. Both the dense layers and dropout layers have a separate hyperparameter. Each dense layer has a separate number of units which can be one of 32, 64, 128, or 256. All dropout layers will have a separate dropout fraction, a number in {0,0.2,. . . ,1}.

The `BayesianOptimization` function of Keras tuner is used as an optimization algorithm. The optimization algorithm needs an objective, which is a way to decide what the best-tested hyperparameters are. Normally, it makes sense to choose the loss function. However, the loss function is a hyperparameter. As the results of the loss functions are not exactly comparable (one function being smaller than the other does not mean the one function is better), the weighted F1 score is chosen as objective instead. The training set is split into a training set and a validation set with a 0.75/0.25 split. The optimization algorithm runs for 30 trials which take 50 epochs at maximum. A trial stops early if the validation loss does not improve in 4 epochs. After finding the best hyperparameter values, the best number of epochs is found by running the model for 50 epochs and taking the epochs with the best weighted F1 score.

6.4 LSTM

The Long Short-Term Memory (LSTM) implementation of `TensorFlow` is used. The model consists of an LSTM layer, a dense layer, a dropout layer, and an output

layer. The dense layer has ReLU activation. The hyperparameters are explained in section 6.3. To hyper-tune the model, `keras-tuner` is used [40].

Parameter	Values
LSTM units	{32, 64, 128, 256, 512, 1024}
Dense layer units	{32, 64, 128, 256, 512, 1024}
Dropout	{0.2,0.3, . . . , 0.8}
Learning rate	[0.01, 0.0001]
Loss function	{Binary, Focal}
Alpha focal	{0.1,0.2,. . . , 1}
Gamma focal	{2, 2.2,. . . , 3.4}
Batch size	{32,64,128,256}
α (TR)	[0.1, 1]

Table 6.5: LSTM hyperparameters

Table 6.5 shows which hyperparameters will be tuned and what values will be tested for these. The same parameters will be tuned for the non-TR and TR models, except for the α (TR) parameter which is exclusive to the TR model. The α parameter is explained in section 5.4.3. The `BayesianOptimization` function of the Keras tuner is used as an optimization algorithm. The training set is split into a training set and a validation set with a 0.75/0.25 split. Again, the weighted F1 score is used as the optimization objective and the optimization algorithm is run for 30 trials. Each of those 30 trials runs for 50 epochs maximum. A trial stops early if the validation loss does not improve in 4 epochs.

6.5 Time Aware LSTM

The Time aware LSTM implementation of the original authors, Baytas et al., has been used [8]. The model consists of an LSTM layer, a dense layer, a dropout layer, and an output layer. The dense layer has ReLU activation. As defined in section 5.4.2, the model needs a monotonic decreasing discount function g. The original authors, Baytas et al. [8], recommend $g(\Delta_t) = \frac{1}{\log(e+\Delta_t)}$ for datasets which have long elapsed time between time steps, for example, days. As hematology determinations are not something that is regularly measured more than once per day, the difference between time steps is likely to be multiple days. Therefore, g is chosen to be $g(\Delta_t) = \frac{1}{\log(e+\Delta_t)}$. The hyperparameters are explained in section 6.3.

To hyper-tune the model, `hyperopt` is used [12].

Table 6.6 shows which hyperparameters will be tuned and what values will be tested for these. Compared to the LSTM models, the number of epochs is added as a hyperparameter. This is done due to the fact that the T-LSTM implementation cannot validate and train at the same time. This means for every number of epochs tested, you need to retrain the entire model. As this takes a lot of time, this is infeasible for the current research. The same parameters will be tuned for the non-TR and TR models, except for the α (TR) parameter which is exclusive to the TR model. The α parameter is explained in section 5.4.3. To hyper-tune the model,

Parameter	Values
LSTM units	{32, 64, 128, 256, 512, 1024}
Dense layer units	{32, 64, 128, 256, 512, 1024}
Dropout	{0.2,0.3, . . . , 0.8}
Learning rate	[0.01, 0.0001]
Loss function	{Binary, Focal}
Alpha focal	{0.1,0.2,. . . , 1}
Gamma focal	{2, 2.2,. . . , 3.4}
Batch size	{32,64,128,256}
α (TR)	[0.1, 1]
Epochs	{1,2,. . . ,50}

Table 6.6: T-LSTM hyperparameters

`hyperopt` is used [12]. Hyperopt was chosen as keras-tuner does not support TensorFlow 1.x while the model is programmed using TensorFlow 1.x. As focal loss was introduced in TensorFlow 2.x, it is not included as a hyperparameter. The training set is split up into a smaller training set and a validation set with a 0.75/0.25 split. Each parameter combination is trained on the smaller training set and validated on the validation set. The weighted F1 score is used as the optimization objective, and the optimizer is run for 30 trials.

6.6 Subgroup discovery

In this section, the setup and hyperparameter tuning of the performed subgroup discovery using decision trees will be discussed. Decision trees will be used, as these are good at the subgroup discovery task in the clinical domain, even surpassing algorithms specifically designed for the task [3]. As we have a multi-label problem, each class will get its own decision tree. The `DecisionTreeClassifier` function from the `sklearn` library will be used. For each decision tree, the class weight will be balanced to overcome the obstacle of class imbalance. The trees will be tuned as described by Abu-Hanna et al. [3]. The minimum samples in one leaf will be set to 3% of the training set, which is based on expert opinion [3]. The complexity of the trees will be tuned using 10-fold cross-validation, meaning that the tree will be pruned for different complexity values. The complexity values from the `cost_complexity_pruning_path` function in `sklearn` will be used as possible values for the hyperparameter, which returns all complexity values used in the trees pruning process [1]. For each model, the complexity value which has the highest average cross-validated accuracy will be taken.

Chapter 7

Merging similar classes

	Precision	Recall	F1-score	support
micro avg	0.42	0.58	0.48	2055
macro avg	0.19	0.20	0.17	2055
weighted avg	0.40	0.58	0.46	2055

Table 7.1: Averages of LSTM model performance on all 30 classes

After all preprocessing steps, there are 30 classes left. Table 7.1 shows the performance metrics of a hyper-tuned LSTM model on these 30 classes. It shows quite a low F1 score, especially when taking the unweighted average of the F1 score of all classes (macro average). Appendix A.1 shows the performance metrics of all classes separately. The medication classes all have low precision, most department classes are never even predicted, and most of the operation classes have low F1 scores. This might be due to classes being too similar to each other. For example, distinguishing whether someone needs C01 or C02 medication (both given to cardiology patients) using primarily blood samples seems hard. A solution to this might be to merge certain classes. For example, you could merge C01 and C02. That would mean the model would predict C01 $\vee$ C02 instead of specifically either of the two. Forming these groups might increase model performance. This chapter will research if making these kinds of groups will do so.

