uc3m | Universidad Carlos III de Madrid

Master Degree in Statistics for Data Science
Academic Year 2019-2020

*Master Thesis*

# Prediction of unplanned readmission cases using Natural Language Processing and Deep Learning Architectures

Javier Martínez Llamas

Jenny Alexandra Cifuentes Quintero

Madrid, 2020

## Abstract

Unplanned hospital readmission is one of the main concerns for health care institutions, since it represents a patient's exposure to risk and a preventable waste of medical resources. Automatically identifying patients with a higher risk of Intensive Care Unit (ICU) readmission would help to efficiently reallocate the medical resources that would be unnecessary used during a readmission.

The analysis and prediction of unplanned ICU readmission will be studied by means of Machine Learning (ML), Deep Learning (DL) and Natural Language Processing (NLP) techniques. Including state-of-the-art approaches like transfer learning with pre-trained models. In particular, caregivers' text notes about patients discharge at the ICU will be analysed. For that purpose, de-identified database MIMIC-III ('Medical Information Mart for Intensive Care'), of limited access provided by the MIT Lab for Computational Physiology, will be used. This large, single-center data-set includes high-resolution information associated to 61532 patients admitted to critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012.

Based on this data analysis, a performance comparison of the different DL-based strategies will be presented. Results show that simpler statistical models yield the best performance in comparison to DL-based models.

***Keywords***— Natural Language Processing, Machine Learning, Deep Learning, Neural Network, Transformers

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Motivation

Hospitals and healthcare systems collect and maintain massive amounts of patient data and records. However, the use of this information in a predictive manner is still emerging. In this specific topic, Artifical Intelligence (AI) could lead to significant medical advances, not only improving patient treatment but also reducing costs or enhancing and optimizing healthcare systems. In particular, a major concern of healthcare systems and hospitals is their optimization, mainly due to the limitations of medical resources.

In this line, unplanned hospital re-admissions have been presented as a preventable expense in medical equipment and personnel. In order to address this problematic, U.S. Patient Protection and Affordable Care Act, for instance, has penalized hospitals which report high re-admission rates. The research line of hospital re-admissions has been a recurring and widely studied topic in the medical literature, being an indicator of quality in healthcare system. As such, numerous studies have been carried out in order to highlight the percentage of re-admissions that can be prevented (around 27% of the total [1], [2]), as well as the costs that could be saved, which represent over 1 billion dollars per year [3] in the United States or thousands of euros per patient in Spain [4]. As in the case of any other medical information, the results exigency and criticality is high: early discharge can lead to later complications and late discharge leads to the inability to treat additional patients.

Automatic identification of patients with a higher risk of Intensive Care Unit (ICU) re-admission would help to efficiently reallocate the medical resources that would be unnecessary used during a readmission. However, traditional models rely on large, structured and relational databases, with growing complexity and cost. In addition, most of the information is then discarded, focusing the effort on the maintenance and pre-processing of data rather than on their analysis.

In this context, notes and texts written by healthcare personnel about patients contain a large amount of information, including insights that might otherwise not be included in relational databases. In particular, discharge notes contain the most up-to-date information on their medical condition at the time of discharge. It may include a summary of the medical record or prescribed medications among others. This unstructured and complete data could allow for scalable and affordable predictive techniques. However, Natural Language Processing NLP and Classification of clinical records remains to be a complex problem, where high performance metrics from text analysis are very difficult to achieve [5], [6].

## 1.2 Objectives

The aim of this project is to evaluate the viability of using Machine Learning and Natural Language Processing for the prediction of unplanned re-admissions in hospitals. As such, the processing and analysis of patients' discharge notes will be performed in order to analyse if they could be a sufficient and valid indicator for prediction. Accordingly, a comparison of state-of-the-art ML-based techniques will be developed, taking into account DL strategies, and more specifically transfer learning and pre-trained Transformers. Results will show a performance evaluation of the techniques involved in this work.

To this end, a number of objectives are defined:

- Data-base analysis and pre-processing for the application studied in this project.
- Research and analysis of the state of the art in general Machine Learning and specific Deep Learning techniques oriented to a binary classification medical application.
- Natural Language Processing of discharge text notes in order to clean and filter the relevant information required for the classification stage.
- Training, validation, comparison and evaluation of different ML-based techniques to predict unplanned re-admissions. Conclusions will be drawn based on the results.

## 1.3 Manuscript Organization

This thesis is organized in a total of 6 chapters, which includes the current introductory chapter. Chapter 2 establishes a theoretical framework for the project along with technical references and comparisons of previous work in the same field. Chapter 3 providing a detailed explanation of the database used (MIMIC-III) and all treatment and pre-processing strategies applied, including the NLP techniques and their application to patient discharge notes. Chapter 4 focuses exclusively on detailed descriptions and implementations of both ML and DL techniques and models. Chapter 5 compares the results and metrics of the previously developed models. Concluding in Chapter 6, discussing concerns and possible improvements. All code involved for the subsequent sections is publicly available [1].

[1]https://github.com/javiermzll/NLP-Prediction-Unplanned-Readmissions.git

# 2 Related Work

Traditional statistical models, proposed to address this research topic, rely on standardized quantifiable data gathered during a patient's stay at the ICU. These numerical scores, like The Stability and Workload Index for Transfer (SWIFT), Sequential Organ Failure Assessment (SOFA) or the Therapeutic Intervention Scoring System (TISS), include the analysis of a wide set of medical indicators at the time of discharge, allowing the estimation of unplanned re-admissions at ICU. Based on this premise, Veloso, Portela, Santos, *et al.* [7] have set out the physical differences of re-admitted patients using clustering algorithms (k-means, k-medoids, x-means) to process SWIFT values. In this line of thought, Rosa, Roehrig, Oliveira, *et al.* [8] have compared SWIFT, SOFA and TISS-28 scores in predicting death or unplanned hospital readmission within 48 hours of discharge by implementing a logistic regression. Results in this study showed that analysed scores maintain similar accuracy while a moderate performance measures are achieved to predict unplanned readmissions (AUROC 0.65-0.74).

Machine Learning approaches considerably improve the performance of their predecessors, including different data. Rojas, Carey, Edelson, *et al.* [9] analyses clinical data from 25,000 patients, including medications, interventions, vital signs or laboratory results during their entire ICU stay. In this case, authors implement a Gradient Boosting model, improving predictive performance (AUROC 0.76) in comparison to previous models. Similarly, Loreto, Lisboa, and Moreira [10] considers the entirety of the patients' evolution at ICU. With a total of 185 predictors and pre-discharge medical attributes several models are tested: Naïve Bayes, decision trees, rule-based models and ensemble methods. Random Forest and AdaBoost delivering slightly better performance. Resulting scores have been reported showing a significant performance improvement (AUROC 0.91).

To that extent, the experiments results highlight the clear relation between high performances and the algorithm complexity increase. In addition, the strategies reported in the literature process different data-sets, which difficult the corresponding strategies comparison and their scalability and point out the data dependency in some models.

Several studies have been proposed to address this issue. Rajkomar, Oren, Chen, *et al.* [5] employs unstructured medical records in a Fast Healthcare Interoperability Resources (FHIR) format, designed for the rapid exchange of information. These records include patients' numerical metrics in a non-relational form along with clinical notes. Various DL-based algorithms such as Long Short-Term Memory (LSTM) neural networks were implemented with different predictive tasks: in-hospital mortality, 30-day unplanned readmission (AUROC 0.75) and prolonged length of stay at ICU. Despite addressing the problem of scalability and increasing complexity in information, its performance is comparable with less complex and more interpretable models.

First approaches that incorporated the processing of a bedside medical notes, as a substitute for numerical predictors, have included the implementation of traditional text classifiers. In this research line, Curto, Carvalho, Salgado, *et al.* [11] have processed the MIMIC-II database (composed of detailed information of more than 32,000 patients including nursing notes) to predict the readmission of patients to the same hospital within 24 to 72 hours. Two strategies have been implemented: a fuzzy fingerprints based approach and a set of traditional classification algorithms (*e.g.* Multinomial Naïve Bayes, Random Forest, SVM). These models achieve similar scores (AUROC 0.65 – 0.80) to those found in the literature [6], [12] by using only text as the input source. This text-based approach considerably reduces the complexity of the required data structure. However, one of the main drawbacks of classical text classification algorithms is their low performance at modelling long-term dependencies in texts and words, focusing only in observable information as word counts.

Understanding these long-term dependencies yields a greater insight. LSTM networks or Convolutional Neural Networks (CNN) address this syntactic dependence. Craig, Arias, and Gillman [6] ha proposed NLP and DL-based techniques to predict 30-day unplanned re-admissions based solely on Doctors' discharge notes. Authors train a CNN on clinical notes from the Sarasota Memorial Hospital of mostly Caucasian patients, achieving similar scores in text classification (AUROC 0.7).

Similarly, Huang, Altosaar, and Ranganath [12] considers a new approach based on novel Neural Network architecture Transformers [13] and Google's pre-trained BERT model [14]. The model is trained and fine-tuned on clinical notes (MIMIC-III database) obtained during the patient's entire stay in the ICU, including every recorded report (radiology reports, nurses' and physician notes, pharmacy reports and discharge notes), considering a huge amount of information and achieving a higher performance (AUROC 0.768) when compared to models with numerical predictors.

Table 2.1 presents the results of the comparison of the previously described papers.

| Author | Features | Methods | Results |
|---|---|---|---|
| Rosa, Roehrig, Oliveira, *et al.* [8] | SWIFT, SOFA, TISS-28 scores at the time of discharge | Logistic Regression | AUROC 0.74 |
| Rojas, Carey, Edelson, *et al.* [9] | Demographic data, vital signs, laboratory tests, interventions during ICU admission, nursing scores, and diagnostic categories | Gradient Boosting | AUROC 0.76 |
| Loreto, Lisboa, and Moreira [10] | Demographic data , length of stay prior to ICU admission, comorbidities, severity indexes, interventions, organ support care during ICU stay and laboratory results. | AdaBoost<br>Random Forest<br>Naïve Bayes<br>Logit Boost | AUROC 0.91 |
| Rajkomar, Oren, Chen, *et al.* [5] | Unstructured medical records including clinical notes | LSTM Networks | AUROC 0.75 |
| Curto, Carvalho, Salgado, *et al.* [11] | MIMIC-II medical and nurse text along numerical predictors | Fuzzy Modelling<br>Naïve Bayes<br>Random Forest<br>SVM | AUROC 0.80 |

| | | | |
|---|---|---|---|
| Craig, Arias, and Gillman [6] | Discharge clinical Notes from the Sarasota Memorial Hospital | Convolutional Neural Networks | AUROC 0.70 |
| Huang, Altosaar, and Ranganath [12] | MIMIC-III clinical notes during the entire ICU stay, including radiology reports, nurses' and physician notes, pharmacy reports and discharge notes | Pre-trained BERT models | AUROC 0.76 |

Table 2.1: Comparative analysis of related work on ICU readmissions

Considering the results reported, we intend to perform an analysis of different techniques based on ML, DL and transfer learning (some of which were previously reported) by means of pre-trained transformers on the MIMIC-III database, pre-processed by means of Natural Language Processing strategies, in order to compare the results. As an initial analysis in this research area, this work will use the same input information to compare the different strategies, focusing only on the discharge notes. These data were selected due to its wide acceptance in unplanned readmissions' research. Finally, we will conclude on the performance metrics obtained.

