



UNIVERSIDAD TORCUATO DI TELLA

Master in Management+Analytics Thesis

---

# A Machine Learning Approach for Prediction of Hospital Bed Availability

---

by

**Josefina Dalla Via Monti**

Advisor: **Gustavo Vulcano, PhD**  
Co-Advisor: **Nicolás Schapchuck, MBA**

July, 2020



# Un enfoque de aprendizaje automático para predicción de disponibilidad de camas de internación

Tesis de Maestría en Gestión y Análisis de datos

Josefina Dalla Via Monti

## Resumen

Las camas de internación constituyen un recurso escaso en las instituciones hospitalarias, los datos, en cambio, no.

En el presente trabajo se argumenta que, haciendo uso de técnicas de aprendizaje automático, puede sacarse provecho del enorme volumen de data disponible en los sistemas de información de hospitales y sanatorios para construir soluciones de *analytics* que potencien la eficiente utilización de las camas de internación mediante la mejora del proceso de toma de decisiones.

Con el objetivo de poner a prueba esta hipótesis, se trabajó en conjunto con una de las instituciones hospitalarias más importantes de la ciudad de Buenos Aires. El foco del trabajo estuvo puesto en la construcción de un modelo de aprendizaje automático que pudiera predecir la probabilidad de que un paciente sea dado de alta en las próximas veinticuatro horas, en función de su historia clínica, datos demográficos y algunos otros factoriales ambientales. Para lograrlo se aplicaron técnicas de ingeniería de datos y aprendizaje supervisado, en el contexto de un problema de clasificación. Se experimentó con diferentes algoritmos así como formas de abordar la representación de atributos para sacar el máximo provecho de la data disponible.

Como resultado, se obtuvo un modelo con un rendimiento prometedor que alcanza un puntaje de 0.84 de *área bajo la curva ROC* y ha demostrado generalizar muy bien en datos desconocidos. Dicho modelo fue la base sobre la cual se montó una herramienta de pronóstico de altas. Esta solución permite obtener tres predicciones, con diferentes niveles de incertidumbre asociada, de las altas esperadas en el Sanatorio para la fecha especificada. Los "niveles de confianza" reportados fueron obtenidos mediante un ejercicio de simulación sobre la data histórica que permitió comparar el pronóstico de la herramienta con el escenario observado en la realidad.

El equipo de gestión de operaciones del hospital en cuestión ha hecho explícito su interés en la solución propuesta, ya que evalúan que el modelo tiene un enorme potencial para facilitar el proceso de planificación de camas y, de esta manera, ayudar a mejorar la eficiencia operacional del sanatorio.



# A Machine Learning Approach for Prediction of Hospital Bed Availability

Master in Management + Analytics Thesis

Josefina Dalla Via Monti

## Abstract

Hospital beds are a scarce resource for healthcare facilities, data is not. In this thesis, we argue that machine learning techniques could take advantage of the abundant amount of data available at hospitals information systems in order to build analytics solutions that could propel the efficient utilization of beds by improving the management decision making process.

In order to test this hypothesis we have worked together with one of the most relevant medical institutions in Buenos Aires. The focus of our work has been placed in building a machine learning model that could predict the probability of a certain patient being discharged during the following twenty four hours, based on his medical records as well as his demographic data and some environmental factors. To this aim, data engineering and supervised learning techniques have been applied in the context of a classification task. We have experimented with different algorithms as well as feature representation approaches to make the most out of the data at hand.

As a result, a model with a promising performance of 0.84 *AUC-ROC score* was obtained, and its results have demonstrated to generalize quite well on unseen data. This model was the base on top of which a discharges forecaster tool was developed. This solution is able to return three different predictions of the hospital discharges for a specified date with different "confidence levels" associated, thus providing management with a risk-informed prediction of hospital beds availability. The "confidence" reported for each of the forecasts was obtained using a simulation approach for historic data where we were able to contrast the forecast output with the actual scenario.

The hospital management team has made explicit its interest in the solution, as they assess it has an enormous potential for facilitating the bed planning process and by doing so improving the hospital operational efficiency.



To Julián, for his unconditional love.  
To Rafael, a light in the middle of the darkness.





# Contents

<b>Abstract</b>	<b>v</b>
<b>Dedication</b>	<b>vii</b>
<b>Contents</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.1.1 Hospital Bed Management . . . . .	1
1.1.2 Machine Learning . . . . .	2
1.2 Justification . . . . .	3
1.3 Objective . . . . .	3
<b>2 Methods &amp; Procedures</b>	<b>5</b>
2.1 CRISP-DM Methodology . . . . .	5
2.2 Data . . . . .	8
2.2.1 Exploration . . . . .	9
2.2.2 ETL . . . . .	18
2.3 Machine Learning Techniques . . . . .	23
2.3.1 Supervised Learning Algorithms . . . . .	23
2.3.2 Machine Learning Model Evaluation . . . . .	28
2.3.3 Feature Engineering Techniques . . . . .	33
2.4 Software . . . . .	35
<b>3 Results</b>	<b>37</b>
3.1 Model Experiments . . . . .	37
3.2 Model Selection . . . . .	41
3.3 Final Model Assessment . . . . .	50
<b>4 Discussion</b>	<b>53</b>
<b>5 Conclusions and Recommendations</b>	<b>57</b>
5.1 Project Achievements . . . . .	57
5.2 Limitations . . . . .	58
5.3 Management Recommendations . . . . .	60

A List of variables in each dataset	65
B Data Quality Reports	73
C Data Quality Plan	81
D Feature importance plots for different model experiments	87
E Simulation results	91
F Forecaster output example	93
 Bibliography	 95

# Chapter 1

## Introduction

Hospital Operations Management around the world is devoted to one simple but hard to achieve goal: make the most efficient use of hospital scarce resources. These go from expensive and very specific health machinery assets to the most qualified and specialized physicians, including, of course, hospital beds.

Indeed, bed capacity management is one of the key activities of hospital operations management. Poor bed management may result in either overcrowded emergency rooms, valuable resources out of use or treatment limitations for critical patients, just to name a few. Patient dissatisfaction is the immediate result, leading to a detrimental effect for the health unit reputation in the long term. Thus, optimization of hospital bed management is key for hospital's success.