7.1 Theory

Currently, the labels can be separated into three groups: cardiology medication, cardiology operation, and department. This research will only explore merging labels within these three groups. This will be done by looking at which classes are most similar after training a machine learning model. For the machine learning model, the standard LSTM is chosen as it is the simplest model in this research which uses time-series input. The assumption the following algorithm makes is that classes that have similar inputs, will also have similar activations returned by the LSTM.

The training set will be split up into two sets, training_2 and validation with a 0.75 and 0.25 split respectively. The model will be trained on training_2. After training, validation will be analyzed in order to merge classes. The overall principle

is to look at pairs of classes x, y for which x has a high activation in samples with class y and vice versa. After all, similar activation will mean that many of the same neurons are activated, meaning that similar patterns are found by the models. Let $L = \{\{l_1\}, \{l_2\}, \ldots, \{l_n\}\}$ be a set of singleton sets of unique labels. Then, let $B(L, 2)$ be the set of all possible combinations of 2 items in L. Let $P(L, 2)$ be all possible permutations of 2 items in L. The classes l_1, l_2 which will be merged, are the classes that minimize the following formula:

$$\mathcal{M}(L, X) = \underset{\{c_1, c_2\} \in B(L,2)}{\operatorname{argmin}} \frac{1}{|P(c_1 \cup c_2, 2)|} \sum_{\{p_1, p_2\} \in P(c_1 \cup c_2, 2)} \mathcal{A}(p_1, p_2, X) \tag{7.1}$$

$$\mathcal{A}(p_1, p_2, X) = \frac{1}{|X_{p_1}|} \sum_{x \in X_{p_1}} |x_{p_1} - x_{p_2}| \tag{7.2}$$

Here, X is the output of the LSTM model and X_c are the outputs of all samples that have class c. Also, each $x \in X_c$ is a sample and x_c is the activation value for class c in sample x. Function $\mathcal{A}(p_1, p_2)$, as defined in formula 7.2, returns how similar the activation of p_2 is to the activation of p_1 at samples of p_1. For each sample, the absolute difference between the activation of p_1 and p_2 is calculated and summed, after which the mean is taken. Function $\mathcal{A}$ is called for each possible permutation of the set $c_1 \cup c_2$. If L consists of only singleton sets, there are only two permutations: c_1, c_2 and c_2, c_1. Formula 7.1 takes the average of the value returned by $\mathcal{A}$ over all permutations. The classes c_1 and c_2 which get the smallest score, and thus are most similar, will be chosen to merge.

Instead of calculating the similarity between classes using all samples, only the samples of the classes being compared are used. This choice has been made to focus on the patterns which lead to the specific classes. If the patterns found to classify the compared classes are similar, the activation patterns should also be similar. If other samples would be used to calculate similarity, patterns found on noise could for example be included. Using only the class samples allows the formula to focus on the patterns pointing to the specific classes. As the activations of the compared classes are subtracted, formula 7.2 is also a measure of how often two classes are confused by the model. More confused classes are good candidates to merge as the final models might be more accurate in predicting class c_1 or c_2 rather than predicting these classes separately.

7.2 Used model

The LSTM model will hypertuned and trained on the training$_2$ set. The hyper tuning is done as described in section 6.4. However, there are two differences in the setup. The first difference is that the model will be given class weights, to account for class imbalance. This is done because otherwise frequent classes might be given relatively high activations as there is a higher chance of predicting these classes correctly. Therefore, the classes are weighted as shown in formula 7.3.

$$\mathcal{W}_c = \frac{\max F}{F_c} \tag{7.3}$$

```
HP = Hypertuning LSTM on training_2
LSTMs = []
for (i in range(0,10)){
    LSTMs.add(LSTM(HP).fit(training_2).predict(validation))
}

X = mean(LSTMs)

def MergeLabels(L,X){
    S = []
    for (i in range(0,n)){
        c_1, c_2 = M(L, X)
        L = L\{c_1, c_2}
        L = L ∪{{c_1, c_2}}
        S.Add(L)
    }
    return S
}
med = MergeLabels(med_labels,X)
op = MergeLabels(op_labels,X)
dep = MergeLabels(dep,X)
```

Listing 1: Merge algorithm. Here, n is the maximum number of iterations for the function

Here, F is the set of all frequencies of classes in the training set and F_c is that frequency for class c. The model is given a weight $\mathcal{W}_c$ for all classes c. After hyper tuning on the training$_2$ set, the model is trained on the training$_2$ set with the defined hyperparameters. It will predict the validation set. As an LSTM model has random weight initialization, the model is run 10 times and the predictions are averaged. This should provide more stable results. It seems the model is quite stable, as the mean variance is about 0.00319. Listing 1 shows the merge algorithm for a set of labels L. The MergeLabels function returns the list of a few possible class setups. For example, the 2nd element of the returned list features 2 class merges. The function is called for the three groups in which merges will be performed, medication (med_labels), operations (op_labels), and departments (dep_labels). For each merge step returned by MergeLabels, a non-tuned LSTM will be fitted on the training$_2$ set and tested on the validation set. Here a merge step includes all merges before it and the current merge.

Figure 7.2 shows the structure of the LSTM that will be used. This is purposely not hyper-tuned, as hyper-tuning it for every step is infeasible due to hardware constraints.

7.3 Results

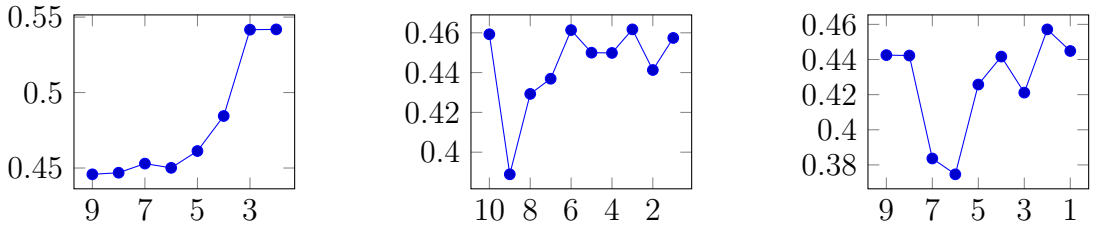
In this section, the results of the LSTM fitted on each merge step will be discussed. This will be done per group. Per group, the best merge step will be chosen based

Parameter	Value
Loss	Focal
Optimizer	Adam
Learning rate	0.001
Batch size	32
Epochs	10

Layer	Layer parameters
LSTM	Units: 256
Dense	Units: 128, activation: ReLU
Dropout	Dropout: 0.2
Output layer	Activation: sigmoid

Table 7.2: Non hyper tuned LSTM structure

on the macro F1 score of the model. This F1 score will be calculated only on the classes of each group.