# 3 Dataset and Pre-processing

## 3.1 Dataset Description

This project relies on the MIMIC-III database [15] ('Medical Information Mart for Intensive Care'). It includes clinical data of patients admitted to critical care units between 2001 and 2012 at the Beth Israel Deaconess Medical Center, comprising data associated with 53,423 distinct hospital admissions for adult patients (aged 16 years or above) and data for 7870 neonates. Given the sensitive nature of the medical information, the access to the database is limited and its information has been de-identified to avoid patients' privacy violations. Access has been requested to the Massachusetts Institute of Technology (MIT) Lab for Computational Physiology after completing a course in ethics and research, in order to maintain the integrity of the information and to ensure responsible processing, complying with security standards and pledging not to identify individual patients.

Data covers 38597 distinct adult patients and 49785 admissions, where the median ICU length stay is 2.1 days. It is structured in a relational database (traceable by ID attributes) consisting of 26 tables distributed in a collection of CSV files.

| Table Name | Description |
|---|---|
| ADMISSIONS | Every unique hospitalization for each patient in the database (defines HADM_ID) |
| CALLOUT | Information regarding when a patient was cleared for ICU discharge and when the patient was actually discharged. |
| CAREGIVERS | Every caregiver who has recorded data in the database (defines CGID). |
| CHARTEVENTS | All charted observations for patients. |
| CPTEVENTS | Procedures recorded as Current Procedural Terminology (CPT) codes. |
| D_CPT | High level dictionary of Current Procedural Terminology (CPT) codes. |
| D_ICD_DIAGNOSES | Dictionary of International Statistical Classification of Diseases and Related Health Problems (ICD-9) codes relating to diagnoses. |
| D_ICD_PROCEDURES | Dictionary of International Statistical Classification of Diseases and Related Health Problems (ICD-9) codes relating to procedures. |
| D_ITEMS | Dictionary of local codes ('ITEMIDs') appearing in the MIMIC database, except those that relate to laboratory tests. |
| D_LABITEMS | Dictionary of local codes ('ITEMIDs') appearing in the MIMIC database that relate to laboratory tests. |
| DATETIMEEVENTS | All recorded observations which are dates, for example time of dialysis or insertion of lines. |
| DIAGNOSES_ICD | Hospital assigned diagnoses, coded using the International Statistical Classification of Diseases and Related Health Problems (ICD) system. |

| | |
|---|---|
| DRGCODES | Diagnosis Related Groups (DRG), which are used by the hospital for billing purposes. |
| ICUSTAYS | Every unique ICU stay in the database (defines ICUSTAY_ID). |
| INPUTEVENTS_CV | Intake for patients monitored using the Philips CareVue system while in the ICU, e.g., intravenous medications, enteral feeding, etc. |
| INPUTEVENTS_MV | Intake for patients monitored using the iMDSoft MetaVision system while in the ICU, e.g., intravenous medications, enteral feeding, etc. |
| OUTPUTEVENTS | Output information for patients while in the ICU. |
| LABEVENTS | Laboratory measurements for patients both within the hospital and in outpatient clinics. |
| MICROBIOLOGYEVENTS | Microbiology culture results and antibiotic sensitivities from the hospital database. |
| NOTEEVENTS | Deidentified notes, including nursing and physician notes, ECG reports, radiology reports, and discharge summaries. |
| PATIENTS | Every unique patient in the database (defines SUBJECT_ID). |
| PRESCRIPTIONS | Medications ordered for a given patient. |
| PROCEDUREEVENTS_MV | Patient procedures for the subset of patients who were monitored in the ICU using the iMDSoft MetaVision system. |
| PROCEDURES_ICD | Patient procedures, coded using the International Statistical Classification of Diseases and Related Health Problems (ICD) system. |
| SERVICES | The clinical service under which a patient is registered. |
| TRANSFERS | Patient movement from bed to bed within the hospital, including ICU admission and discharge. |

Table 3.1: Overview of MIMIC-III tables
Note. Extracted from Johnson, Pollard, Shen, *et al.* [15]

However, since only the notes issued during the patients' stays in ICU will be used, the study will be limited to the NOTEEVENTS and ADMISSIONS tables. Specifically, these two tables contain the following information:

- ADMISSIONS – Subject and admission ID, admission time, discharge time, death time, admission type, admission location, discharge location, type of insurance, language, religion, marital status, ethnicity and diagnoses.
- NOTEEVENTS – Subject and admission ID, date of creation, category of the note, description and the actual text.

ADMISSIONS contains a total of 58976 observations. Since the aim is to predict unplanned readmissions, it is necessary that all observations have an associated admission date to allow this traceability. However, no observation lacks this data. On the contrary, discharge or death times can contain missing values, patients may not have been readmitted or deceased.

To determine whether a patient has been readmitted or to check the time between admissions, two new attributes have been created. This information is processed from the already existing attributes. Therefore, the data-set is ordered according to the patient and time of admission in such a way that the time and type of admission (NEXT_ADMITTIME, NEXT_ADMISSION_TYPE) are obtained from the next admission of that same patient.

Reviewing the types of readmission shows them to be classified as emergency, elective, newborn and urgent. As elective readmissions are not considered urgently necessary (and therefore not unplanned) they are filtered out during the previous process. In such a manner that only subsequent admissions, not of this type, are considered.

Once the date of the next readmission is determined, it is possible to calculate the days until the next re-entry, so that $Days\ to\ readmission = Next\ admission\ time - Discharge\ time$. Obtaining a total of 12456 total readmissions.

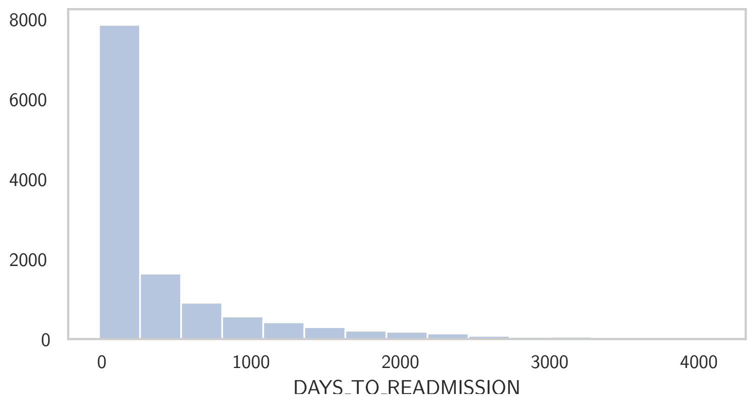

Figure 3.1: Days to Readmission Histogram

Despite the fact that the vast majority of patients have readmission times not exceeding one year (Figure 3.1), it is noteworthy the existence of patients who are re-admitted after 2 to 10 years (Third quartile of Table 3.2).

| Mean | First Quartile | Median | Third Quartile |
|---|---|---|---|
| 404.03 | 25 | 122 | 500 |

Table 3.2: Days to Readmission Mean and Quartiles (in days)

NOTEEVENTS contains 2083180 observations, a significantly higher number than those observed in the ADMISSIONS table. Where a single patient can generate multiple notes during a stay in the ICU.

Notes are divided into the following categories: Discharge Summary, Echo, ECG, Nursing, Physician, Rehab Services, Case Management, Respiratory, Nutrition, General, Social Work, Pharmacy, Consult, Radiology, Nursing/other. Since only information reported at the time of patients' discharge is to be used, Discharge Summary category will be used. Out of the 59652 remaining observations, 6926 patients have more than one discharge note so we only take the last one as a reference.

Once the two tables have been filtered, a left join is made between both tables on the patient ID and admission ID. 10.59% of the total number of patients admitted lack a discharge note. Upon verifying the reason for the missing values, we observe that 53.66% of NEWBORN admissions do not have discharge notes, in contrast to 4% of the other categories. Since most of the NEWBORNs do not have a discharge note, we remove this category.

To predict 30-day unplanned readmissions, READM_WITHIN_30 is defined as a binary response variable: 1 being the patients readmitted in less than 30 days and 0 the remainder. Patients deceased during their ICU stay are also filtered out. Resulting in 45321 observations for our final dataset.

| Non-Readmitted – 0 | Readmitted – 1 | Total |
|---|---|---|
| 42200 | 3121 | 45321 |

Table 3.3: Response Variable Classes

Table 3.3 presents a clear imbalance problem to be addressed (see Section 3.2.7) prior to the implementation of the models. Whereas Figure 3.2 represents and illustrates the pre-processing carried out until the final dataset is obtained.

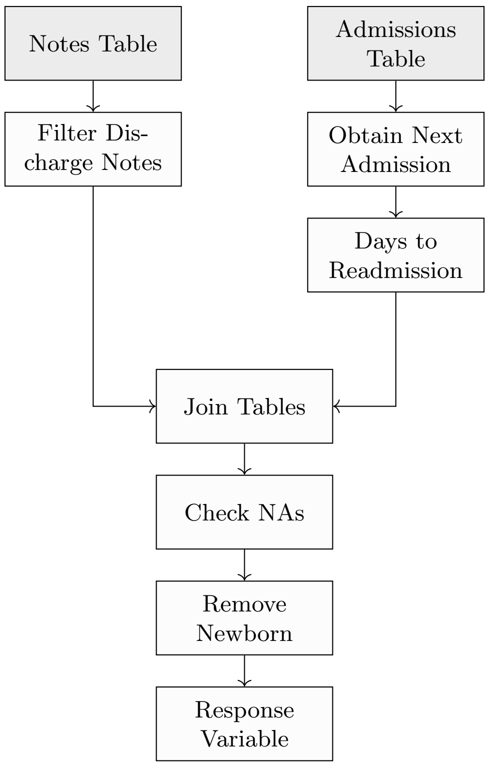

Figure 3.2: Dataset Pre-processing and Creation Flow

## 3.2 Natural Language Processing

Natural Language Processing (NLP) is understood as the ability of systems to understand, interpret and process human language. This language is unstructured, with complex meaning, dependencies and emotions. It is therefore that the proper analysis of texts and the optimal extraction of information is the main challenge when implementing text-based models.

Despite the fact that in this practical case medical discharge notes are analysed, the methodology followed in any text analysis and treatment is similar. During this section the content of discharge notes will be processed, taking as input raw text and converting it into interpretable data by mathematical models.

The content of the discharge notes varies widely. Nevertheless, the vast majority contains a description of the patient and a conclusion of his medical condition, which may include additional medications or observations. For reasons of ethics and discretion, none of these notes will be disclosed during this section. Limiting itself to the exemplification of the techniques NLP with alternative texts.

### 3.2.1 Missing Values

As stated in section 3.1, around 4% of the total readmissions lack a discharge note. To deal with missing values there are different approaches: removal or imputation.

Removing observations with missing values is usually associated with loss of information in models with numerical data due to the large number of predictors. When the text is missing, only the response variable is known, and the loss of information is minimal if the observation is removed.

Data imputation can be performed in numerous manners: inferred from known information (e.g., mean, median) and noise, or with zero or empty constants. Since no valid text can be generated for missing notes, text can only be imputed as empty strings.

Both options would have a similar effect in this case given the minimal percentage they represent. Since the lack of a discharge note does not necessarily imply its non-existence, where it might not be deemed necessary by medical staff to draft the note, all missing values are imputed as empty strings.

### 3.2.2 Removal of Characters and Numbers

As the notes are transcribed by medical staff, characters corresponding to carriage return (\r) or line feed (\n) are introduced during their writing to facilitate their visualization. These characters have no meaning within the text and need be removed.