While hospital beds are a scarce resource for a hospital, data is not. There might be exceptions but, in general terms, hospitals' information systems are flooded with data. Each time a patient is hospitalized, each study performed, each symptom informed by the patient, each diagnosis received, each physician a patient was treated by, everything is recorded in the hospital's database with its corresponding timestamp. Data is everywhere, but most of the time, it is misused by management.

Can this abundant amount of data propel the efficient utilization of those scarce hospital beds? Our hypothesis is that it surely can.

## 1.1 Background

### 1.1.1 Hospital Bed Management

Bed management can be defined as: *keeping a balance between flexibility for admitting emergency patients and high bed occupancy [which] has been an indicator of good hospital management* (Green and Armstrong, 1994). It concerns the placement of emergency admissions, but may extend to balancing between emergency and elective admissions since the two may require the same resources.

Essentially, bed management can be viewed as the process of matching up demand for beds with the supply of beds, in a completely dynamic context in which both, supply and demand, are changing minute by minute. Demand changes together with the inflow of emergency patients, while variations in supply are determined by the evolution of discharges.

According to Infosys, a multinational consulting firm, there are two main constraints to hospital management efforts to optimize bed allocation: lack of powerful forecasting tools and lack of real-time information flows. *Even today, hospitals use rule of thumb or rudimentary forecasting methods to predict patients inflow [outflow]. Since capacity and resource planning are done based on such forecasts, any inaccuracy results in an under or over-allocation of both* (Infosys, 2018).

### 1.1.2 Machine Learning

Machine learning refers to a category of algorithms that use statistics to find patterns in massive amounts of data. Leveraged by the revolution of big data, the application of these techniques to different industries, powers the vast majority of the current artificial intelligence advancements: recommendation systems in *e-commerce*, feeds customization in social media, delivery time prediction for logistics, just to name a few.

Moreover, machine learning supervised algorithms have proved extremely useful in forecasting. Supervised learning is a subset of the machine learning universe in which the learning process is led by a target variable, as opposed to unsupervised learning, in which data has no labels and thus algorithms just look for whatever pattern they may find. Efficiency of supervised learning in forecasting tasks has been reflected several times (Bontempi et al, 2013; Di Piazza et al, 2016; Krollner et al, 2010). In this sense, supervised learning appears as an opportunity to tackle the hospital management problems regarding the lack of accurate forecasts that has been previously explained.

However, the application of machine learning to healthcare operations management is not a widespread practice and even today still remains in its early stages. While there clearly exists a need for advanced analytics tools, hospital management remains reluctant to delegate its decisions on an automated, self-learned model. As an example, Glintt (Global Intelligent Technologies), a Portugal-based leading IT solutions company in the healthcare area claims to have developed, together with IBM, WiseWard, a solution that uses predictive analytics to improve bed assignment decisions. However, the current version relies completely on analytical and business rules rather than artificial intelligence (Forneas, 2018).

In the academic field, a paper published in March 2019, took advantage of recursive neural networks to forecast monthly bed occupancy levels in medium scale hospitals (Kutafina et al, 2019). Another paper reports the development of a data-mining-based model using only the number of patients hospitalized and the respective date, with an acceptable performance on predicting weekly discharges. There are also examples of

machine learning models applied to the prediction of length of stay (Davis and Lowell, 1999; Pendharkar and Khurana, 2014; Turgeman et al, 2017; Walczak et al, 1998).

## 1.2 Justification

While interesting, the above described approaches for supervised-learning forecasting do not make use of all the available patient's data which we believe has enormous predictive potential. Moreover they provide acceptable predictions at an aggregate level, but they tell little about how likely is that a certain bed (in certain conditions) might be idle in the subsequent hours.

Our proposal is to build a classification model, to predict the probability of a given patient being discharged in the following twenty four hours, conditional on his entire medical history. While ambitious, this objective is aligned with the needs of hospital management which, on a daily basis, has to balance supply and demand of each bed aiming to make the most efficient use of the resources. It will not only let the hospital management team have a prediction for the aggregate discharges, but also to have a probabilistic measure of how likely is that each bed will be liberated and consequently, the resources associated with each bed. Thus, while aiming at improving bed capacity management, the model could also be integrated with other scarce resources management systems within the hospital unit.

To our best knowledge, by the time of submitting this document, no scientific work using the described machine learning approach is currently available.

## 1.3 Objective

The objective of this project is to develop a machine learning model that, by taking advantage of the full hospitals database, is able to predict, with an acceptable performance, the odds that a given patient will be discharged during the following day. This being said, the problem will be approached as a classification task and, to this aim, ensemble methods will be applied. This type of methods constitutes the gold standard technique for tabular data in the industry due to its optimal trade-off between performance and training speed.

Moreover, we expect not only to provide the model and its predictions, but to understand how the different features impact on the probability of discharge. To this end, we will take advantage of visualizations as well as simulation analysis. We aim to prevent the black-box effect associated with machine learning approaches, thus facilitating management adoption of our model as a business solution.



## Methods & Procedures

The CRISP-DM (Cross-industry standard process for data mining) provides a structured approach to planning a data mining/machine learning project. This framework was conceived in the late 1996 by representatives from SPSS, Teradata, Daimler, NCR and OHRA, when the data mining market was still young and immature. Their guidelines remain widely-used in most analytics models nowadays and were used to guide most part of this project.

The diagram illustrates the Machine Learning lifecycle as a continuous, iterative process. It features a central cylinder labeled "Data" representing the data source. The lifecycle is composed of several interconnected stages arranged in a circle:

- Business Understanding** and **Data Understanding** are connected by a double-headed arrow, indicating a reciprocal relationship.
- Data Understanding** leads to **Data Preparation** via a downward arrow.
- Data Preparation** and **Modeling** are connected by a double-headed arrow, indicating an iterative relationship.
- Modeling** leads to **Evaluation** via a downward arrow.
- Evaluation** leads to **Deployment** via an upward arrow.
- Deployment** leads back to **Business Understanding** via an upward arrow, completing the cycle.