(a) Medication classes (b) Operation classes (c) Department classes

Figure 7.1: For each group of classes, the F1 score by an LSTM plotted against how many classes were left in the merge step that was trained and tested on

Figure 7.1a shows the results for the medication class merge steps. In the first point, there are 9 classes left. Here, no merge has taken place yet. At 8, one merge has taken place, at 7, two merges have taken place, etcetera. The figure shows that the F1 score is the highest when leaving two classes. Therefore, this merge step is chosen and there are two classes left.

Figure 7.1b shows the results for the operations class merge steps. Here, the highest F1 score is found when there are three classes left. This merge step will be chosen, meaning that there will be three operation classes left.

Figure 7.1c shows the results for the department class merge steps. Here, the highest F1 score is found when there are two class left. This merge step will be chosen, meaning that there will be two department classes left.

7.4 Class distribution

Figure 7.2 shows the label distribution after merging classes. This shows a quite high label imbalance. While the cardiology and medication classes are very frequent and the none class is quite frequent, the other classes are quite sparse.

Cardio	None	PM	Med_1	Med_2	Op_1	Op_2	Dep_1	Dep_2
0.65	0.25	0.05	0.51	0.51	0.03	0.13	0.02	0.07

Table 7.3: Fraction of correct samples when predicting one for each class

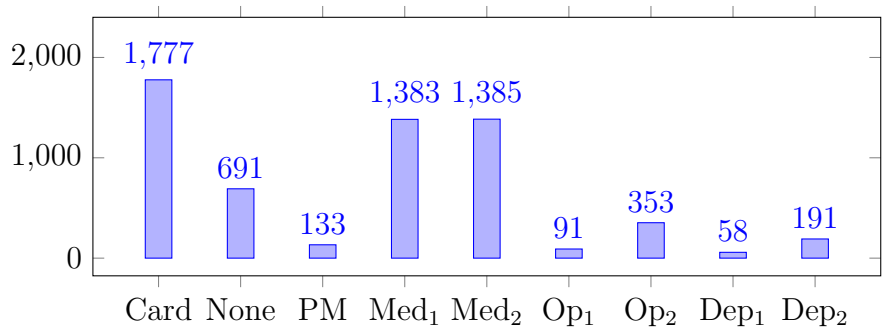


Figure 7.2: Label distribution after merging. Here Card stands for cardiology, PM stands for pacemaker, Med stands for medication, Op stands for operation and Dep stands for departments

Table 7.3 shows the fraction of correct samples achieved when predicting only 1 for each class. The table further accentuates the fact that especially the operations and departments classes are very infrequent.

Classes	1	2	3	4	5	6
Samples	940	670	753	251	99	4

Table 7.4: The number of samples corresponding to each possible amount of classes

Table 7.4 shows how many samples belong to a certain number of classes. Most samples only have one class, which makes sense as the none and department classes cannot have a second label. Hence, summing the frequencies of these classes is equal to the number of samples that belong to one class: $691 + 58 + 191 = 940$. The other samples have at least the cardiology class. Hence, the number of samples minus the samples with one class is equal to the frequency of the cardiology class: 2,717 - 940 = 1,777.

Across all samples, there are 6,062 classes to predict, meaning that on average there are $\frac{6{,}062}{2{,}717} = 2.23$ classes per sample. As there are nine classes over 2,717 samples, there are $9 \cdot 2{,}717 = 24{,}453$ total predictions to be made. From this follows that $\frac{6{,}062}{24{,}453} \cdot 100 = 24.79\%$ of the possible predictions are a class.

Chapter 8

Results

This chapter discusses the results of this research. The performance of each model will be discussed, after which model performance will be compared and tested for significant differences.

8.1 XGBoost

Hyperparameter	Value	Hyperparameter	Value
max_depth	11	reg_lambda	0.43
n_estimators	120	subsample	1.0
colsample_bytree	0.78		

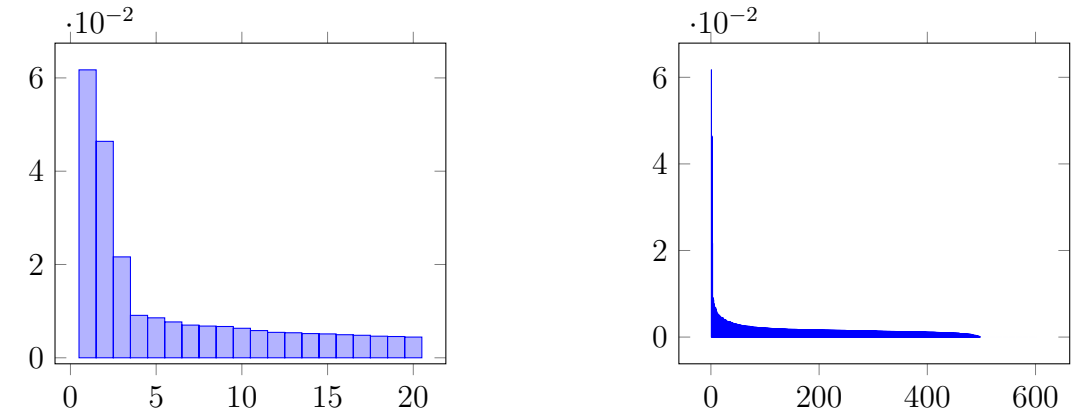
Table 8.1: Optimal values from hyperparameter tuning the XGB model

The XGBoost model is hyper-tuned as described in section 6.1. Table 8.1 shows the resulting values. From this follows that the model is somewhat complex, with 120 trees with a maximum depth of 12. Both of these are on the higher end of the ranges used for the hyperparameter tuning. The number of features used to train each tree is high, with colsample_bytree being 0.78. As subsample is 1, all samples are used for training. Lastly, reg_lambda is 0.43.

	Cardio	None	PM	Med_1	Med_2	Op_1	Op_2	Dep_1	Dep_2
Precision	0.71	0.52	0.43	0.6	0.6	0.5	0.36	0.0	0.33
Recall	0.87	0.14	0.12	0.71	0.67	0.04	0.05	0.0	0.02
F1 score	0.78	0.23	0.19	0.65	0.63	0.07	0.08	0.0	0.04
AUC	0.69	0.68	0.8	0.69	0.65	0.64	0.72	0.54	0.68

Table 8.2: Precision, recall, F1 score, and AUC for each class predicted by the XGBoost model

Table 8.2 shows the performance of the hyper-tuned XGBoost model. The model does not predict the operation and department classes often. Thus, these classes have higher precision than recall, resulting in a low F1 score. The pacemaker class does have a high AUC of 0.8. Similarly, the none class has a higher precision than



(a) Twenty most important features

(b) All features

Figure 8.1: Feature importance in XGBoost model

recall, resulting in a low F1 score, but still an AUC score of 0.68. The Departements_1 class is never predicted, resulting in an F1 score of 0 and a low AUC. The medication classes and cardiology classes have a higher recall but a lower precision, meaning that the class is overpredicted.