De-identification forces all patients' names or dates to be removed or altered in such a way that the original string is replaced by "[** New Text**]". Where the dates do not correspond to the actual ones but maintain the same chronology. We additionally remove all possible numbers and special characters from the texts, in order to focus solely on text.

Firstly, the text is run through and each character sequence matching "\r", "\n" is removed. As to remove special characters and numbers a mapping table is drawn, where each character in *!”#$%&\’()*+,-./:;¡=¿?@[]ˆ_‘{|}˜0123456789* is mapped with an empty string "", replaced by it when encountered.

Taking a raw string (entire discharge note) as input it returns another processed string. So the following example text

> *There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be late!".*
>
> — Lewis Carroll, *Alice in Wonderland*

Will be converted to

> *There was nothing so very remarkable in that nor did Alice think it so very much out of the way to hear the Rabbit say to itself Oh dear Oh dear I shall be late*

### 3.2.3 Case-Folding

Words represented with capital letters, despite having the same meaning, are considered as different words. That is, the encoding is case sensitive. In order to overcome it, a common practice is to normalize the text.

There are different approaches to standardization, the most common of which is to convert all words to lower case. However, this may affect proper names or acronyms, where words with different meanings would be clustered together. To overcome this, it is a common practice to lower-case only those words at the beginning of sentences. Where the meaningful capitalization is maintained while the imposed capitalization is removed.

For discharge notes, it is decided to convert all the words to lower case as these are texts transcribed by medical personnel and do not necessarily have to be redacted perfectly.

Taking a string corresponding to the previously processed note as an entry, the normalized text is returned in lower case.

> *there was nothing so very remarkable in that nor did alice think it so very much out of the way to hear the rabbit say to itself oh dear oh dear i shall be late*

### 3.2.4 Tokenization

Once the text has been processed globally, each word must be treated individually. Prior to this, it is necessary to separate the note string into different sub-strings.

Tokenization methods require of a delimiter or separator, it varying according to each specific text. Usually a string of one character (e.g., ",", ";", " ") present between every pair of words; so that whenever encountered, the string is split.

As all special characters have been removed, a blank space is taken as delimiter. This eases the separation of words to a great extent, as there may be conflicts if not previously processed – e.g., contraction "can't" can be split in several forms *can,',t*; *can', t* or *can, 't*. Instead it is treated as a single word "cant".

Where each text string is converted into a vector of sub-strings

> ['there', 'was', 'nothing', 'so', 'very', 'remarkable', 'in', 'that', 'nor', 'did', 'alice', 'think', 'it', 'so', 'very', 'much', 'out', 'of', 'the', 'way', 'to', 'hear', 'the', 'rabbit', 'say', 'to', 'itself', 'oh', 'dear', 'oh', 'dear', 'i', 'shall', 'be', 'late']

and each individual word is referred as token.

In other types of text with complex and unique character sequences, it is common to use regular expressions for the definition of tokens.

### 3.2.5 Stop Words

Certain words in a language are more common than others; these are articles, pronouns, prepositions or similar. Due to their high frequency these words do not add significance to the text and consequently they are filtered out in NLP tasks. That is, their presence is meaningless for the retrieval of information.

Their removal reduces largely the complexity and dimensions of the vocabulary to be treated, as less words need to be indexed later (Section 3.2.8). Particularly important for large, varied-themed texts.

A total of 186 words are used for filtering the discharge notes, based on the list provided by Python's `nltk` library for the English language. Comprising lower-cased words like *i, you, is, were, have, do, but, if, or, because, while, to, a.* Additionally, common words to all notes are included, such as: *patient*, *date*, *admission*, *discharge*, *lastname*, *firstname* and *sex*. These words are arbitrarily selected based on the frequency of occurrence in the discharge notes and their functionality (i.e., all notes start with a header indicating admission and discharge time and sex of the patient).

Taking the vector of tokens previously split, each stop word is filtered out. Reducing the size of the vector.

['nothing', 'remarkable', 'alice', 'think', 'much', 'way', 'hear', 'rabbit', 'say', 'oh', 'dear', 'oh', 'dear', 'shall', 'late']

### 3.2.6 Morphological Normalization

Likewise, many words can be presented in various forms, suffering modifications (inflections) on the same root (e.g, read, reads, reading). Since we intend to classify text according to the number of occurrences of a word – and there is no need for the integral meaning of the text – it is necessary to cluster words with the same potential meaning and reduce complexity of the vocabulary.

There are different approaches to this, the main ones being Stemming and Lemmatization. Where the purpose of both is to group words according to their root. However, there are substantial differences between the two.

**Stemming** Words are reduced to its stem. That is, the word before any inflection is added. An algorithm removes suffixes or prefixes from the word (e.g, *-ed*, *-ing*, *-less*). Although the implementation of this heuristic is simple and does not require computational complexity, it can result in reductions that do not exist in the language. A clear example is *nothing* $\rightarrow$ *noth*, where the suffix *-ing* is removed.

**Lemmatization** Instead of eliminating any known inflection, the morphology of the word is taken into account. To do this, it is necessary to have a complete dictionary of the language in question with the lemma associated with each word. Unlike the stem, the lemma is a valid word within the language (canonical form) – Considering the word *writing*, its lemma would be *write* while *writ* the stem.

We use Python's `nltk` implementation of the Porter stemming algorithm [16] to eliminate suffixes. It considers a word to be formed by consonants (C) and vowels (V) on the form $[C](VC)^m[V]$, where in parsing and tokenization (widely used in compilers) "$[X]$" is optional. Based on the combination of vowels and consonants defined in 5 sets of rules, the suffixes are eliminated.

Since this is a rule-based algorithm the resulting words may be inaccurate from a formal perspective and not exist in the English language. However, these words do not need to be properly represented but rather their meaning must be accurately conveyed.

So that the previously cleaned text would result in

['noth', 'remark', 'alic', 'think', 'much', 'way', 'hear', 'rabbit', 'say', 'oh', 'dear', 'oh', 'dear', 'shall', 'late']

As previously remarked, *nothing* is converted to *noth*, preserving its meaning while it has been wrongly trimmed. Similarly, the proper name of Alice is shortened.

### 3.2.7 Downsampling

Once the processing of the database and the text has been carried out and prior to the final conversion of the data for the models, it is necessary to deal with the unbalanced dataset. Patients readmitted to the ICU account for 6.9% of the total (3121 over 45321 observations). If not treated, it could lead to an automatic prediction by the models of the majority class, while maintaining a high accuracy. Therefore, in addition to class balance, metrics such as AUROC, precision or recall will be preferred instead.

Therefore, a downsampling of the dataset is performed. We obtain $n = 3121$ (size of class 1) random samples on the majority class (0, not readmitted), so that the final dataset is completely balanced, 50% each class. Composed by the total of the minority class and a random sample, of the same size, of the majority class. Furthermore, a train and test split of the data corresponding to 80% and 20% of the total respectively is established.

### 3.2.8 Word Count

Finally, all the words are represented in a sparse matrix with the frequency of appearance of each one of them. A dictionary is established with all the words of the texts belonging to the data train set and the number of occurrences of the words is represented. That is, the resulting matrix would be of dimension $n \times m$, where $m$ is the number of words in all the train dataset and $n$ the number of notes.

As to limit vocabulary size and to reduce words with extremely low frequency, the maximum number of words ($m$) has been fixed to 3000. That is, only the 3000 most frequent words (after removing stop words) in all documents will be considered.

Since most of the words will not appear in the texts (and therefore the count will be zero) the matrix is sparse. Where all words are coded as integers.

$$\mathbf{S} = \begin{array}{c} \begin{array}{ccccccccccccc} \text{alic} & \text{dear} & \text{hear} & \text{late} & \text{much} & \text{noth} & \text{oh} & \text{rabbit} & \text{remark} & \text{say} & \text{shall} & \text{think} & \text{way} \\ \end{array} \\ \left[ \begin{array}{ccccccccccccc} 1 & 2 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 1 & 1 \end{array} \right] \end{array}$$

This dictionary is developed solely with the train set to prevent data leakage. Words may exist in the test set that have not been previously considered.

### 3.2.9 Pre-processing Flow

Each note is individually treated as follows

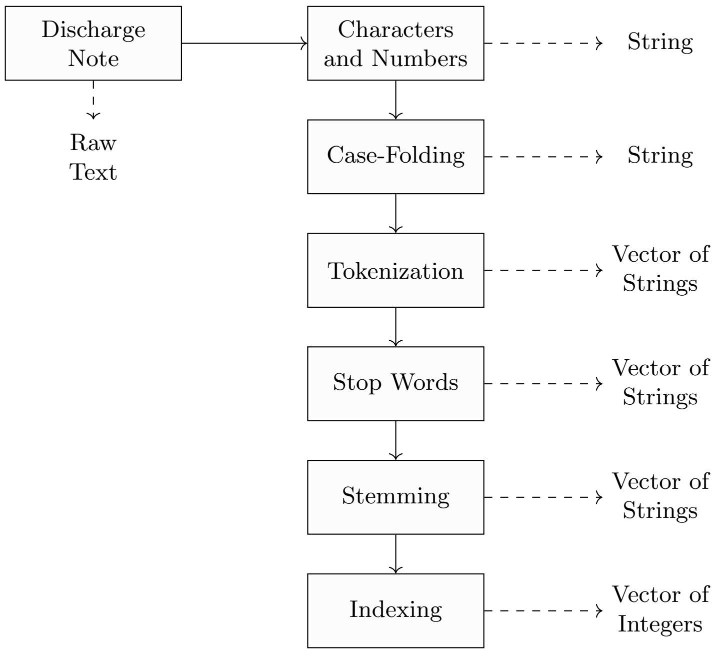

Figure 3.3: Discharge Note Pre-processing Flow

To eventually, aggregate all integer-converted texts into an sparse matrix to input the models.

# 4 Readmissions Prediction

The problem is modelled as a binary or binomial problem. In this work, the response variable label associated with a given observation is represented as $Y \in \{0, 1\}$ (two classes) measuring the patients' hospital readmissions within 30 days of discharge and a set of $n = 3000$ predictors (words).

This section is divided into three approaches associated to binary text classifiers. Section 4.1 covers statistical and Machine Learning algorithms, Section 4.2 specifically focuses on DL-based models like neural networks while Section 4.3 considers a state-of-the-art technique in NLP problems, using pre-trained Transformers and transfer learning approaches.

## 4.1 Machine Learning Algorithms

As mentioned in Section 3.2.7, the data used is completely balanced (50% class 1 and 50% class 0), while a train-test split is performed, where train set accounts for 0.8 of the total sample.

With the purpose of testing the performance of the models and whether a case of overfit exists, the following metrics are used on the train and test sets. Considering $TP$ as True Positives, $TN$ as True Negatives, $FP$ as False Positives and $FN$ as False Negatives.

**Accuracy** is the number of correctly predicted observations over the total of observations, which is, the proportion of correct predictions, defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

**Precision** or proportion of relevant observations (class 1) among the predicted ones, which is, the proportion of predicted relevant observations that are actually true:

$$Precision = \frac{TP}{TP + FP}.$$

**Recall** or proportion of relevant observations (class 1) among the true ones, which represents, how many observations were labelled correctly out of the total of relevant observations:

$$Recall = \frac{TP}{TP + FN}.$$

**Area Under the Receiver Operating Characteristics (AUROC)** states how the model distinguishes between classes, obtained by integrating the area from the ROC Curve by plotting Recall (x-axis) against the False Positive Rate (FPR) on y-axis, where:

$$FPR = \frac{FP}{TN + FP}.$$

### 4.1.1 Logistic Regression

Our response variable $Y_i$ for $i = 1, \ldots, m$ takes values one or zero with an associated probability, assuming it follows a Bernoulli distribution, then:

$$\Pr(Y_i = 0) = 1 - p_i \qquad \Pr(Y_i = 1) = p_i. \tag{4.1}$$

Logistic regression is based on the idea of a linear regression model to predict the probability $p_i$ of belonging to class $Y_i$, which is the probability of an observation to belong to a certain class.