A large, thick, curved arrow on the outer perimeter of the diagram points clockwise, signifying the overall progression of the machine learning process.

5

The initial phase, Business Understanding, focuses on understanding the project objectives and requirements from a business perspective and then translating this into a Machine Learning problem definition together with a preliminary plan to achieve the objectives.

Business Understanding	Data Understanding	Data Preparation	Modeling	Evaluation	Deployment
<b>Determine Business Objectives</b> <i>Background</i> <i>Business Objectives</i> <i>Business Success Criteria</i>	<b>Collect Initial Data</b> <i>Initial Data Collection Report</i>	<b>Select Data</b> <i>Rationale for Inclusion/Exclusion</i>	<b>Select Modeling Techniques</b> <i>Modeling Technique</i> <i>Modeling Assumptions</i>	<b>Evaluate Results</b> <i>Assessment of Data Mining Results w.r.t. Business Success Criteria</i> <i>Approved Models</i>	<b>Plan Deployment</b> <i>Deployment Plan</i>
<b>Assess Situation</b> <i>Inventory of Resources</i> <i>Requirements, Assumptions, and Constraints</i> <i>Risks and Contingencies</i> <i>Terminology</i> <i>Costs and Benefits</i>	<b>Describe Data</b> <i>Data Description Report</i>	<b>Clean Data</b> <i>Data Cleaning Report</i>	<b>Generate Test Design</b> <i>Test Design</i>	<b>Review Process</b> <i>Review of Process</i>	<b>Plan Monitoring and Maintenance</b> <i>Monitoring and Maintenance Plan</i>
<b>Determine Data Mining Goals</b> <i>Data Mining Goals</i> <i>Data Mining Success Criteria</i>	<b>Explore Data</b> <i>Data Exploration Report</i>	<b>Construct Data</b> <i>Derived Attributes</i> <i>Generated Records</i>	<b>Build Model</b> <i>Parameter Settings</i> <i>Models</i> <i>Model Descriptions</i>	<b>Determine Next Steps</b> <i>List of Possible Actions</i> <i>Decision</i>	<b>Produce Final Report</b> <i>Final Report</i> <i>Final Presentation</i>
<b>Produce Project Plan</b> <i>Project Plan</i> <i>Initial Assessment of Tools and Techniques</i>	<b>Verify Data Quality</b> <i>Data Quality Report</i>	<b>Integrate Data</b> <i>Merged Data</i>	<b>Assess Model</b> <i>Model Assessment</i> <i>Revised Parameter Settings</i>		<b>Review Project Experience</b> <i>Documentation</i>
		<b>Format Data</b> <i>Reformatted Data</i>			
		<i>Dataset</i> <i>Dataset Description</i>			

Figure 2.2: Generic tasks and outputs of the CRISP-DM reference model (SPSS, 2000)

The second phase, Data Understanding, embraces data collection and exploration. The ultimate goal of this phase is to become familiar with the data, identify data quality problems and uncover some first insights from the data. During this insightful exploration, some new hypothesis might flourish, sometimes requiring to redefine the business problem (thus going back to the first phase).

The Data Understanding phase is followed by Data Preparation, which covers all activities needed to construct the final dataset: data transformation and cleansing, feature engineering and feature selection.

After finishing these tasks, the Modeling phase comes. In this phase several modeling techniques are applied and their parameters are calibrated. As different modeling techniques might have different requirements on the forms of the data, going back to the Data Preparation phase is often necessary.

After having built one or more high quality models, the Evaluation phase comes in. Before proceeding to Deployment, the steps executed to create the model should



be reviewed and evaluated to make sure that it properly achieves the business objectives. The outcome of this phase should be a decision on the use of the achieved results.

The last phase of the framework is Deployment. This stage involves applying “live” models within an organization’s decision making process so that the Machine Learning customers can take advantage of it. Depending on the case, this phase can be as simple as generating a report or as complex as implementing a real-time Machine Learning process across the company.

Rather than just splitting the process into stages, the CRISP-DM methodology proposes a hierarchical breakdown into four levels of abstraction that goes from general to specific as illustrated in the Figure 2.3.

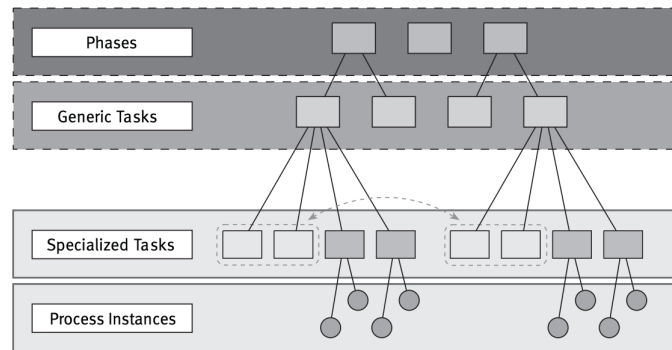


Figure 2.3: Four level breakdown of the CRISP-DM methodology (SPSS, 2000)

At the top level, the data mining process is organized into the different phases previously described. Each phase consists of several second-level generic tasks. The generic tasks are intended to be as complete and stable as possible. Complete means covering both the whole process of data mining and all possible data mining applications. Stable means that the model should be valid for yet unforeseen developments like new modeling techniques.

The third level, the specialized task level, is the place to describe how actions in the generic tasks should be carried out in certain specific situations. For example, at the second level there might be a generic task called clean data. The third level describes how this task differs in different situations, such as cleaning numeric values versus cleaning categorical values, or whether the problem type is clustering or predictive modeling.

The fourth level, the process instance, is a record of the actions, decisions, and results of an actual data mining engagement. A process instance is organized according to the tasks defined at the higher levels, but represents what actually happened in a particular engagement, rather than what happens in general (Chapman et al, 2000).