Figure 8.1 shows the feature importance given by the model, sorted from high to low. Figure 8.1a shows the twenty most important features. Here, there are three features that have especially high importance, after which the importance drops quickly. The important features are "Min_missing_HR", "Max_pacemaker", and "Mean_pacemaker". These are all summarization features of the previous time steps. Here, HR stands for heart rate.

Group of features	Summed importance	Mean importance
Previous labels	0.26	0.0022
Hematology values	0.25	0.0014
Missingness	0.21	0.0016
Lab	0.21	0.0013
Measurements	0.2	0.0021
GP letter	0.1	0.0014

Table 8.3: The summed and mean importance given by the XGB model per group of features, sorted by summed importance

Table 8.3 shows the summed and mean importance given by the XGB model for all groups of features. It shows that the previous labels group have the highest mean importance and summed importance of all groups of features. Two of the three most important features, "Max_pacemaker", and "Mean_pacemaker", are from that group. The GP letter group has the lowest summed importance, with a gap of 0.1 to the other groups.

Hyperparameter	Value
C	433.02
gamma	0.53
kernel	RBF

Table 8.4: Optimal values from hyperparameter tuning the SVM model

8.2 SVM

The SVM model is hyper-tuned as described in section 6.2. Table 8.4 shows the resulting values. The best performing kernel is the RBF kernel. A rather high C value of 433.02 is returned. This might result in a conservative model which does not predict classes often. Lastly, the optimal value for gamma found is 0.53, which is on the higher end of the hyperparameter range.

	Cardio	None	PM	Med_1	Med_2	Op_1	Op_2	Dep_1	Dep_2
Precision	0.64	0.0	0.0	0.47	0.5	0.0	0.0	0.0	0.0
Recall	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
F1 score	0.78	0.0	0.0	0.64	0.67	0.0	0.0	0.0	0.0
AUC	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5

Table 8.5: Precision, recall, F1 score, and AUC for each class predicted by the SVM model

Figure 8.5 shows the performance of the hyper-tuned SVM model. Except for the medication classes and the cardiology class, no class is predicted. The model seems to just predict these three classes for all samples, as the recall is 1, but the precision is the same as predicting only ones for these classes. This results in an AUC of 0.5 for all classes, which means that the model is similar to random guessing. Therefore, the model does not learn anything about the labels from the data at all.

8.3 Neural Network

Hyperparameter	Value	Hyperparameter	Value
Layers	1	Alpha_focal	0.9
Units	512	Gamma_focal	2.0
Dropout	0.7	loss fn	Focal
Learning rate	0.0006	Epochs	7
Batch size	256		

Table 8.6: Optimal values from hyperparameter tuning the Neural Network model

The Neural Network model is hyper-tuned as described in section 6.3. Table 8.6 shows the resulting values. Here, it seems that using only one set of dense and dropout layers outperformed the more complex models with more layers. This

means that the Neural Network was only able to find shallow patterns in the data. More complex patterns generally take more layers to find. The model uses 512 units, which is relatively high in the range given when tuning. Hence, it seems that the model is able to find a lot of these shallow patterns. The focal cross-entropy loss function performs better than the binary cross-entropy loss function. The focal loss function is given an alpha value of 0.9, which means that correct positive classes are weighed much higher than correct negative classes, which are given a weight of 1-0.9 = 0.1. The gamma value is equal to the recommended value by the authors [32]. Furthermore, a small learning rate of 0.0006, a large batch size of 256, and 7 epochs were selected as best performing. A small learning rate results in a small adjustment of weights every time, while the large batch size and low amount of epochs result in weights not being adjusted often. This further confirms that the patterns found by the model were shallow, as more complex patterns generally take more and larger weight adjustments to find.

	Cardio	None	PM	Med_1	Med_2	Op_1	Op_2	Dep_1	Dep_2
Precision	0.64	0.32	0.07	0.48	0.5	0.06	0.17	0.03	0.09
Recall	1.0	0.9	0.24	0.99	0.99	0.19	0.56	0.15	0.33
F1 score	0.78	0.47	0.11	0.65	0.66	0.1	0.27	0.06	0.14
AUC	0.48	0.63	0.58	0.51	0.49	0.6	0.59	0.68	0.62

Table 8.7: Precision, recall, F1 score, and AUC for each class predicted by the Neural Network model

Table 8.7 shows the performance of the hyper-tuned neural network model. For all classes, the recall score is much higher than their precision. The often occurring cardiology and medication classes have a recall of 1 or almost 1 but have a precision similar to the accuracy of only predicting ones, as shown in figure 7.3. So, it seems that the model does not learn anything about these classes, but rather predicts these classes for all samples. This results in low AUC scores, making the model similar to random guessing for these classes. The none class is similar, only with a bit higher precision and a bit lower recall. The other classes have better AUC scores, but low F1 scores.

8.4 LSTM

Hyperparameter	Value	Hyperparameter	Value
LSTM units	512	Epochs	19
Dense units	1024	Loss	Focal
Dropout	0.4	Alpha_focal	0.9
Learning rate	0.007	Gamma_focal	2.4
Batch size	32		

Table 8.8: Optimal values from hyperparameter tuning the LSTM model

The LSTM model is hyper-tuned as described in section 6.4. Table 8.8 shows the resulting values. Here, the number of LSTM and dense layer units is on the high end of the given value ranges at 512 and 1024 respectively. The dropout after the dense layer is set to 0.4. A learning rate of 0.007, a batch size of 32, and 19 epochs perform best. With the low batch size and a high number of epochs, the model seems to learn complex patterns. As a loss function, focal cross-entropy loss performs best. The loss functions alpha value is set to 0.9, and the gamma value is set to 2.4. This alpha is very high, weighting the 0 classes as only 0.1.

	Cardio	None	PM	Med_1	Med_2	Op_1	Op_2	Dep_1	Dep_2
Precision	0.68	0.34	0.18	0.55	0.57	0.18	0.23	0.0	0.2
Recall	0.86	0.53	0.32	0.86	0.84	0.15	0.37	0.0	0.44
F1 score	0.76	0.42	0.23	0.67	0.68	0.17	0.28	0.0	0.28
AUC	0.63	0.6	0.69	0.69	0.61	0.62	0.66	0.55	0.7

Table 8.9: Precision, recall, F1 score, and AUC for each class predicted by the LSTM model

Table 8.9 shows the performance of the LSTM model. The departement_1 class is never predicted. The operation_1 class has low precision and recall, resulting in a low F1 score. The other classes have a higher recall than precision, meaning that the model predicts that a class occurs often wrong. The cardiology and medication classes have a precision close to the accuracy of predicting only ones, but still a relatively high F1 score. The departement_2 and pacemaker classes have the highest AUC while scoring a low F1 score.