Since a classical linear regression represented by $z = \beta_0 + \sum_{i=1}^{n} \beta_i x_i$, where $\beta$ coefficients weight the predictors, does not output a value in the interval $[0, 1]$ $(0 \leq z \leq 1)$ we need a link function (logit) that transforms the output in that interval.

$$z = \log(\frac{p}{1-p}) \implies p = \frac{e^z}{1 + e^z} \tag{4.2}$$

The observation is classified given a threshold $\tau = 0.5$, so that if $p \geq \tau$ the output class will be 1, 0 otherwise. The resulting output will then measure the probability of belonging to class 1. In order to obtain the coefficients $\boldsymbol{\beta}$ it is necessary to minimize the log-likelihood function by means of optimization algorithms since no analytical expression exists.

$$\hat{\boldsymbol{\beta}} = \min_{\beta} \; -log\mathcal{L}(\boldsymbol{\beta}; \boldsymbol{Y}, \boldsymbol{X}) \tag{4.3}$$

However, in order to avoid over-fit, an L2 regularization is added that penalizes the complexity of the model. Removing the presence of very specific words that prevent the generalization of the model by setting their coefficients to zero. As such, the function to minimize is

$$\hat{\boldsymbol{\beta}} = \min_{\beta} \; -log\mathcal{L}(\boldsymbol{\beta}; \boldsymbol{Y}, \boldsymbol{X}) + \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2, \tag{4.4}$$

where $\lambda$ is the regularization term. In this case, the Quasi-Newton method L-BFGS is used to optimize the function. Likewise, a grid search is also defined on the regularization parameter $\lambda$ with the following possible values: 1, 0.1, 0.01, 0.001, 0.0005, 0.0001, where $\lambda = 0.0005$ yields the best result. As shown in Table 4.1 Logistic Regression achieves on the test set an AUROC of 0.706. Figure 4.1 shows the ROC Curve for both train and test set.

| Metric | Train | Test |
|---|---|---|
| AUROC | 0.804 | 0.706 |
| Accuracy | 0.726 | 0.649 |
| Precision | 0.757 | 0.666 |
| Recall | 0.665 | 0.597 |

Table 4.1: Logistic Regression Metrics

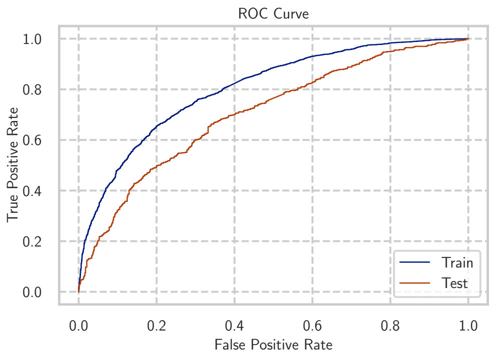

Figure 4.1: Logistic ROC Curve

Despite the simplicity of the technique, it allows for performance similar to more complex methods. Facilitating in addition the interpretability of the model.

### 4.1.2 Ridge Regression

A Linear Regression defined as $\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is the random noise, suffers from over-fitting and multicollinearity (redundancy in the predictors). With this strategy, the $\boldsymbol{\beta}$ coefficients are estimated by means of the Ordinary Least Squares (OLS) as:

$$\hat{\boldsymbol{\beta}} = \min_{\beta} \ \|\boldsymbol{Y} - \boldsymbol{X\beta}\|^2 \tag{4.5}$$

or with the explicit solution $\boldsymbol{\beta}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{Y}$.

In this work, to prevent the over-fitting an L2 regularization is considered as in the case of the Logistic Regression, in order to penalize the model complexity and shrink certain coefficients (yet not zero), minimizing the importance of certain predictors but never neglecting them. Based on this premise, the function to be minimized is:

$$\hat{\boldsymbol{\beta}} = \min_{\beta} \ \|\boldsymbol{Y} - \boldsymbol{X\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2, \tag{4.6}$$

where $\lambda$ is the regularization term. However, on the contrary to Logistic Regression, there is an analytical solution for the optimal $\boldsymbol{\beta}$ coefficients, defined as:

$$\boldsymbol{\beta}^* = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda I_n)^{-1}\boldsymbol{X}^\top \boldsymbol{Y}, \tag{4.7}$$

where $I_n$ is the identity matrix.

Despite being a regression model, Ridge Regression or Tikhonov regularization can be used as a classifier by converting the variable response as $\{-1, 1\}$ prior to fitting the model. Subsequently, Aagrid search on the regularization term $\lambda$ is defined with the following possible values: $10000, 10500, 11000, 11500, 12000, 12500, 13000, 13500, 14000, 14500$. Where $\lambda = 13000$ yields the best results.

Since the output class was previously converted, the model predicts values in the real interval $[-1, 1]$. In this case, a threshold was defined so that if the output value is greater than 0 it is predicted as class 1 and 0 otherwise.

| Metric | Train | Test |
|---|---|---|
| AUROC | 0.788 | 0.705 |
| Accuracy | 0.712 | 0.646 |
| Precision | 0.748 | 0.669 |
| Recall | 0.639 | 0.578 |

Table 4.2: Ridge Regression Metrics

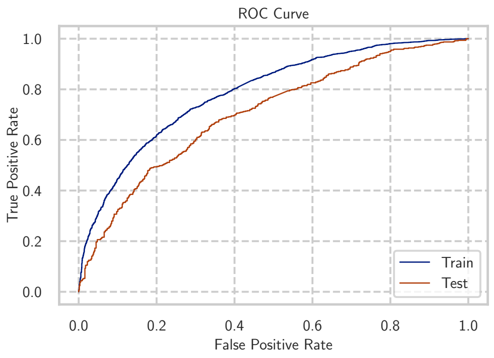

Figure 4.2: Ridge ROC Curve

As in the previous case, despite the simplicity of the model it achieves similar results and interpretability. With a slightly worse recall than Logistic regression.

### 4.1.3 Lasso Regression

As in the case of Ridge Regression, Lasso aims to reduce the over-fitting of a Linear Regression Model by shrinking the $\boldsymbol{\beta}$ coefficients to zero. This so-called L1 regularization acts both as a variable selection method and shrinkage at the same time, on the contrary to Ridge Regression (where coefficients are never zero). In this case, coefficients are penalized by a regularization term $\lambda$. Such that the function to minimize is

$$\hat{\boldsymbol{\beta}} = \min_{\beta} \ \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1. \tag{4.8}$$

Specifically, using this method, there is no analytical solution and the resulting model can attain sparsity, where a great number of coefficients are set to zero. During the implementation, a grid search on the regularization term $\lambda$ is defined with the following possible values: $0.1, 0.01, 0.001, 0.0001$. Where $\lambda = 0.01$ yields the best results.

| Metric | Train | Test |
|---|---|---|
| AUROC | 0.804 | 0.706 |
| Accuracy | 0.675 | 0.635 |
| Precision | 0.711 | 0.655 |
| Recall | 0.589 | 0.571 |

Table 4.3: Lasso Regression Metrics

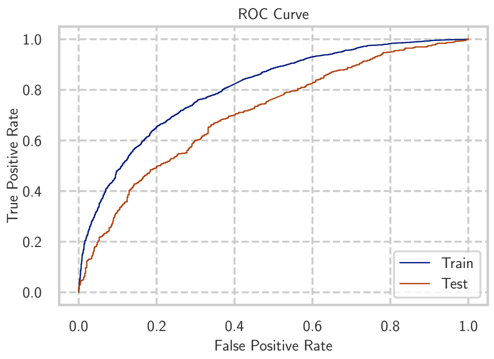

Figure 4.3: Lasso ROC Curve

To perform classification, given the output of the model $y = \beta_0 + \sum_{i=1}^{n} \beta_i x_i$, predicted class is set to 1 if $y \geq 0.5$ and zero otherwise. All predicted values are previously constrained in the range $[0, 1]$. With a slightly worse performance than Ridge and Logistic regression all models behave almost identical.

### 4.1.4 Elastic Net

In order to overcome the limitations of both Lasso and Ridge regression for linear models, Elastic Net combines both L1 and L2 regularizations, with a function to minimize equal to:

$$\hat{\boldsymbol{\beta}} = \min_{\beta} \; \|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \alpha\lambda\|\boldsymbol{\beta}\|_1 + \alpha(1-\lambda)\|\boldsymbol{\beta}\|_2^2, \tag{4.9}$$

where $\lambda$ regulates the type of normalization used and $\alpha$ its strength.

As in the previous experiments, a grid search on both parameters $\lambda$ and $\alpha$ is defined with the following possible values:

| Parameter | Values |
|---|---|
| $\alpha$ | $0.01, 0.1, 1, 10, 100$ |
| $\lambda$ | $0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$ |

Table 4.4: Grid Search on Elastic Net

Where $\lambda = 0.1$, and $\alpha = 0.1$ achieve the best results.

| Metric | Train | Test |
|---|---|---|
| AUROC | 0.739 | 0.695 |
| Accuracy | 0.674 | 0.642 |
| Precision | 0.711 | 0.666 |
| Recall | 0.587 | 0.570 |

Table 4.5: Lasso Regression Metrics

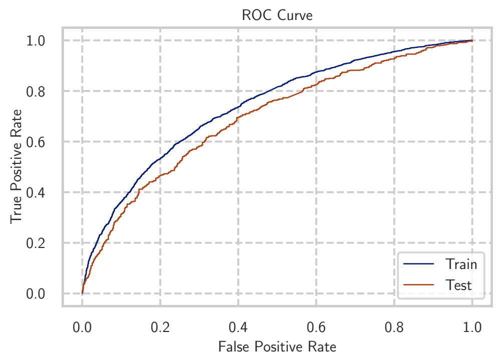

Figure 4.4: ElasticNet ROC Curve

Despite maintaining the same performance as Ridge and Lasso, the over-fitting is reduced, with a lower difference between train and test sets. Previous regression techniques recorded results similar to those reported in the literature, with the advantages of being intuitive approaches with intrinsic interpretability.

### 4.1.5 Support Vector Machine

All previous models were linear, meaning that the data needs to be linearly separable in order to achieve greater performance. Support Vector Machine (SVM) is a supervised learning method that bases its functioning on a linear classifier by defining an hyper-plane, extending this idea to non-linear data by the use of kernel functions.