## 2.2 Data

The data used for the development of the project has been provided by one of the most relevant hospitals in Argentina under a *non-disclosure agreement*. We were granted access to the full hospital's database. However, not all the hospital data was of interest for our problem. The data we analyzed and took advantage of for these project includes:

- Patient admissions dataset

This dataset contains information corresponding to all hospitalizations. Each row represents an admission which is identified by a unique *admission-id*, and a *patient-id* (that is shared among the different hospitalizations of a certain person). For each admission there is data regarding patient's demographics, the diagnosis received, physicians who treated the patient, admission conditions, etc.

- Laboratory studies dataset

This dataset contains information regarding the laboratory studies performed to the hospital's patients. Each row represents a set of laboratory tests requested at a certain point in time and it is linked to a *patient-id*, an *admission-id* and the corresponding date. For each laboratory test there is data regarding specifications of the tests required, physicians who requested the study and datetime variables.

- Images studies dataset

This dataset contains information regarding the images studies performed to hospital's patients. Each row represents a set of images requested at a certain point in time and it is linked to a *patient-id*, an *admission-id* and the corresponding date. For each image study, there is data regarding specifications of the images required, physicians who requested the study and datetime variables.

- Surgeries dataset

This dataset contains information regarding all surgeries performed at the hospital. Each row represents a surgery and it is linked to a *patient-id*, an *admission-id* and the corresponding date. For each surgery there is data regarding the patient's previous and post-surgery conditions, specifications on the performed surgery and its requirements, physicians and related staff that were involved in the surgery and datetime variables.

- Sectors admissions dataset

This dataset contains information regarding patients flow across different hospital sectors. Each row represents an admission to a particular sector and it is associated with a *patient-id* and an *admission-id*.

- Hospital sectors dataset

This dataset reflects the structure of the different Sectors in which the hospital is organized.

A full list of the available variables in each dataset is included in Appendix A.

### 2.2.1 Exploration

Data exploration is a key part of both the Data Understanding and Data Preparation phases of CRISP-DM. According to Kelleher, Mac Namee and D’Arcy (2015), its main objectives are two:

- Detecting data quality issues that could adversely affect the models: missing values, outliers, irregular cardinality.
- Getting to know the data: understanding characteristics such as the types of values a feature can take, the ranges into which the values in a feature fall, and how the values in a dataset for a feature are distributed across the range that they can take.

#### Data Quality

One of the most important tools in exploratory analysis for detecting data quality issues is the data quality report. This report includes tabular reports (one for continuous features and one for categorical features) that describe the characteristics of each feature in the data using standard statistical measures of central tendency and variation.

According to Kelleher et al (2015), the table in a data quality report that describes continuous features should include a row containing the minimum, first quartile, mean, median, third quartile, maximum, and standard deviation statistics for that feature as well as the total number of instances in the dataset, the percentage of instances in the dataset that are missing a value for each feature and the cardinality of each feature. The table in the data quality report that describes categorical features should include a row for each feature in the dataset that contains the most frequent levels for the feature (for example the mode and second mode) and their frequency. Each row should also include the percentage of instances in the dataset that are missing a value for the feature and the cardinality of the feature.

The data quality report should also include a histogram for each continuous feature in the dataset. For continuous features with cardinality less than 10, it is suggested to use bar plots instead of histograms as this usually produces more informative data visualization. For each categorical feature in the dataset a bar plot should be included in the data quality report (Kelleher et al, 2015).

We have followed these guidelines in order to conduct our data quality analysis. The `Dataset` class, from `thesis_lib` Python’s module, provides a method called `get_quality_report()` that generates a complete quality report for both the categorical as well as the numerical variables in the dataset at stake, including distribution plots and tabular data for each feature.

An example of the Data Quality report provided by the `get_quality_report()` method, following the aboved described guidelines, is included in Appendix B.

After analyzing the data quality reports, some data quality issues were identified. Following the *best-practices* provided by Kelleher et al, in Fundamentals of Machine Learning for Predictive Analytics (2015), a comprehensive data quality plan was built. We have summarized here some of the main data quality issues found and how we approached each of them. For a more detailed view of the full Data Quality Plan see Appendix C.

- Structural missing values

Several variables have missing values due to valid data. For example, the variable *previous\_sector* in the admissions dataset (which refers to the previous hospital sector in which the patient was hospitalized) has 98% of missing values, because most patients don't have a previous admission. While this does not represent a problem in terms of the quality of the information, it is a problem for machine learning models as most algorithms do not handle missing values. Different approaches could be followed here depending on the case. As most variables dealing with this issue are categorical, in the Data Quality Plan we have suggested to generate a new category for the missing values, as the simple fact that data is missing for a certain datapoint can be informative itself.

- Missing values due to data collection errors

We have identified a collection of variables that have more than 60% of missing values with apparently no reason for the value to be missing, we assumed that these represent data collection issues. Unfortunately, in these situations, we will consider discarding the features as we consider they will not provide enough information for a model to learn.

- Extremely high cardinality

There are some categorical variables with as many levels as rows in the table. These are usually keys (like *patient\_id* or *surgery\_id*) that are useful to understand the relationship between different tables in the hospital's database but do not carry intrinsic meaning. This type of variables must be excluded from the model, as learning anything from them would mean that the model is overfitting the training data. In other cases, there are high cardinality variables which are certainly not keys (for example *presumptive\_diagnosis* or *scheduled\_surgery*). We have identified that in these cases, the variable levels are not standardized (there are many labels that actually correspond to the very same category but they are encoded differently, probably due to data collection errors). Thus, we should standardize their levels when possible, by manually detecting the different labels that map to the very same level.

- Inappropriate data types

This is a common issue, especially when data has been exported to different file formats as has been the case with our database. The most noticeable inappropriate data types variables in our data are time related variables which are treated as categorical (for example: *surgery\_starttime*, or *surgery\_end\_time*). These variables should be converted to the appropriate data type.

- Large number of outliers

As illustrated in the boxplots of our data quality report (see Appendix B), many of the continuous variables in the data presented distributions with an important number of outliers. At a first sight, some of them may look like invalid data (for example a patient having 134 surgeries or a surgery taking 1439 minutes long). However, digging into the data, we have found evidence of these data points being consistent. In the literature there is no strong take in regard to how to approach these cases. The only concern related to this type of outliers (valid outliers) is that some machine learning techniques do not perform well in the presence of extreme values, so we will just take notes of this for possible handling later on.