8.5 LSTM - Target replication

Hyperparameter	Value	Hyperparameter	Value
LSTM units	32	Epochs	11
Dense units	64	Loss	Focal
Dropout	0.30	Alpha_focal	0.70
Learning rate	0.003	Gamma_focal	3.2
Batch size	256	α (TR)	0.70

Table 8.10: Optimal values from hyperparameter tuning the LSTM (TR) model

The LSTM model with target replication (TR) is hyper-tuned as described in section 6.4. Table 8.10 shows the resulting values. Here, the number of LSTM and dense units is low, so it seems the model did not find a lot of patterns. The dropout is 0.3. The batch size is high at 256, the learning rate is only 0.003 and the model is trained for 11 epochs. From this, it seems that weights are not updated often. As a loss function, focal cross entropy with an alpha of 0.7 and a gamma of 3.2 perform best. Lastly, the target replication α value is 0.7. This means that the loss function weighs the final time step with a fraction of 0.3 and the other time steps with a fraction of $\frac{0.7}{19} = 0.04$.

	Cardio	None	Pacemaker	Med_1	Med_2	Op_1	Op_2	Dep_1	Dep_2
Precision	0.68	0.41	0.32	0.55	0.55	0.27	0.26	0.0	0.25
Recall	0.89	0.34	0.36	0.82	0.85	0.15	0.28	0.0	0.33
F1 score	0.77	0.37	0.34	0.65	0.67	0.2	0.27	0.0	0.28
AUC	0.67	0.64	0.7	0.67	0.63	0.7	0.63	0.5	0.77

Table 8.11: Precision, recall, F1 score, and AUC for each class predicted by the LSTM TR model

Table 8.11 shows the performance of the LSTM (TR) model. The $department_1$ class is never predicted. The often occurring cardiology class and medication classes have the highest F1 scores. Their precision is relatively low, which results in AUC scores of around 0.63-0.67. The other classes have low F1 scores but better AUC scores, between 0.63 and 0.77. Most of these classes have a precision similar to their recall.

8.6 Time-aware LSTM

Hyperparameter	Value	Hyperparameter	Value
LSTM units	128	Batch size	32
Dense units	256	Epochs	40
Dropout	0.30	Loss	Binary
Learning rate	0.0005		

Table 8.12: Optimal values from hyperparameter tuning the TLSTM model

The TLSTM model is hyper-tuned as described in section 6.5. Table 8.12 shows the resulting values. 128 LSTM layer units are chosen, while 256 dense layer units are chosen. The dropout is set to 0.3. The learning rate is 0.0005. However, the batch size is 32 and the number of epochs is 40. It seems the weights are updated often but very gradually. As loss function, binary cross entropy performs best.

	Cardio	None	PM	Med_1	Med_2	Op_1	Op_2	Dep_1	Dep_2
Precision	0.7	0.47	0.78	0.56	0.57	0.0	0.28	0.0	0.24
Recall	0.85	0.22	0.28	0.75	0.6	0.0	0.29	0.0	0.11
F1 score	0.77	0.3	0.41	0.64	0.58	0.0	0.28	0.0	0.15
AUC	0.6	0.56	0.64	0.61	0.58	0.5	0.59	0.5	0.54

Table 8.13: Precision, recall, F1 score, and AUC for each class predicted by the TLSTM model

The performance of the TLSTM model is shown in 8.13. Here, the $Operation_1$ and $Department_1$ classes are never predicted. The pacemaker and none classes have higher precision than recall, meaning they are underpredicted. The cardiology and

medication classes have higher recall than precision. All classes have a low AUC between 0.5 and 0.64.

8.7 Time-aware LSTM - Target replication

Hyperparameter	Value	Hyperparameter	Value
LSTM units	512	Batch size	64
Dense Units	256	Epochs	21
Dropout	0.60	Loss	Binary
Learning rate	0.004	α (TR)	0.93

Table 8.14: Optimal parameters from hyperparameter tuning the TLSTM (TR) model

The Time-aware LSTM model with Target Replication (TR) is hyper-tuned as described in section 6.5. Table 8.14 shows the resulting values. The number of LSTM units is 512 while the number of dense layer units is 256. There is a high dropout value of 0.6, meaning that the model needs quite heavy regularization. The learning rate is 0.004, the batch size is 64 and the number of epochs is 21. This means that the model is updating weights quite often. The binary cross entropy function is chosen as the loss function. Lastly, the α (TR) value is very high at 0.93. This means that the final time step is weighted with a fraction of 0.07 while the other time steps are weighted with a fraction of $\frac{0.93}{19} = 0.05$. This means that the final time step is weighted almost equal to the other time steps.

	Cardio	None	Pacemaker	Med_1	Med_2	Op_1	Op_2	Dep_1	Dep_2
Precision	0.67	0.6	0.55	0.56	0.54	0.29	0.24	0.0	0.25
Recall	0.91	0.08	0.24	0.71	0.76	0.08	0.14	0.0	0.16
F1 score	0.77	0.14	0.33	0.63	0.63	0.12	0.17	0.0	0.19
AUC	0.56	0.53	0.62	0.61	0.55	0.53	0.54	0.5	0.56

Table 8.15: Precision, recall, F1 score, and AUC for each class predicted by the TLSTM TR model

Table 8.15 shows the performance of the TLSTM (TR) model. The department_1 class is not predicted. The cardiology and medication classes have a higher recall than precision, meaning that they are overpredicted. The other classes are under-predicted, with higher precision than recall. Overall, the classes have AUC scores between 0.5 and 0.63, which is quite low.

8.8 Significance tests

To find if there are any significant differences between models, the McNemar test [37] is performed for every pair of models. As there are 7 models, $\binom{7}{2} = 21$ McNemar tests will be performed. The McNemar test works on a contingency table. The null

hypothesis states that the probability $p_{\text{correct,incorrect}}$ is the same as the probability $p_{\text{incorrect,correct}}$. Thus, the null and alternate hypotheses are defined as:

$$H_0 : p_{\text{correct,incorrect}} = p_{\text{incorrect,correct}}$$

$$H_1 : p_{\text{correct,incorrect}} \neq p_{\text{ incorrect,correct}}$$

As multiple tests are performed, the probability of getting a type 1 error increases [2]. A type 1 error means that a null hypothesis is rejected while it is true. To reduce the probability of a type 1 error, the Holm-Bonferroni [28] method, as recommended by Cangur et al. [16], is applied with a significance level of $\alpha = 0.05$.