SVM tries to maximize the distance or margin between the data points and the hyper-plane. As such, based on a linear model, the following hyper-plane is defined as

$$H_0 : \boldsymbol{w}^\top x + b = 0, \tag{4.10}$$

where $\boldsymbol{w}$ are the weights (equivalent to $\boldsymbol{\beta}$) and $b$ the residual or bias. Considering support vectors as the closest points to the hyper-plane, the aim is to maximize the gap between the hyper-plane and its support vectors. Assuming support vectors distant one unit from $H_0$, then both classes ($y \in \{-1, 1\}$) can be defined by:

$$\begin{aligned} \boldsymbol{w}^\top x_i + b &\geq -1 \quad \text{if } y_i = 1 \\ \boldsymbol{w}^\top x_i + b &\leq -1 \quad \text{if } y_i = -1, \end{aligned} \tag{4.11}$$

allowing the definition of two additional hyper-planes over the support vectors:

$$H_1 : \boldsymbol{w}^\top x + b = -1 \qquad H_2 : \boldsymbol{w}^\top x + b = 1. \tag{4.12}$$

Since the distance from any point to the hyper-plane is defined as $d = \frac{\boldsymbol{w}^\top x + b}{\|\boldsymbol{w}\|}$, the total margin ($H_1$ to $H_2$) will be $\rho = \frac{2}{\|\boldsymbol{w}\|}$, obtaining a constrained optimization problem, defined as:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\top x_i + b) \geq 1 \quad \forall i \end{aligned} \tag{4.13}$$

Figure 4.5 illustrates SVM on random two-dimensional data. Where the support vectors are the circled observations.

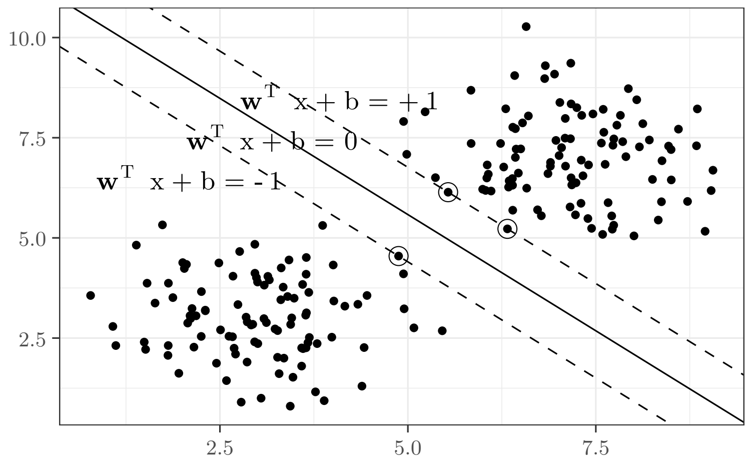

Figure 4.5: Linear SVM Exemplification

In order to control the over-fitting in this problem, the error is introduced in the optimization, reformulating the problem by:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} + C\sum_i \mathcal{E}_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^\top x_i + b) \geq 1 - \mathcal{E}_i \quad \forall i \\ & \mathcal{E} > 0 \end{aligned} \tag{4.14}$$

being $C$ the regularization term. In addition, in order to deal with non-linear data, SVM employs Kernel functions to transform the data into a different space where a linear classifier can be fit. Based on this idea, the following grid is defined

| Parameter | Values |
|---|---|
| $C$ | $0.1, 1, 2$ |
| Kernel | Linear, Poly, Gaussian |
| $\gamma$ | 0.001, 0.01, 0.1, 1 |

Table 4.6: Grid Search on SVM

| Kernel | Function |
|---|---|
| Linear | $K(x, x') = (x \cdot x')$ |
| Polynomial | $K(x, x') = (x \cdot x' + c)^d$ |
| Gaussian | $K(x, x') = \exp(\gamma \lVert x - x' \rVert^2)$ |

Table 4.7: SVM Kernels

where a Gaussian Kernel and $C = 1$, $\gamma = 0.001$ yield the best result. However, the train set achieves 0.99 accuracy while the test set 0.6, indicating a huge over-fitting. In order to reduce this difference, several approaches were considered.

In a first place, cross-validation is implemented for the metrics, using a 5-fold division to train and test the model, in order to check whether the result is dependant on the random sample taken. So that the 80% of data corresponding to the train partition is split into five; using one part for testing. A total of five models are trained on random partitioned data.

We define again a grid-search over the kernel function and $\gamma$ parameter for the Gaussian or Radial Basis Function (as defined in Table 4.7), but instead, the regularization parameter $C = 0.02$ was fixed. So that the results are

| Metric | Mean Train | Sd Train | Mean Test | Sd Test |
|---|---|---|---|---|
| AUROC | 0.814 | 0.004 | 0.646 | 0.018 |
| Accuracy | 0.678 | 0.004 | 0.601 | 0.019 |
| Precision | 0.654 | 0.004 | 0.585 | 0.015 |
| Recall | 0.654 | 0.004 | 0.695 | 0.023 |

Table 4.8: SVM Metrics

for $\gamma = 0.001$ and Radial Kernel, reducing considerably the previous over-fit of the model. Performing better than regression models, with a slightly worse AUROC and precision but significantly better recall.

### 4.1.6 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) is an Artificial Neural Network (ANN) with a a given number of hidden layers. Since we are dealing with a binary classification problem we define a MLP of one hidden layer of size 10, an output layer of size 2 and the input layer. The number of hidden layers and neurons was established after a grid-search, in which a greater number of layers and neurons meant over-fit, and fewer neurons impaired the model's performance.

A layer is formed by several neurons or perceptrons (size). Where each perceptron bases on a logistic regression $z = \boldsymbol{w}^\top x + b$ with an activation function (sigmoid) on the form $g(z) = \frac{1}{1+e^{-z}}$.

So that the output of one layer is taken as input for the next one. That is, one hidden layer learns on the form

$$f(x) = \boldsymbol{w}_2 g(\boldsymbol{w}_1^\top x + b_1) + b_2 \tag{4.15}$$

where $\boldsymbol{w}_2, b_2$ are the weights and bias corresponding to the hidden layer and $\boldsymbol{w}_1, b_1$ the weights and bias of the input layer. Being $g(.)$ the sigmoid (denoted $\sigma$) activation function. That output will be taken as input on the next layer with the corresponding activation function. Random initialized weights $\boldsymbol{w}$ are estimated by minimizing a loss function (Cross-entropy for binary classification) on the form

$$Loss(\hat{y}, y, \boldsymbol{w}) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y})) + \lambda||\boldsymbol{w}||_2^2 \tag{4.16}$$

where $\hat{y}$ is the resulting probability of belonging to a certain class, $y$ the true class and $\lambda||\boldsymbol{w}||_2^2$ the L2 regularization term. When optimizing on each iteration $i$, weights are updated according to the gradient of the loss function and a learning rate $\alpha > 0$

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \nabla Loss_i \tag{4.17}$$

We define a grid search on the learning rate and regularization parameters along with the solver used for optimization. Where Adam optimizer, stochastic gradient descent and L-BFGS (quasi-newton) are implemented. Yielding the best results $\lambda = 20, \alpha = 0.001$ with stochastic gradient descent. As in the case of SVM, this model is tested on the test data partition (20%), where the model is fit and cross-validated using a 5-fold partition on the train set, obtaining the results shown in Table 4.9.

| Metric | Mean Train | Sd Train | Mean Test | Sd Test |
|---|---|---|---|---|
| AUROC | 0.706 | 0.107 | 0.647 | 0.070 |
| Accuracy | 0.653 | 0.079 | 0.605 | 0.052 |
| Precision | 0.663 | 0.085 | 0.613 | 0.059 |
| Recall | 0.663 | 0.085 | 0.680 | 0.165 |

Table 4.9: MLP Metrics

Performing similar to SVM, with more variance in recall and better precision. When compared to regression techniques, slightly less AUROC is obtained but a much higher recall. With a superior overall performance.

### 4.1.7 Random Forest

This strategy is defined as an ensemble method formed by decision trees trained on random sub-samples with replacement (i.e., bootstrap), which means that for a decision tree $i$, a sub-sample of the $n$ total observations is drawn with replacement to act as training data. In this case, the decision nodes are established using a random subset of predictors, maintained for the entire specific tree. Subsequently, all decision tress are then averaged, defining the predicted class as the majority in the forest. This randomness in both sample and predictors reduces drastically the over-fitting and variance of the model.

We define a grid search on the number of trees, the maximum depth of each tree, the minimum number of samples to split a node (avoid granularity) and the minimum number of samples for the resulting leaf nodes, which means that a node cannot be split if there are not enough samples in it or in the resulting leaf nodes.

| Parameter | Values |
|---|---|
| Number of Trees | 100, 200, 300 |
| Min Sample Split | 1, 2, 5 |
| Min Sample Leaf | 1, 2, 5 |
| Maximum Depth | 2, 3, 4, 6 |

Table 4.10: Grid Search on Random Forest

The number of features considered per decision is $\sqrt{m}$, $m = 3000$, being the number of words. As it was previously shown, greater maximum depths of the tree were considered. However, since the training set achieved scores of 0.9 while the test set remained in 0.6, they were reduced to avoid over-fitting. In this case, using this implementation, 200 trees, 1 minimum sample per leaf, 2 samples per split achieved the best results. Finally, with the resulting metrics using a 5-fold cross-validation on the training data set, the results summarized in Table 4.11 were obtained.

| Metric | Mean Train | Sd Train | Mean Test | Sd Test |
|---|---|---|---|---|
| AUROC | 0.696 | 0.003 | 0.653 | 0.015 |
| Accuracy | 0.648 | 0.002 | 0.623 | 0.007 |
| Precision | 0.662 | 0.005 | 0.633 | 0.010 |
| Recall | 0.662 | 0.005 | 0.585 | 0.012 |

Table 4.11: Random Forest Metrics

### 4.1.8 Naïve Bayes

Given the Bayes theorem we can obtain the probability of belonging to class $y$ given a set of predictors by:

$$P(y|x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots x_n|y)}{P(x_1, \dots, x_n)}, \tag{4.18}$$

where $P(y)$ is our prior probability, $P(x_1, \dots, x_n)$ the predictor probability and $P(x_1, \dots x_n|y)$ the likelihood. Assuming all predictors are independent, the previous formula can be reformulated as:

$$P(y|x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, \dots, x_n)}, \tag{4.19}$$

where $P(x_1, \dots, x_n)$ remains constant for a given set of predictors:

$$P(y|x_1, \dots, x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y). \tag{4.20}$$

Therefore, the class with maximum probability (the predicted class) will be given by:

$$\hat{y} = \arg\max_{y} \; P(y)\prod_{i=1}^{n} P(x_i|y), \tag{4.21}$$

while the maximum likelihood supports additive (Laplace) smoothing. While a grid-search is implemented on the regularization parameter the model performs better without smoothing. So that the resulting metric using a 5-fold cross-validation are

| Metric | Mean Train | Sd Train | Mean Test | Sd Test |
|---|---|---|---|---|
| AUROC | 0.692 | 0.003 | 0.657 | 0.009 |
| Accuracy | 0.644 | 0.004 | 0.621 | 0.002 |
| Precision | 0.663 | 0.007 | 0.636 | 0.004 |
| Recall | 0.663 | 0.007 | 0.568 | 0.019 |

Table 4.12: Naïve Bayes Metrics

Obtaining a similar result to Random Forest but worse than the MLP.

### 4.1.9 XGBoost

Gradient boosting is, as RF, a decision tree ensemble method, where the training is sequential in order to correct misclassified observations. Using this method, each tree considers information from previously grown trees. Starting from an imperfect model comprising a total of $K$ trees, in the $m$ iteration the prediction will be improved by adding a new estimator:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x) = \hat{y}_m, \tag{4.22}$$

where $h_m(x) = y - F_{m-1}(x)$ and $\gamma$ is its weight. Based on this definition, we fit a decision tree on the residuals and add it to our model, correcting previous errors. Therefore, the additive model is written on the form as a weighted sum of functions (decision trees)

$$\hat{y} = \sum_{k=1}^{K} \gamma_k h_k(x) \tag{4.23}$$

Allowing for the optimization of an arbitrary loss – cross-entropy function 4.16 for binary classification with modified regularization term – through gradient descent, where $\gamma_m$ is optimized in each iteration.