- Invalid outliers

We have also identified some variables with invalid outliers. This is the case, for example, for the variable *new\_born\_gestation\_age* in the admission dataset, which has a bimodal distribution with peaks of forty weeks (the standard gestation length) and zero weeks. All cases with a zero value on this field are not newborn patients, meaning that they should actually have a missing value instead. For this case, we considered it would not be problematic as, despite its invalidism, the zero value will be a sign for the learning algorithm that the patient is not a newborn.

In other cases, were we detected invalid outliers could be problematic, we proceeded to discard the corresponding datapoints or discard the variable itself (if we considered the proportion of invalid outliers was arbitrarily high).

## Data Understanding

Besides getting to know the available data, its distribution and quality issues, the exploratory stage was insightful in terms of finding relationships between the different variables available in the data and our objective: predicting discharges. We made special emphasis on the variables that correlated with the admission length. Here we present our main findings from the exploratory phase:

- Hospitalization length and patient age

As common sense will dictate, there is an evident relationship between patients' length of stay and their age. While not unexpected, it was pleasing to observe such a clear pattern in the data. Figure 2.4 illustrates how older people tend to be hospitalized for longer periods of time, as far as they are not too old, in which case hospitalization length starts to decrease (probably due to sooner death).

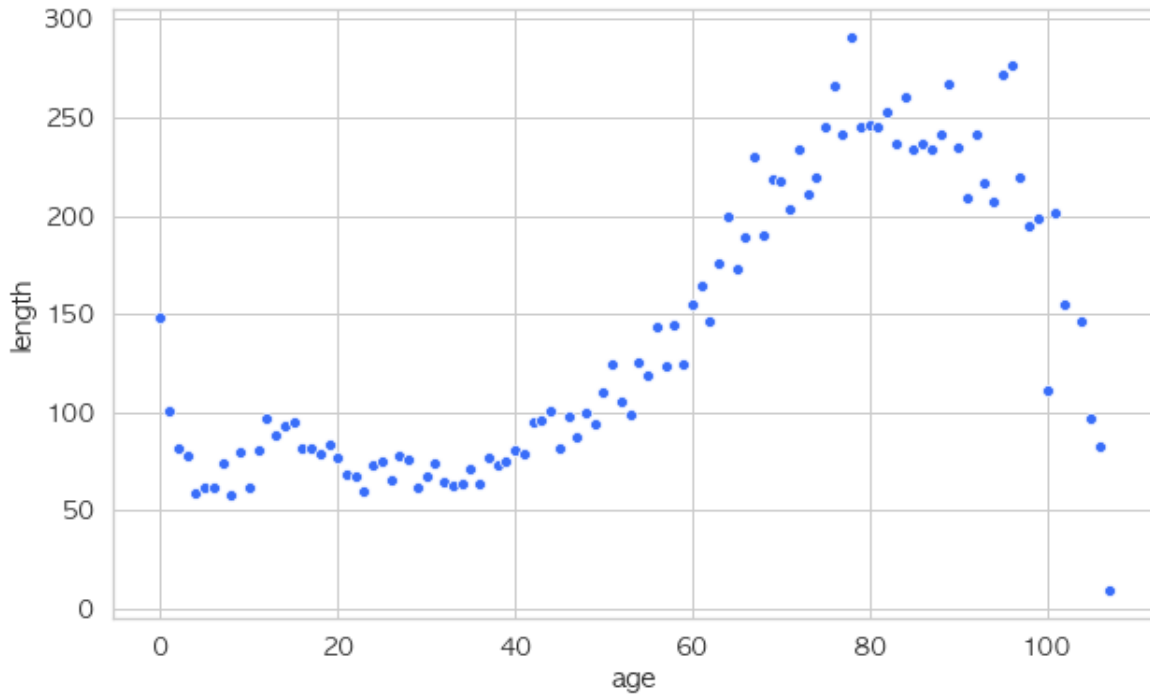


Figure 2.4: Avg. Hospitalization length (hours) vs. Patient age

- Hospitalization length and medical insurance

Another compelling relationship found in the data is the one between patients' length of stay and their medical insurance service (*entity\_group*). The extremely high expected hospitalization length that patients from PAMI exhibit is particularly notorious (2.61 times the hospitalization length general average). This was not a surprise: PAMI is the public health insurance agency for retirees in Argentina, thus the average PAMI patient is older and the rest of the patients which may explain why they tend to stay longer hospitalized. However, there might be another underlying reason explaining this relationship. According to the insights gathered from our conversations with the hospital management team, PAMI, as many public agencies in Argentina, is known for its bureaucratic processes and poor operational performance, deriving in many patients being left at the hospital for much longer time than needed.