	XGB	SVM	NN	LSTM	LSTM(TR)	TLSTM
SVM	2.0e-14	-	-	-	-	-
NN	2.5e-103	1.6e-93	-	-	-	-
LSTM	3.1e-23	2.8e-04	6.9e-50	-	-	-
LSTM(TR)	5.6e-10	2.5e-01	5.9e-80	8.2e-07	-	-
TLSTM	3.6e-05	8.7e-04	4.6e-82	2.1e-11	2.7e-02	-
TLSTM(TR)	1.1e-05	3.5e-04	3.4e-84	3.5e-10	3.7e-02	7.9e-01

Table 8.16: P-values returned by pair-wise McNemar tests between models. The blue-colored cells are considered significant by the Holm-Bonferroni method

Table 8.16 shows the p-values returned by applying the pairwise McNemar test to model pairs. The cells of the values that are considered significant by the Holm-Bonferroni method are colored blue. Most McNemar tests are considered significant.

8.9 Summary of model performance

	XGB	SVM	NN	LSTM	LSTM(TR)	T-LSTM	T-LSTM(TR)
Accuracy	0.82	0.78	0.67	0.76	0.79	0.8	0.8
F1 mic	0.61	0.62	0.57	0.6	0.62	0.59	0.6
F1 mac	0.3	0.23	0.36	0.39	0.39	0.35	0.33
F1 wt	0.55	0.52	0.6	0.61	0.6	0.56	0.55
AUC mic	0.86	0.83	0.79	0.81	0.84	0.73	0.74
AUC mac	0.68	0.5	0.58	0.64	0.66	0.57	0.55
AUC wt	0.68	0.5	0.52	0.64	0.66	0.59	0.56

Table 8.17: Summary of model performances in accuracy, F1 score and AUC. Here, mic stands for micro, mac stands for macro, and wt stands for weighted

Table 8.17 shows the micro, macro, and weighted F1 and AUC scores. Overall, the vanilla LSTM models have the highest weighted and macro F1 scores. The LSTM(TR) model has the highest F1 micro score together with the SVM model. However, as shown by the macro AUC of the SVM score, it performs as well as a random model. As discussed in section 8.2, the SVM model only predicts 1 for the

high-occurring classes and never predicts the other classes. The XGBoost model has the highest AUC scores, closely followed by the LSTM models. The LSTM models have better AUC and F1 scores than the T-LSTM models.

8.10 Subgroup discovery

Class	Complexity	Nodes	Class	Complexity	Nodes
Cardio	0.012	3	Op_1	0	23
None	0.008	7	Op_2	0.007	15
Pacemaker	0.128	1	Dep_1	0	33
Med_1	0.004	27	Dep_2	0	45
Med_2	0.009	3			

Table 8.18: The hypertuned complexity value and the number of nodes of each tree

For the subgroup discovery task, nine decision trees, one per class, will be tuned as described in section 6.6. Table 8.18 shows the hyper-tuned complexity value and how many nodes are in the tree for each class. The higher this complexity value gets, the fewer splits will be made in the tree. It seems that most trees are, even after optimizing the complexity, quite large. There are even three trees for which the complexity value is 0, meaning that no pruning is done at all.

Class	C0	C1	C1 fraction	Rules
Med_1	25	67	0.73	Mean_cardiology $\leq 0.58 \wedge$ Max_urem ≤ 0.35
Cardio	144	343	0.70	Cardiology ≤ 0.5

Table 8.19: The two most pure leaves for all classes

Table 8.19 shows the two most pure classes in all models. Here, a high class 1 fraction ($\frac{class1}{class0+class1}$) is achieved using only a few rules. The nine trees created consist of 157 nodes in total. Of these 157 nodes, only 11 achieved a class 1 fraction higher than 0.5. Of those nodes, 8 were in the $Medication_1$ tree, 2 were in the cardiology tree and 1 was in the $Medication_2$ tree. Except for one of the nodes in the cardiology tree, all nodes that achieved a class 1 fraction higher than 0.5 achieved a fraction higher than the occurrence fraction of the class that the tree was based on in the training and test set. These fractions are shown in table 7.3.

Chapter 9

Conclusion

In this chapter, the research questions will be answered. After this, they will be discussed. Lastly, recommendations for future research will be made.

9.1 Research questions

To answer research question one, first the other questions will be answered.

The second research question compares the Time-aware LSTM models against the LSTM models. The LSTM models have better or equal F1 and AUC scores when compared to the T-LSTM models. As per the significance tests, the LSTM model is significantly better than both T-LSTM models. The difference between the LSTM (TR) model and the T-LSTM models is insignificant. Overall it can be concluded that the T-LSTM models perform worse than the LSTM model and are equal to the LSTM(tr) model. Do note that the F1 and AUC scores of the LSTM(TR) model are much higher than those of the T-LSTM models.

The third research question compares the non-target replication (TR) models to the TR models. In the vanilla LSTM case, the target replication model performs a bit better overall, with a higher micro F1 and a higher AUC score, while the LSTM model does have a slightly higher weighted score. As there is a significant difference between the models, the LSTM (TR) model performs significantly better than the LSTM model. The T-LSTM and T-LSTM (TR) models perform very similarly. There is an insignificant difference between the models. Overall, target replication significantly improves performance in the vanilla LSTM case, while there is no significant difference in the T-LSTM case.

The fourth research question compares the time series models to the non-time series models. When looking at the non-time series models, the SVM performs very poorly. Concerning the other non-time series models, the neural network model has a higher weighted and macro F1, while the XGBoost model has a higher micro F1. The neural network does have a low AUC score due to almost always predicting the cardiology class and the medication classes. This results in a lower AUC. The XGBoost model performs significantly better than both the neural network and SVM models. There is a significant difference between all LSTM models and the XGBoost model. As the vanilla LSTM models have a higher macro and weighted F1 score, they are considered higher performing. Overall, when comparing the neural network and SVM models to the time-series models, the time-series models mostly outperform them. However, the difference in AUC and F1 scores between

the XGBoost model and the time series model is much smaller. The XGBoost model has a much higher AUC than the T-LSTM models. Overall, the time series models significantly outperform the non-time series models. However, a caveat here is that the LSTM (TR) model does not significantly outperform the SVM model. Due to the fact that the SVM model does not learn anything about the data shown by it only predicting 1 for three classes and only predicting 0 for other classes, this model cannot be considered valid.

The fifth research question asks if there are any classes that can be identified using subgroup discovery. Overall, subgroup discovery works for only a few classes. Of all nodes, only a few achieved a class 1 fraction higher than 0.5, and those nodes occurred in only three of the nine trees. Those three classes, the two medication classes, and the cardiology class were also the most occurring classes, which means achieving a high class 1 fraction is easier. Still, an improvement is made over the fraction of 1 class for 10 nodes.