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} Loss(y, \hat{y}) = \arg\min_{\gamma} \sum_{i=1}^{n} Loss(y, F_{m-1}(x) + \gamma_m h_m(x)) \tag{4.24}$$

During this implementation, the following grid search on the learning rate, maximum depth, minimum loss reduction $\gamma$ and minimum child weight is proposed:

| Parameter | Values |
|---|---|
| $\gamma$ | 100, 200, 300 |
| Min Child Weight | 1, 2, 5 |
| Maximum Depth | 1, 2, 5 |
| Learning Rate | 2, 3, 4, 6 |

Table 4.13: Grid Search on XGBoost

However, while the train set scores $\sim 0.8$, the test set scores $\sim 0.6$; showing a clear over-fitting to be addressed. Similar results were obtained when using five-fold cross-validation on the train set and the parameters being optimized on the test set. In order to reduce this over-fitting, an L2 regularization is introduced on the weights $\gamma$, without a meaningful impact. As such, while graphically exploring the training of the model (Figure 4.6), we noticed how the improvement halts prior to the final iterations (See Figure reftrainXGB).

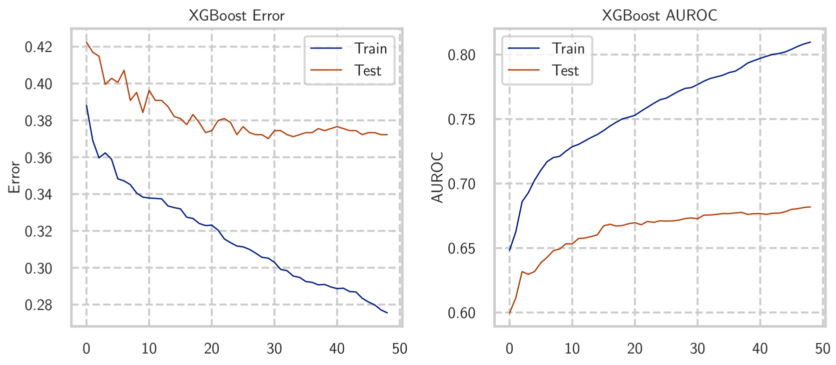

Figure 4.6: Error and AUROC per iteration XGBoost

With this idea in mind, we defined an early-stop methodology in the model, where the training will terminate if there is not improvement in the AUROC of an evaluation set. Three data sets are created: train (0.8), test (0.1) and validation (0.1). They were used for training, optimizing the hyper-parameters and as early-stop criteria, respectively. Subsequently, the training error (binary classification error) was set as evaluation metric, if during the training, this measure did not improve in 20 iterations the training was halted. The resulting model is defined as the one prior to those 20 iterations without improvement.

While the model was trained until 49 iterations (Figure 4.6), the final model is the state in the 29th iteration. The following metrics correspond to the L2 regularization model and to the early-stop model, the first one shows a higher difference for train/test metrics.

| Metric | Train | Test |
|---|---|---|
| AUROC | 0.759 | 0.617 |
| Accuracy | 0.759 | 0.620 |
| Precision | 0.800 | 0.611 |
| Recall | 0.691 | 0.563 |

Table 4.14: XGBoost L2 Metrics

| Metric | Train | Test |
|---|---|---|
| AUROC | 0.694 | 0.636 |
| Accuracy | 0.694 | 0.639 |
| Precision | 0.727 | 0.638 |
| Recall | 0.623 | 0.568 |

Table 4.15: XGBoost Early-Stop Metrics

With an almost identical performance as Naïve Bayesa and RF. However, the overall performance is worse than SVM, MLP and regression models.

## 4.2 Deep Learning Algorithms

This section discusses three different deep neural network architectures, focusing on long-term information retention and the relationship between words. In the machine learning models the input to the models was the frequency of appearance of the words, eliminating the order and relationship between words. Therefore, a prior reworking of the data is necessary as to consider these long-term dependencies.

Instead of counting each word's frequency a dictionary of words is created, where all words in the training set are considered, in this case, each unique word is assigned to an integer index, so every text is converted to a vector of integers of same length as the original token vector. Based on the text showed in Section 3.2.6, the resulting vector will be:

| noth | remark | alic | think | much | way | hear | rabbit | say | oh | dear | oh | dear | shall | late |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [ 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 1 | 2 | 1 | 2 | 12 | 13 ] |

where non-appearing and non-indexed words in the text will be referred as 0.

### 4.2.1 Word Embeddings

The representation of words in sparse matrix, where texts will lack most words, presents difficulties during neural network training. Therefore, it is necessary firstly to convert these representations in a dense form. Word embeddings represent each word as a real-valued vector of similarities, stating the relationship with other words. In this case, the word space is represented as a dense matrix. These similarities are initially unknown and are adjusted during training, where the first layer of the Deep Learning model is an embedding layer, learning jointly with the other layers of the model.

Word embedding acts as an index matrix of the words, Where similar words have similar encoding. The dimension of the matrix is defined, taking into account that larger dimensions attain more detailed relationships while the computational cost and data required increases.

### 4.2.2 LSTM Networks

In feed-forward networks (e.g., MLP) the information passes through the nodes in a unidirectional and straightforward way, where in each iteration, the weights are updated according to the errors by means of a back-propagation strategy. As such, it computes the gradient of the loss function (cross-entropy for binary classification problems) with respect to the weights as stated in Section 4.1.6. This leads to a rapid decline in learning, which means that the impact of new information decreases over time: as the error is minimized and the gradient decreases, weights are less affected. In order to overcome this effect, a Recurrent Neural Network (RNN) uses the internal state of the network at the previous output as input to the model, following a chained module structure, so that the information is recurrently analysed.

Figure 4.7 shows this dependence structure, where $A$ is a repeating module and $x_t, h_t$ the input and output at time $t$ respectively. In traditional RNN this module will consist only of a single Artificial Neural Network.

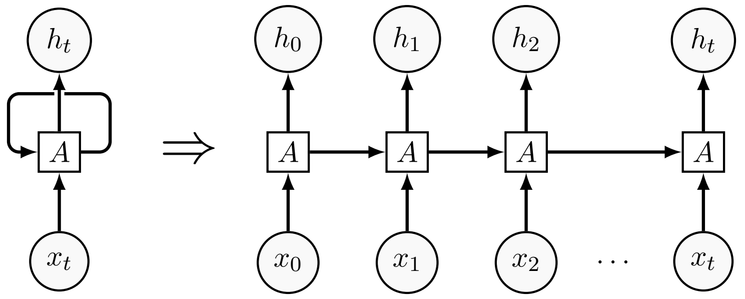

Figure 4.7: RNN Structure

Long Short-Term Memory (LSTM) Networks [17] are a type of RNN where each module is formed by cells, input gates, output gates and forget gates. Using this methodology, information passing through the cells is altered by the corresponding gates.

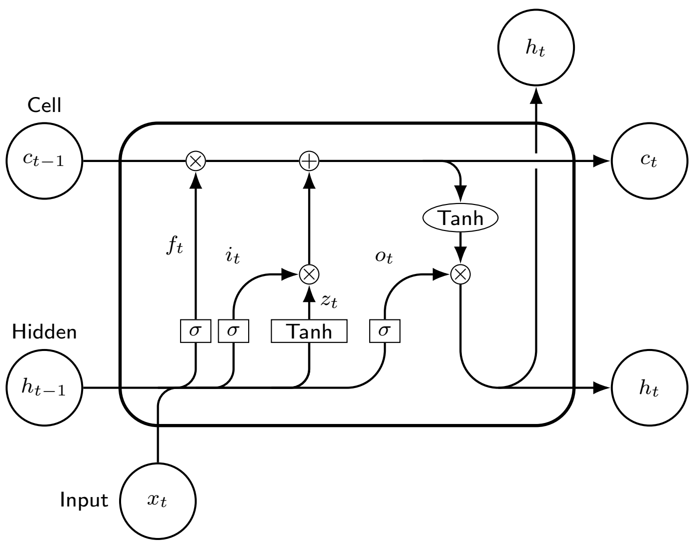

Figure 4.8: LSTM Module Representation

Information goes raw through the cell pipe $c_{t-1}$ to $c_t$. The forget gate $f_t$ is formed by an ANN with a Sigmoid (logistic function as mentioned in Section 4.1.6) activation function and a point-wise multiplication (Figure 4.8), such that the output of the forget gate will be given by:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f) \tag{4.25}$$

where $[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]$ is the concatenation of the previous output of the model and the current input, $\boldsymbol{W}_f$ a matrix containing the weights of the network and $\boldsymbol{b}_f$ a vector of biases. The output is a vector of real numbers in the interval $[0, 1]$ acting as a proportion of previous information. The input gate states the new information to be added to the model, formed by two ANN, with a Sigmoid activation function and a Tanh activation function. Specifically, the Tanh function is a re-scaled Sigmoid function on the form $tanh(x) = 2\sigma(2x) - 1$, with an output range $[-1, 1]$. Their outputs are given by:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i) \qquad \boldsymbol{z}_t = tanh(\boldsymbol{W}_z[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_z) \tag{4.26}$$

Therefore, the resulting state of the cell will be given by $\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \boldsymbol{z}_t$. The output of gate $h_t$ being given by $\boldsymbol{h}_t = tanh(\boldsymbol{c}_t) \circ \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o)$.

### Optimization

Two optimization algorithms are tested: Adam [18] or Adaptative Moment Estimation and RMSprop or Root Mean Square Propagation are used, both being gradient descent variants, without significant difference in the performance metrics. Since Adam optimizer is expected to behave better, all remaining networks will use this algorithm as default. In order to avoid the calculation of the whole gradient, we select random subsamples of fixed size and use its gradient for weight update. This selection avoids computation expense, with faster training and convergence times while maintaining good performance. The selected batch size was 64.

### Network Structure

Several combinations of layers are implemented: two stacked LSTM networks, single LSTM, bi-directional LSTM – with two hidden layers of opposite directions where the information not only flows forward but backward – and an LSTM with a dense network connected to it (i.e., MLP). All of these are ultimately connected to a dense layer of two neurons for binary category output. The differences in performance of all these structures are negligible.

| Network | Neurons | Approx. Time (min) | Accuracy | AUROC |
|---|---|---|---|---|
| Single LSTM | 32 | 260 | 0.582 | 0.582 |
| Single LSTM | 64 | 391 | 0.587 | 0.587 |
| Single LSTM | 256 | 1510 | 0.584 | 0.584 |
| LSTM & Dense | 32 - 32 | 258 | 0.585 | 0.584 |
| Bidirectional LSTM | 126 | 705 | 0.573 | 0.574 |
| Two Stacked LSTM | 64 - 64 | 406 | 0.589 | 0.590 |

Table 4.16: LSTM Structure Variations

However, with rising complexity the cost and time of training increases, so a single LSTM network with 32 units will be taken as a baseline due to its trade-off between performance and computation time.

A dropout of 0.2 is added to the LSTM for both the input units and the recurrent units. In order to prevent over-fitting a proportion of the units (dimensionality of output space) is removed and excluded, which means that on each iteration, 20% of the units will not affect the weight updates and the remaining training operations.