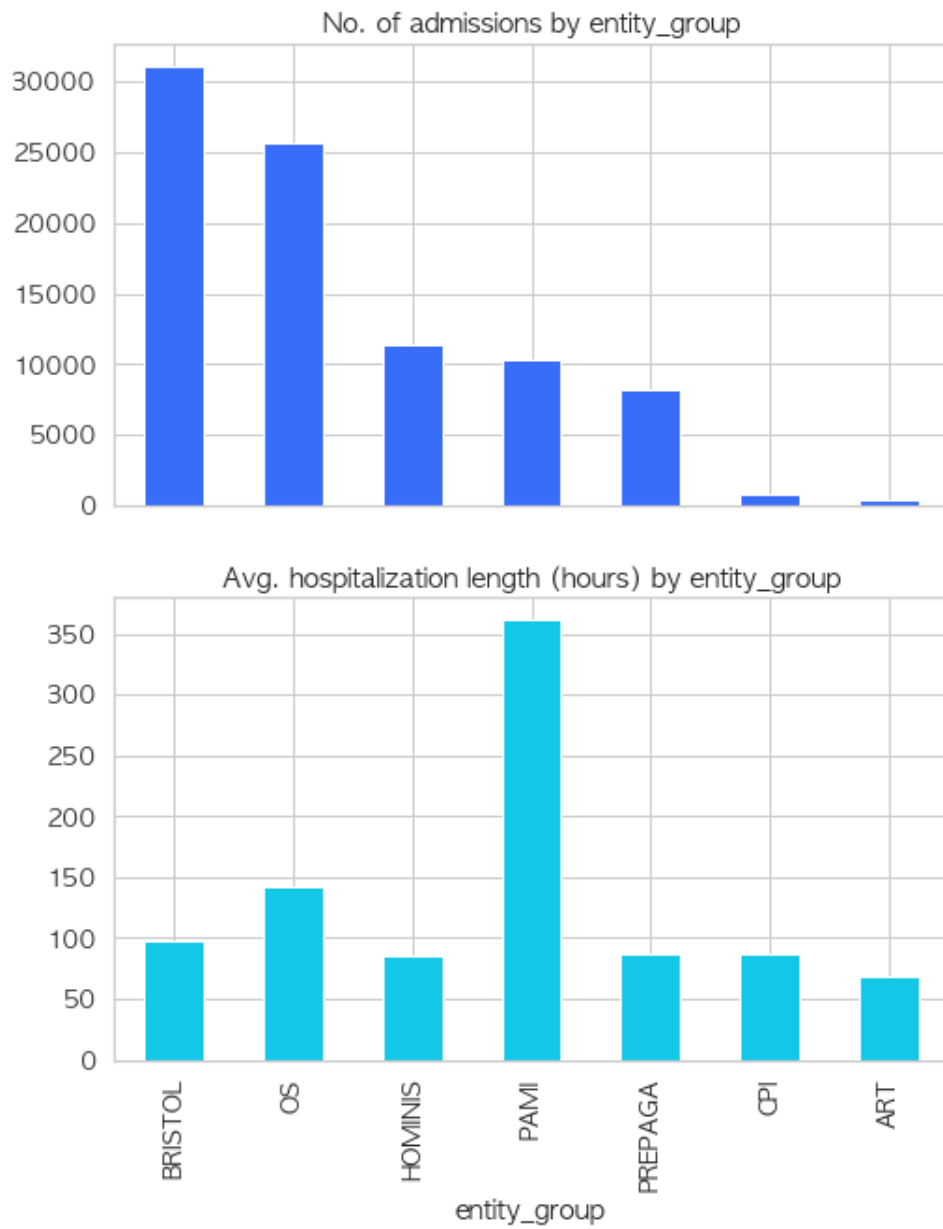


Figure 2.5: Hospitalization length and medical insurance

- Hospitalization length and types of surgeries

Another question posed during the exploratory stage was whether there was any relationship between the surgeries received by the patient during his admission, and the length of his stay at the hospital. As Figure 2.6 exhibits, hospitalization length varies widely among the different types of surgeries patients might undertake, suggesting this would be an important feature for the model to be taken into account.

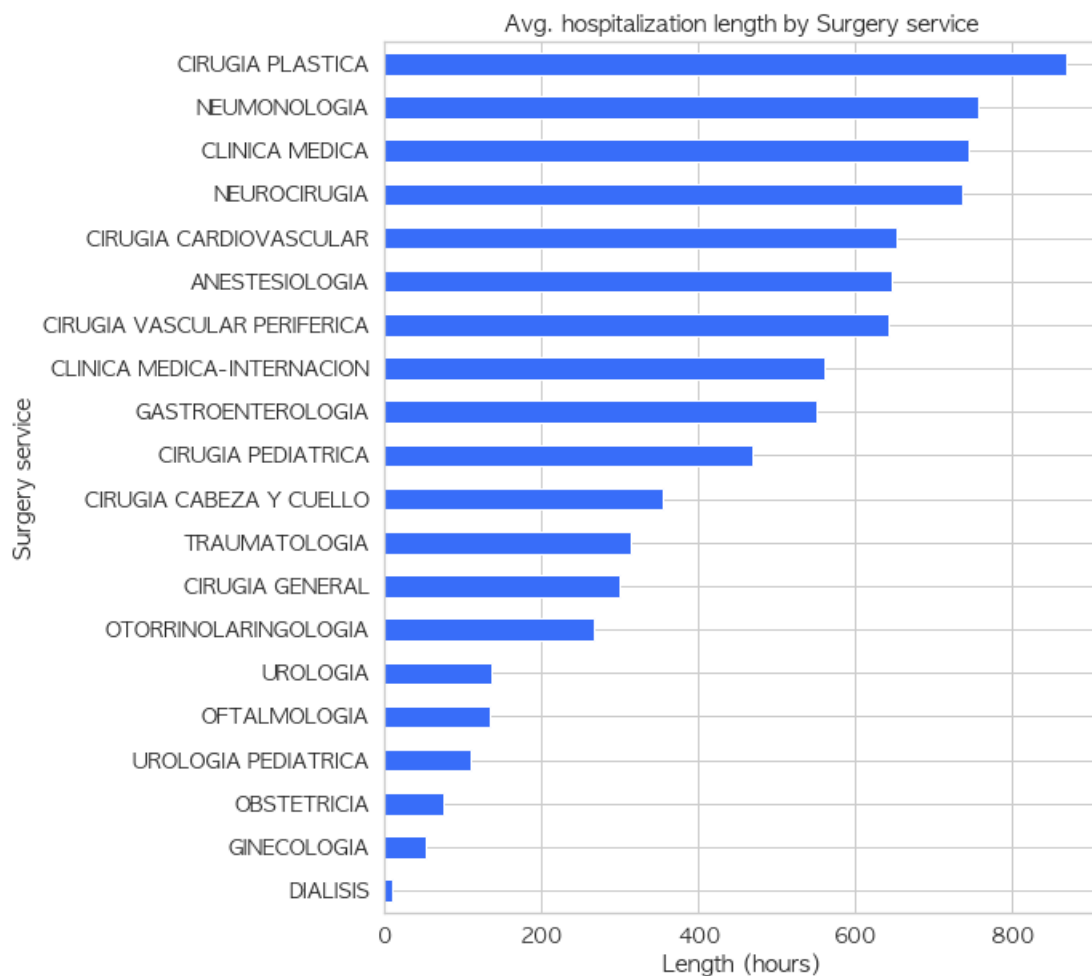


Figure 2.6: Hospitalization length by surgery type

- Hospitalization length and day of the week

A further insightful finding was the one concerning the day of the week in which the admission takes place. Patients who are hospitalized during the weekend (days 5 and 6) should expect to stay 19-23% longer at the hospital than the average as shown in Figure 2.8. This can be explained by an overall decline in the hospital activity during the weekend.