The first research question asks how well the models were able to predict the classes. The LSTM (TR) model performs best, but even that model is not able to predict classes very accurately. Except for the cardiology and medication classes, the model F1 scores are quite low. The micro AUC score is quite good at 0.84, however, the weighted and macro AUC scores are only 0.66. The subgroup discovery method is able to get higher class 1 fractions for subgroups of the data for the two medication and cardiology classes. Overall, it seems difficult for all models to make accurate predictions.

9.2 Discussion

The low performance of the machine learning models can be the cause of a few different options. First of all, the labels might not be defined correctly. As the labels had to be constructed, arbitrary choices had to be made. It might be that a large group of patients is given the wrong classes. Another cause might be that the hyperparameters of the model were not optimized enough. The Bayesian optimizer does not consider all options. This would be very costly, and due to hardware and time constraints, this was not feasible. But with more options considered, the models might have performed better. It is also possible that the problem is simply too complex. It might be that no well-discernable patterns can be found in the blood measurements or any of the other values relating to the patient's medication, operation, or department.

Another factor that might reduce performance is the fact that multi-label classification was performed on unbalanced classes. With unbalanced classes, it makes sense for the less frequent classes to be more conservative and the more frequent classes to be less conservative. Having both cases in one model, however, means that the model has to balance this out. Performance might have increased if all classes were given their own models, making nine binary classification problems.

The time-aware LSTM did not provide a performance increase over the vanilla LSTM. The time-aware LSTM has a discount factor, which should allow the model to be able to retrieve the general profile of a patient, independent of short-term effects. This, however, did not seem to work well for these inputs. This might also be related to the fact that there is missing data on each patient before admittance to the hospital. Using this data, the T-LSTM model might be able to define the

general profile of a patient more accurately and using that provide better predictions. Another option is that the T-LSTM performs worse or equal to the LSTM because the T-LSTM was tuned less extensively due to hardware limitations.

The performance difference between the time-series models and non-time-series models is quite small, with the XGBoost model even having a higher AUC score than the time-series models. From this, it seems that the LSTM models did not find a lot of time-related patterns. This might be caused due to the fact that there is missing data from before being admitted at the UMCU. It could also be that the patterns found in blood pointing to disease are just as identifiable using summary statistics as they are when looking at the entire trend.

9.3 Future research

In this research, all labels had to be constructed from the ground up, using the available data. This introduces a lot of variation in the possible labels. There are many options to consider. Future research could try many different options and choose the one with the best performance.

To be able to define these labels, having all the GP letters of the selected patients would be useful. In the UMCU, switching between departments is done via the General Practitioner. This means if someone came into the cardiology department but was referred to another department, two GP letters will exist: the one referring to the cardiology department and the one referring to the other department. These letters could be text mined as described in section 4.3. Based on how similar the resulting embeddings are, one could decide whether both letters describe the same symptoms. If so, it seems the patient needs treatment from the other department. For this research, only letters from the cardiology department were available, which made it hard to differentiate which patient ended up being treated in other departments.

To be able to provide research similar to this at the UMCU, a care pathway identifier given to every entry in the database would be really useful. In the current database, one cannot be sure which entries belong to which care demand. Patients can have multiple care demands at one time, and currently, there is no way to differentiate between these demands in the database. Adding a care pathway identifier to all database entries could make it so that it is clear where a care demand starts and ends and which entry belongs to which demand.

In this research, only the data of patients who came into the cardiology department with a specific complaint (chest pains) was available. The aim was to be able to differentiate between patients who needed to be treated at the cardiology department and patients who needed to be treated at other departments. However, the data available on patients who need to be treated at these other departments was very limited. This results from the fact that patients are originally referred to the cardiology department. So, it becomes difficult to correctly predict the patients who need to be treated at these other departments. In future research, using the data of the patients who are referred to the other departments would possibly greatly improve the prediction performance of these other classes.

9.4 Relevance to the Computing Science Program

This thesis applies techniques and models taught during algorithmic data analysis courses to a real-life project. The research provides valuable insights into preprocessing unstructured data into a set of inputs and labels. Also, novel techniques are introduced, like the method to merge similar labels. The models used have been hyper-tuned extensively and compared. Also, the LSTM models have been altered to use target replication, which is not integrated into the LSTM or T-LSTM by default. This is very relevant to the degree: adding improvements to models which were taught in class. The research also provides insight into predicting with a time series input using time-series and non-time-series models.

Appendix A

Other results

A.1 LSTM without merged classes

	Precision	Recall	F1-score	support
B01	0.40	0.66	0.50	225
C01	0.24	0.36	0.29	66
C02	0.00	0.00	0.00	4
C03	0.35	0.52	0.42	124
C07	0.39	0.73	0.50	213
C08	0.28	0.43	0.34	137
C09	0.33	0.51	0.40	174
C10	0.37	0.62	0.46	221
Cardio_operation_other	0.00	0.00	0.00	2
Cardiology	0.67	0.92	0.77	429
Departement_other	0.00	0.00	0.00	2
Gynecology	0.00	0.00	0.00	2
Internal Medicine	0.00	0.00	0.00	16
Otorhinolaryngology	0.00	0.00	0.00	5
Pulmonary medicine	0.00	0.00	0.00	7
Gastrointestinal and liver diseases	0.00	0.00	0.00	4
Medication_other	0.00	0.00	0.00	0
Neurology	0.00	0.00	0.00	8
None	0.39	0.48	0.43	187
Rheumatology	1.00	0.10	0.18	10
Urology	0.00	0.00	0.00	4
Aneurysm	0.00	0.00	0.00	4
bypass	0.17	0.07	0.10	15
Cardiothoracic	0.17	0.12	0.14	69
Cardioversion	0.00	0.00	0.00	6
Coronaries	0.00	0.00	0.00	7

Dotter	0.15	0.05	0.07	44
Valves	0.00	0.00	0.00	7
Pacemaker	0.50	0.28	0.36	25
Stent	0.33	0.13	0.19	38
micro avg	0.42	0.58	0.48	2055
macro avg	0.19	0.20	0.17	2055
weighted avg	0.40	0.58	0.46	2055
samples avg	0.43	0.54	0.43	2055

Bibliography

[1] URL: https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html.

[2] Hervé Abdi. "Holm's sequential Bonferroni procedure". In: *Encyclopedia of research design* 1.8 (2010), pp. 1–8.

[3] Ameen Abu-Hanna et al. "PRIM versus CART in subgroup discovery: When patience is harmful". In: *Journal of Biomedical Informatics* 43.5 (2010), pp. 701–708.

[4] Maryam AlJame et al. "Ensemble learning model for diagnosing COVID-19 from routine blood tests". In: *Informatics in Medicine Unlocked* 21 (2020), p. 100449.

[5] Ethem Alpaydin. *Machine learning*. Mit Press, 2021.