The resulting network follows the structure depicted in Figure 4.9, starting with the input layer, a sparse matrix (as defined in Section 4.2) of 5000 features (most frequent words), followed by the embedding layer. The number of most frequent words to be considered has been expanded in comparison with ML-based models (3000), allowing to reflect deeper long-term dependencies. Where the best performance is obtained with 5000 words, worsening slightly as the least significant words are introduced.

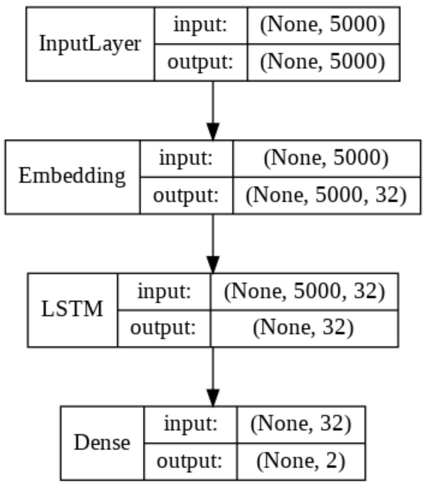

Figure 4.9: LSTM Network Structure

### Training and Metrics

With a fixed learning rate of 0.00001 the network is trained on 100 epochs. Where the model is trained on the complete dataset once per epoch in mini-batches of size 64. The original dataset is split with a proportion 0.8 for the training set and 0.2 for the test set. However, another 20% of the training set is used for evaluation purposes (Test on Figure 4.10).

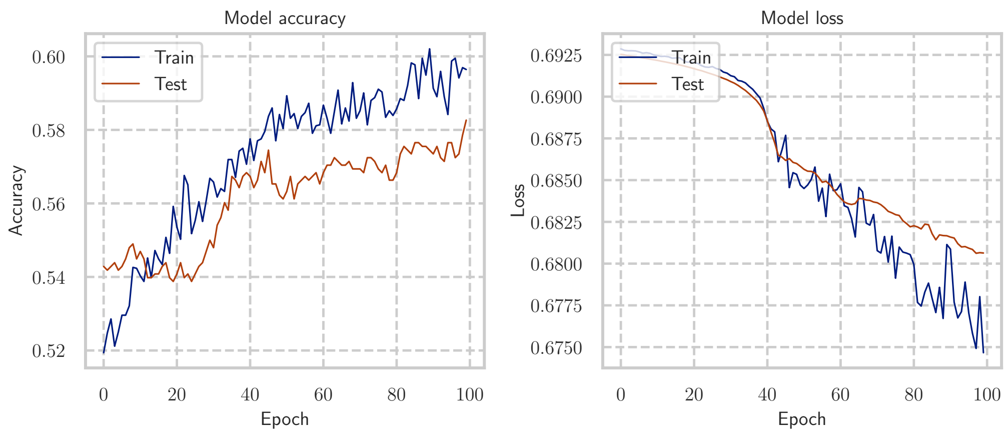

Figure 4.10: LSTM Network Training

From iteration 20 the train and test set metrics start to diverge, where the evaluation score improvement starts halting.

Obtaining the following metrics on the original test set.

| AUROC | Accuracy | Precision | Recall | $F_1$ |
|---|---|---|---|---|
| 0.579 | 0.579 | 0.579 | 0.579 | 0.578 |

Table 4.17: LSTM Metrics

Where performance does not surpass simpler models such as those contemplated in Section 4.1 despite the higher complexity and cost.

## 4.2.3 GRU Networks

Following the idea of gated LSTM Networks, Gated Recurrent Unit (GRU) are another variant of RNN with just two gates (reset and update) to prevent gradient vanishing. A simpler structure than LSTM with an even performance.

Where the update gate $z_t$ as depicted in Figure 4.11 acts as input gate of the LSTM, stating how much of past information is let through. The reset gate $r_t$ states how much of past information is forgotten. Formed by a fully connected layer with a Sigmoid activation function. Such that

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_z) \qquad \boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_r) \tag{4.27}$$

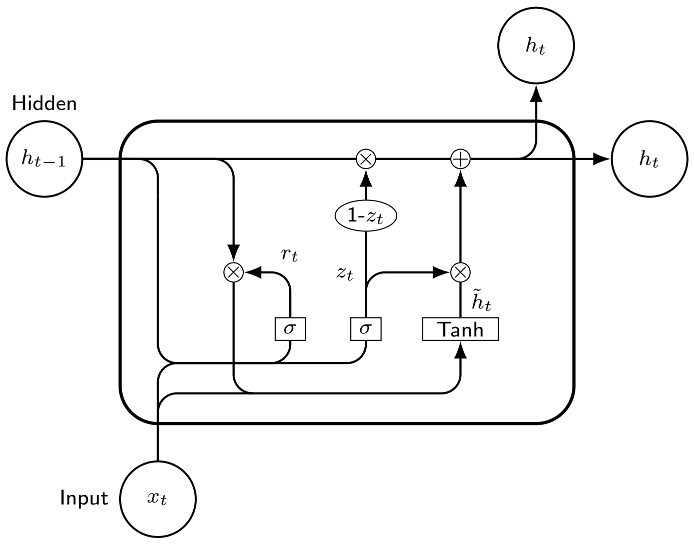

Figure 4.11: GRU Module Representation

The output of the reset gate is combined with the hidden state, serving as input for another fully-connected layer with a Tanh activation function, $\tilde{h}_t$. Resulting in the following hidden state candidates

$$\tilde{\boldsymbol{h}}_t = tanh(\boldsymbol{W}_h[\boldsymbol{r}_t \circ \boldsymbol{h}_{t-1}] + \boldsymbol{b}_h) \tag{4.28}$$

Then, the resulting output for the GRU module would be

$$\boldsymbol{h_t} = \boldsymbol{z}_t \circ \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t) \circ \tilde{\boldsymbol{h}}_t \tag{4.29}$$

### Network Structure

As in the case of LSTM networks, several combinations are tested. The best-performing network being the one depicted in Figure 4.12 with just a single GRU layer.

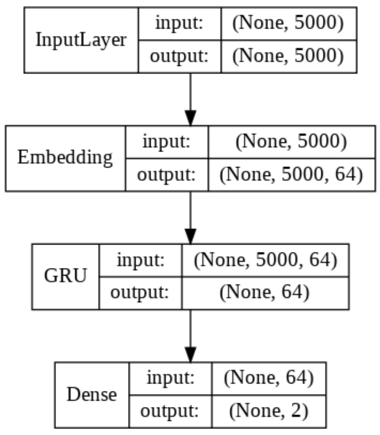

Figure 4.12: GRU Network Structure

As in the previous case, a 0.2 dropout is added to the GRU layer.

#### Training and Metrics

With a fixed learning rate of 0.0001 the network is trained on 20 epochs in mini-batches of size 64. The dataset division structure being the same as in LSTM. With faster convergence than LSTM. The resulting metrics being

| AUROC | Accuracy | Precision | Recall | $F_1$ |
|---|---|---|---|---|
| 0.612 | 0.613 | 0.615 | 0.613 | 0.610 |

Table 4.18: GRU Metrics

### 4.2.4 Convolutional Neural Networks and RNN

Convolutional Neural Networks (CNN) are a feed-forward ANN for feature extraction. Originally developed for 2-dimensional data and image recognition, these networks perform a series of operations on the data matrix to reduce its size. One-dimensional CNN are widely used in text recognition problems to identify patterns on time-sequenced data.

By using a filter or kernel of size $n$ the matrix (2D) or vector (1D) is subsampled into a series of $n \times n$ matrices or vectors of $n$ size. Then this subsample is summarized by certain mathematical operations (e.g., average, maximum or minimum). Data is normalized using a Rectified Linear Unit (ReLU) activation function on the form $f(x) = \max(0, x)$ where all negatives values are removed.

#### Networks Structure

By stacking a 1D CNN between the embedding and the LSTM and GRU layers we allow for feature extraction, reduction of problem size and faster computing time.

The resulting structures for both CNN-LSTM and CNN-GRU being

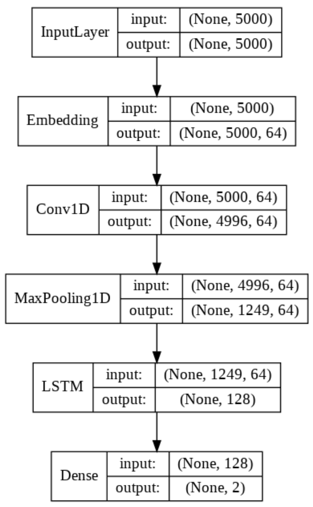

Figure 4.13: LSTM-CNN Structure

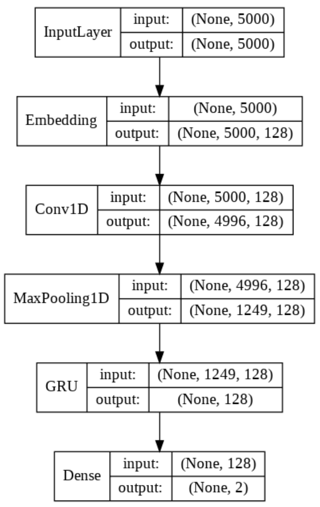

Figure 4.14: GRU-CNN Structure

### Training and Metrics

As reflected in Figure 4.15, after 40 iterations the train and test loss start to diverge while the accuracy remains to improve. To prevent over-fitting the training is limited to a 100 epochs on the LSTM and 80 on the GRU.

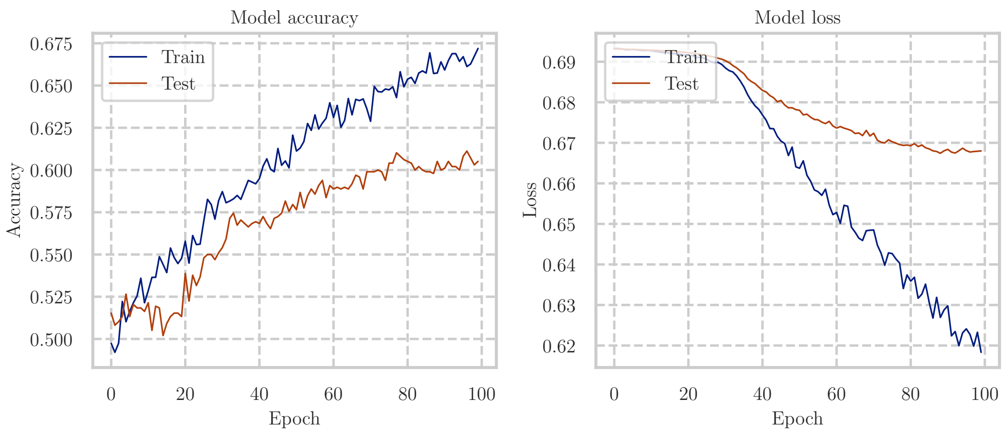

Figure 4.15: LSTM-CNN Network Training

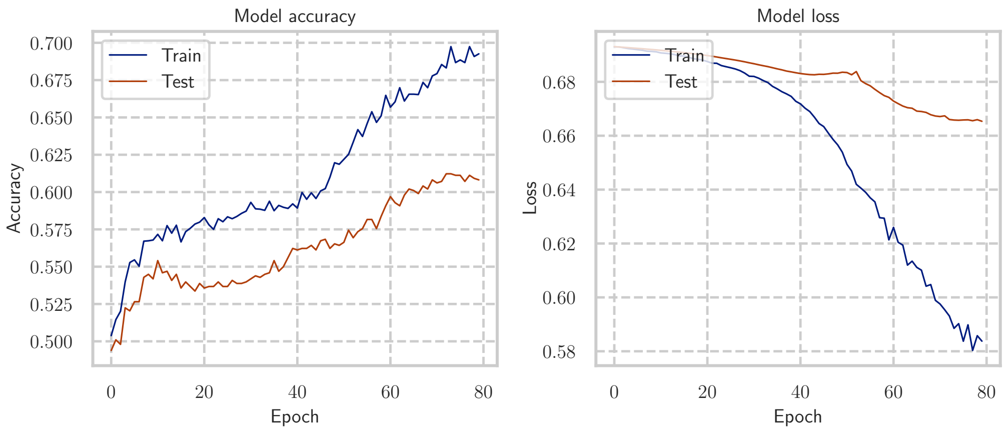

Figure 4.16: GRU-CNN Network Training

With the following metrics on the test set:

| AUROC | Accuracy | Precision | Recall | $F_1$ |
|---|---|---|---|---|
| 0.625 | 0.625 | 0.625 | 0.625 | 0.625 |

Table 4.19: LSTM-CNN Metrics

| AUROC | Accuracy | Precision | Recall | $F_1$ |
|---|---|---|---|---|
| 0.632 | 0.633 | 0.633 | 0.633 | 0.632 |

Table 4.20: GRU-CNN Metrics

Achieving considerably better performance than their counterparts without convolutional layers.