Figure 2.7 reflects how both, discharges and admissions, sink during the weekends. While the admissions decrease has certainly to do with a lower demand during this time of the week, this explanation does not hold for the fall in discharges. Instead, it seems to be propelled by the sanitary staff itself by anticipating discharges before the weekend.

This certainly represents an opportunity for the hospital management in terms of balancing the hospital occupation along the week by redirecting the demand to the weekends and thus being able to decongest the systems during the weekdays (Monday-Friday). They could so, for example, by scheduling non-emergency hospitalizations (like prearranged surgeries) on Saturdays and Sundays.



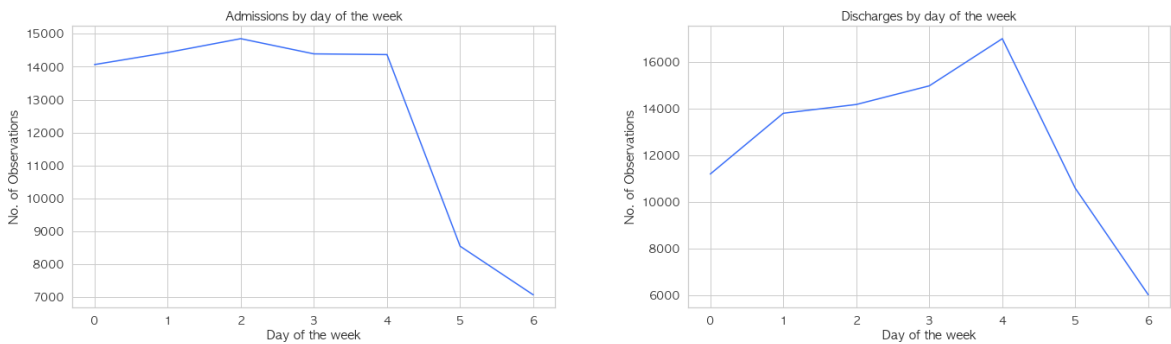


Figure 2.7: Global caption.

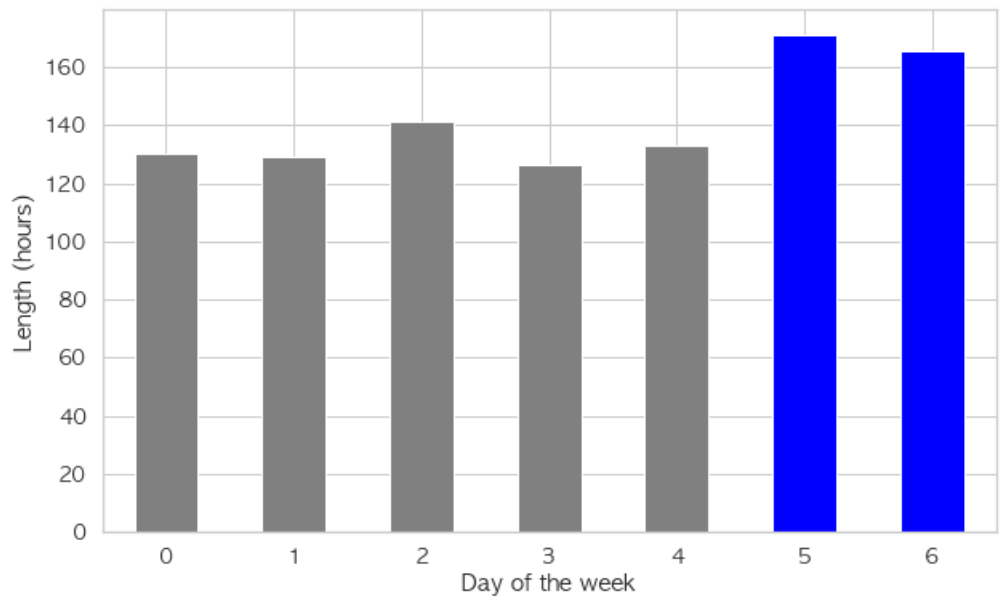


Figure 2.8: Hospitalization length by admission day of the week.  
Week goes from Monday (0) to Sunday (6).

- Hospitalization length and request sector

There are also significant differences in patient hospitalization length based on the hospital sector that requested the admission. For instance, patients who are directly admitted to Intensive Care Unit (ICU) and premature newborns (Neonatology Unit) have a longer expected hospitalization. They also represent a small fraction of the patients admitted as shown in Figure 2.10.