[6] Fahad Kamal Alsheref and Wael Hassan Gomaa. "Blood diseases detection using classical machine learning algorithms". In: *International Journal of Advanced Computer Science and Applications* 10.7 (2019).

[7] Gustavo EAPA Batista, Maria Carolina Monard, et al. "A study of K-nearest neighbour as an imputation method." In: *His* 87.251-260 (2002), p. 48.

[8] Inci M Baytas et al. "Patient subtyping via time-aware LSTM networks". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 65–74.

[9] Andrew L Beam and Isaac S Kohane. "Big data and machine learning in health care". In: *Jama* 319.13 (2018), pp. 1317–1318.

[10] Andrew L Beam et al. "Clinical concept embeddings learned from massive sources of multimodal medical data". In: *Pacific Symposium on Biocomputing 2020*. World Scientific. 2019, pp. 295–306.

[11] Maarten J ten Berg et al. "Linking laboratory and medication data: new opportunities for pharmacoepidemiological research". In: (2007).

[12] James Bergstra, Dan Yamins, David D Cox, et al. "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms". In: *Proceedings of the 12th Python in science conference*. Vol. 13. Citeseer. 2013, p. 20.

[13] Tim Bezemer et al. "A human (e) factor in clinical decision support systems". In: *Journal of medical Internet research* 21.3 (2019), e11732.

[14] Sebastian Bock and Martin Weiß. "A proof of local convergence for the Adam optimizer". In: *2019 international joint conference on neural networks (IJCNN)*. IEEE. 2019, pp. 1–8.

[15] Rasmus Bro and Age K Smilde. "Principal component analysis". In: *Analytical methods* 6.9 (2014), pp. 2812–2831.

[16] Sengul Cangur, Handan Ankarali, and Ozge Pasin. "Comparing performances of multiple comparison methods in commonly used 2× C contingency tables". In: *Interdisciplinary Sciences: Computational Life Sciences* 8 (2016), pp. 337–345.

[17] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[18] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[19] Gaurav Chhabra. "Automated hematology analyzers: Recent trends and applications". In: *Journal of Laboratory Physicians* 10.01 (2018), pp. 015–016.

[20] Dennis B DeNicola. "Advances in hematology analyzers". In: *Topics in companion animal medicine* 26.2 (2011), pp. 52–61.

[21] XGBoost Developers. *XGBoost Parameters*. Accessed on 02/07/2023. URL: `https://xgboost.readthedocs.io/en/stable/parameter.html`.

[22] Bram van Es et al. "Negation detection in Dutch clinical texts: an evaluation of rule-based and machine learning methods". In: *BMC bioinformatics* 24.1 (2023), p. 10.

[23] Jerome H Friedman. "Stochastic gradient boosting". In: *Computational statistics & data analysis* 38.4 (2002), pp. 367–378.

[24] Beverly George-Gay and Katherine Parker. "Understanding the complete blood count with differential". In: *Journal of PeriAnesthesia Nursing* 18.2 (2003), pp. 96–117.

[25] Patrick A Gladding et al. "A machine learning PROGRAM to identify COVID-19 and other diseases from hematology data". In: *Future science OA* 7.7 (2021), FSO733.

[26] Gary B Green and Peter M Hill. "Chest pain: cardiac or not". In: *Tintinalli J. Tintinalli's Emergency Medicine: A Comprehensive Study Guide* (2012).

[27] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[28] Sture Holm. "A simple sequentially rejective multiple test procedure". In: *Scandinavian journal of statistics* (1979), pp. 65–70.

[29] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. *A practical guide to support vector classification*. 2003.

[30] Thomas J Hwang et al. "Temporal trends and factors associated with cardiovascular drug development, 1990 to 2012". In: *JACC: Basic to Translational Science* 1.5 (2016), pp. 301–308.

[31] Zeljko Kraljevic et al. "Multi-domain clinical natural language processing with MedCAT: the medical concept annotation toolkit". In: *Artificial intelligence in medicine* 117 (2021), p. 102083.

[32] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[33] Charles X Ling, Jin Huang, Harry Zhang, et al. "AUC: a statistically consistent and more discriminating measure than accuracy". In: *Ijcai*. Vol. 3. 2003, pp. 519–524.

[34] Zachary C Lipton et al. "Learning to diagnose with LSTM recurrent neural networks". In: *arXiv preprint arXiv:1511.03677* (2015).

[35] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, pp. 413–422.

[36] Bridget T McInnes, Ted Pedersen, and John Carlis. "Using UMLS Concept Unique Identifiers (CUIs) for word sense disambiguation in the biomedical domain". In: *AMIA annual symposium proceedings*. Vol. 2007. American Medical Informatics Association. 2007, p. 533.

[37] Quinn McNemar. "Note on the sampling error of the difference between correlated proportions or percentages". In: *Psychometrika* 12.2 (1947), pp. 153–157.

[38] Marianne A Messelink et al. "Identification and prediction of difficult-to-treat rheumatoid arthritis patients in structured and unstructured routine care data: results from a hackathon". In: *Arthritis Research & Therapy* 23.1 (2021), pp. 1–10.

[39] William S Noble. "What is a support vector machine?" In: *Nature biotechnology* 24.12 (2006), pp. 1565–1567.

[40] Tom O'Malley et al. *KerasTuner*. `https://github.com/keras-team/keras-tuner`. 2019.

[41] L Malin Overmars et al. "A Wolf in Sheep's Clothing: Reuse of Routinely Obtained Laboratory Data in Research". In: *Journal of Medical Internet Research* 24.11 (2022), e40516.

[42] L Malin Overmars et al. "Preventing unnecessary imaging in patients suspect of coronary artery disease through machine learning of electronic health records". In: *European Heart Journal-Digital Health* 3.1 (2022), pp. 11–19.

[43] SGOPAL Patro and Kishore Kumar Sahu. "Normalization: A preprocessing stage". In: *arXiv preprint arXiv:1503.06462* (2015).

[44] Sayan Putatunda and Kiran Rama. "A comparative analysis of hyperopt as against other approaches for hyper-parameter optimization of XGBoost". In: *Proceedings of the 2018 international conference on signal processing and machine learning*. 2018, pp. 6–10.

[45] Hojjat Salehinejad et al. "Recent advances in recurrent neural networks". In: *arXiv preprint arXiv:1801.01078* (2017).

[46] Stef Van Buuren and Karin Oudshoorn. *Flexible multivariate imputation by MICE*. Leiden: TNO, 1999.

[47] Jenna Wiens et al. "Do no harm: a roadmap for responsible machine learning for health care". In: *Nature medicine* 25.9 (2019), pp. 1337–1340.

[48] David H Wolpert. "Stacked generalization". In: *Neural networks* 5.2 (1992), pp. 241–259.

[49] Derek Wong and Stephen Yip. *Machine learning classifies cancer*. 2018.