## 4.3 Transfer Learning Approach

Deep Learning algorithms and supervised learning rely on massive amounts of labelled data for model training, where in very specific domains it may be scarce. Transfer learning leverages data from a related domain. By training a model on similar data it can be reused and adapted to the desired and specific field. Using these pre-trained models as a baseline achieves faster training times and less task-specific data.

### 4.3.1 Pre-Trained BERT

Bidirectional Encoder Representations from Transformers (BERT) models [14] presented by Google in 2018 base their operation on the Transformers [13]. Whose architecture, as opposed to recurrent networks and their sequential nature, bases its functioning on attention. First proposed as an enhancement to RNN, the self-attention mechanism works with an Encoder-Decoder structure, where attention is a weighted vector representing the relationship between words. This architecture enables further parallelization as there is no recurrence.

The attention function relies on word embeddings and three matrices, the so-called Query ($Q$), Keys ($K$) of dimension $d_k$ and Values ($V$) on the form

$$\text{Attention}(Q, V, K) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}} V\Big) \tag{4.30}$$

obtained during training as a result of input and trainable weights multiplication. Where the softmax $\sigma$ function transforms a $k$ real values vector into a normalized vector of $k$ probabilities ($\sigma : \mathbb{R}^k \to \mathbb{R}^k$). Such that $\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$ for $i = 1, \ldots, K$.

BERT training follows two steps: pre-training and fine-tuning. The models is first pre-trained on unlabelled data and stored. This model serves as a baseline for posterior training or fine-tuning, where the model is initialized with the pre-trained configuration and parameters and fine-tuned on task-specific labelled data. Several variants of BERT are available for fine-tuning depending on model complexity and data used during training. We will focus on Base $\text{BERT}_{\text{BASE}}$ (110 million parameters) and a lighter version called DistilBERT [19] (66 million parameters), both pre-trained using BooksCorpus and English Wikipedia. All subsequent models are fine-tuned on a Tesla P100 GPU of 16GB, with batch size 10 and 16 respectively.

Models are initialised based on the pre-trained models and fine-tuned during 3 epochs with a learning rate of $3e^{-5}$.

| Model | Approx. Time (min) | AUROC | Accuracy | Precision | Recall |
|---|---|---|---|---|---|
| $\text{BERT}_{\text{BASE}}$ | 7 | 0.599 | 0.601 | 0.615 | 0.599 |
| DistilBERT | 15 | 0.610 | 0.605 | 0.624 | 0.610 |

Table 4.21: BERT Models Performance Comparison

Neither model improves the performance obtained with RNN. However, the training time is substantially reduced. Being 15 minutes for $\text{BERT}_{\text{BASE}}$ and 7 for DistilBERT.

This reduction in training time is associated with the increased complexity of the model and hardware requirements.

Language used in clinical notes differ greatly from pre-training datasets, with domain-specific words and abbreviations. Since BERT models are trained on language found in Wikipedia's articles and books there is a lack of clinical-specific vocabulary. Preventing fine-tuned models from fully leveraging previous extracted knowledge. Where a further pre-training of the model solely on clinical records or training from scratch would improve the performance.

# 5 Results

All results are listed on table 5.1, with the best result for each metric highlighted in bold. When comparing the performance of the different models, it can be observed that the deep learning models are outperformed by machine learning or statistical models. Where regression models, and in particular logistic regression, yield the best results with an AUROC of 0.71. By contrast, SVM has the highest recall, detecting higher number of 30-day readmissions.

| Algorithm | Accuracy | Precision | Recall | AUROC |
|---|---|---|---|---|
| Logistic Regression | **0.65** | **0.67** | 0.60 | **0.71** |
| Ridge Regression | 0.65 | **0.67** | 0.58 | **0.71** |
| Lasso Regression | 0.64 | 0.64 | 0.57 | **0.71** |
| Elastic Net | 0.64 | **0.67** | 0.57 | 0.70 |
| SVM | 0.60 | 0.59 | **0.70** | 0.65 |
| MLP | 0.61 | 0.61 | 0.68 | 0.65 |
| RF | 0.62 | 0.63 | 0.59 | 0.65 |
| Naïve Bayes | 0.62 | 0.64 | 0.57 | 0.66 |
| XGBoost | 0.64 | 0.64 | 0.57 | 0.64 |
| LSTM | 0.58 | 0.58 | 0.58 | 0.58 |
| GRU | 0.61 | 0.62 | 0.61 | 0.61 |
| LSTM-CNN | 0.63 | 0.63 | 0.63 | 0.63 |
| GRU-CNN | 0.63 | 0.63 | 0.63 | 0.63 |
| $\text{BERT}_{\text{BASE}}$ | 0.60 | 0.62 | 0.60 | 0.60 |
| DistilBERT | 0.61 | 0.62 | 0.61 | 0.61 |

Table 5.1: 30-day re-admission Prediction Results

Being the main difference between the deep learning and machine learning models the data processing. Where the former maintain the sequential order of words (thus allowing the establishment of dependencies between words) and the latter only processes the raw frequency of words. This difference highlights the limitations of models based on neural networks, requiring large amounts of information for their proper functioning. In limited data situations the simplicity of statistical models and machine learning provides better performance.

Results show great similarity with those found in the literature using only discharge notes as only predictors. However, in the case of DL-based models the results could be improved by using larger datasets, further development of transfer learning approaches and notes from different sources and databases. Additionally, inclusion of numerical data from patients at the time of discharge would improve the performance.

# 6 Conclusions

The prediction of re-admission of patients to the Intensive Care Unit is of great complexity and criticality. Where a patient's health may be conditioned by numerous factors. This thesis addresses the prediction of readmission using only discharge notes from patients at the time of discharge through the use of Natural Language Processing techniques.

Although the results shown are similar to those previously obtained in the literature, predicting readmission still has limitations. Where the complexity of vocabulary, terminology and schematic writing of notes by medical staff favours traditional models. The main impediment to neural network models being the lack of specialized information in the clinical setting.

There are several proposals that can be addressed in order to improve readmission prediction. The main approach involves improving text pre-processing, particularly focusing on the abundant use of acronyms and abbreviations, morphological standardization and mistake correction. Similarly, and not limited to the use of discharge notes, the inclusion of all notes associated with patients during their stay in the ICU and numerical health indicators. Finally, deepening transfer learning and pre-trained models in order to establish a strong prior medical knowledge.

However, the Natural Language Processing models allow performances similar to those obtained with numerical predictors reported in the literature. Which, combined with the greater simplicity of unstructured text data in comparison to relational databases, allows for the development of less cumbersome systems with even performance.

# Bibliography

[1] C. van Walraven, C. Bennett, A. Jennings, P. C. Austin, and A. J. Forster, “Proportion of hospital readmissions deemed avoidable: A systematic review”, *CMAJ*, vol. 183, no. 7, E391–E402, 2011, ISSN: 0820-3946. DOI: 10.1503/cmaj.101860.

[2] J. Benbassat and M. Taragin, “Hospital Readmissions as a Measure of Quality of Health Care: Advantages and Limitations”, *Archives of Internal Medicine*, vol. 160, no. 8, pp. 1074–1081, Apr. 2000, ISSN: 0003-9926. DOI: 10.1001/archinte.160.8.1074.

[3] B. Friedman and J. Basu, “The rate and cost of hospital readmissions for preventable conditions”, *Medical Care Research and Review*, vol. 61, no. 2, pp. 225–240, 2004, PMID: 15155053. DOI: 10.1177/1077558704263799.

[4] M. de Sanidad, “Revisión sistemática de eventos adversos y costes de la no seguridad”, *Informes, Estudios e Investigación 2015*, 2015.

[5] A. Rajkomar, E. Oren, K. Chen, A. M. Dai, N. Hajaj, M. Hardt, P. J. Liu, X. Liu, J. Marcus, M. Sun, *et al.*, “Scalable and accurate deep learning with electronic health records”, *NPJ Digital Medicine*, vol. 1, no. 1, p. 18, 2018.

[6] E. Craig, C. Arias, and D. Gillman, “Predicting readmission risk from doctors’ notes”, *arXiv preprint arXiv:1711.10663*, 2017.

[7] R. Veloso, F. Portela, M. F. Santos, A. Silva, F. Rua, A. Abelha, and J. Machado, “A clustering approach for predicting readmissions in intensive medicine”, *Procedia Technology*, vol. 16, pp. 1307–1316, 2014.

[8] R. G. Rosa, C. Roehrig, R. P. de Oliveira, J. G. Maccari, A. C. P. Antônio, P. de Souza Castro, F. L. D. Neto, P. de Campos Balzano, and C. Teixeira, “Comparison of unplanned intensive care unit readmission scores: A prospective cohort study”, *PloS one*, vol. 10, no. 11, 2015.

[9] J. C. Rojas, K. A. Carey, D. P. Edelson, L. R. Venable, M. D. Howell, and M. M. Churpek, “Predicting intensive care unit readmission with machine learning using electronic health record data”, *Annals of the American Thoracic Society*, vol. 15, no. 7, pp. 846–853, 2018.

[10] M. Loreto, T. Lisboa, and V. P. Moreira, “Early prediction of icu readmissions using classification algorithms”, *Computers in Biology and Medicine*, vol. 118, p. 103 636, 2020.

[11] S. Curto, J. P. Carvalho, C. Salgado, S. M. Vieira, and J. M. Sousa, “Predicting icu readmissions based on bedside medical text notes”, in *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, IEEE, 2016, 2144–a.

[12] K. Huang, J. Altosaar, and R. Ranganath, “Clinicalbert: Modeling clinical notes and predicting hospital readmission”, *arXiv preprint arXiv:1904.05342*, 2019.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need”, in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding”, *arXiv preprint arXiv:1810.04805*, 2018.

[15] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, “Mimic-iii, a freely accessible critical care database”, *Scientific data*, vol. 3, p. 160 035, 2016.

[16] M. F. Porter *et al.*, “An algorithm for suffix stripping.”, *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[17] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization”, *arXiv preprint arXiv:1412.6980*, 2014.

[19] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter”, 2019.