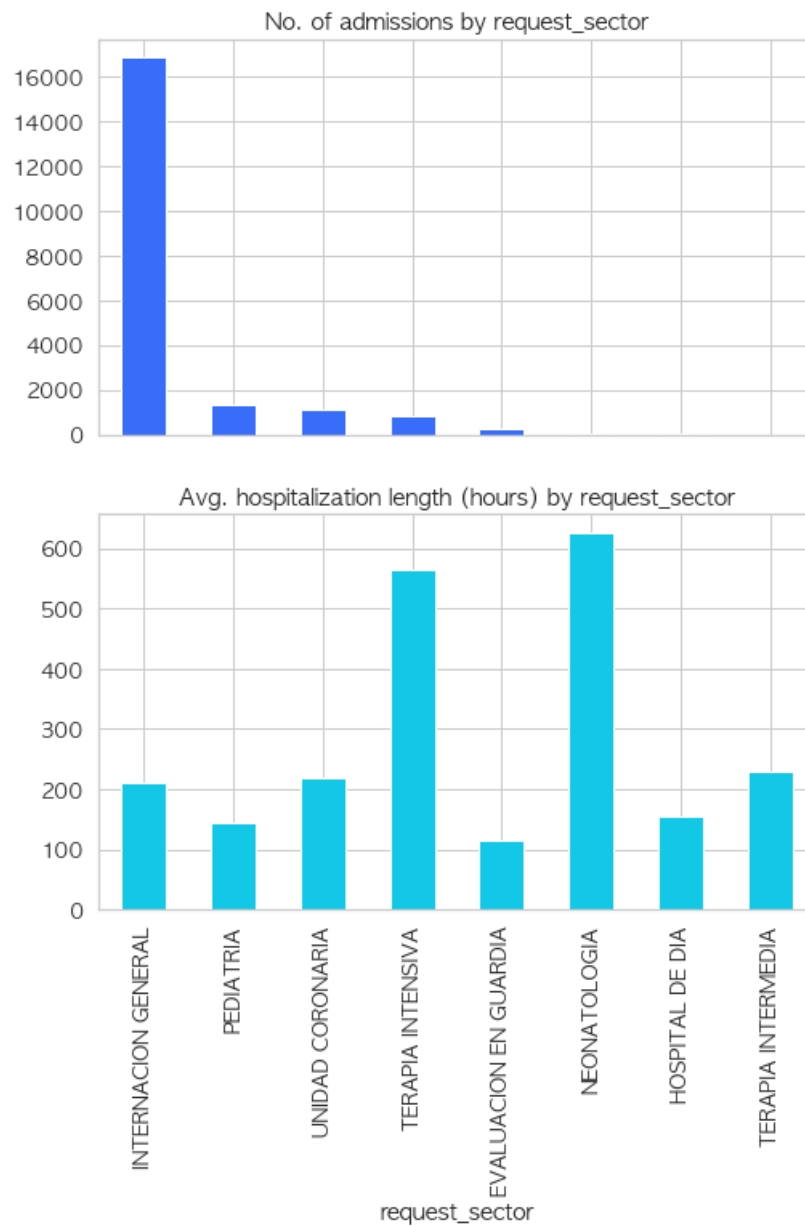


Figure 2.10: Hospitalization length and request sector

- Hospitalization length and patient origin

With regard to patient origin (where does the admission come from), there are significant differences in hospitalization length as well, as Figure 2.11 reflects. It is particularly notorious how shorter scheduled hospitalizations are (labeled as *Programada*). Emergencies (*Emergencias*) appears as the category from which the longest admissions come from, with an average hospitalization length of almost 9 days (209 hours). Moreover, admissions from this origin represent 38% of the hospitals admissions in the dataset.

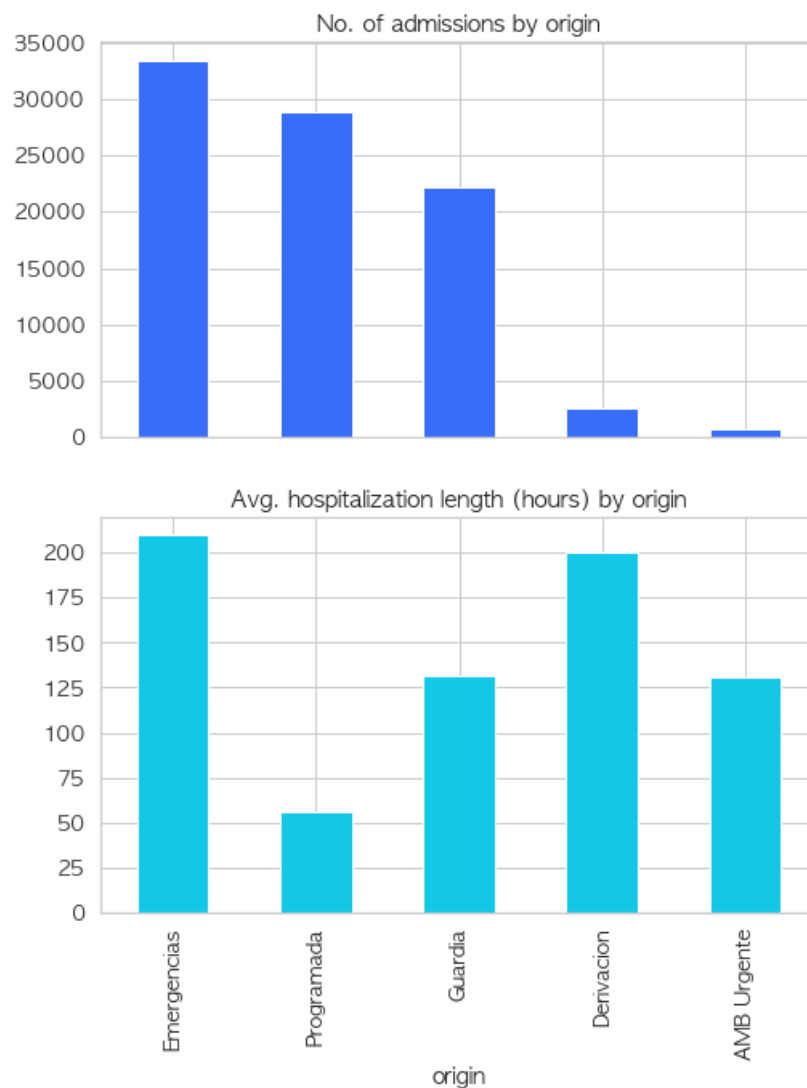


Figure 2.11: Hospitalization length and patient origin

- Hospitalization length and laboratory studies

Last but not least, from the analysis of the laboratory dataset in conjunction with the admissions dataset we were able to plot the relationship between the number of times a certain laboratory test is performed to patients and the average hospitalization length of these patients (Figure 2.12). The underlying hypothesis here was that some unusual, less frequent study types, when performed to a patient just a few times would be an indicator of a complicated health condition, thus causing the patient staying for a long period hospitalized. Studies like *Potasio en Sangre* or *Proteinas totales* seem to follow this pattern.

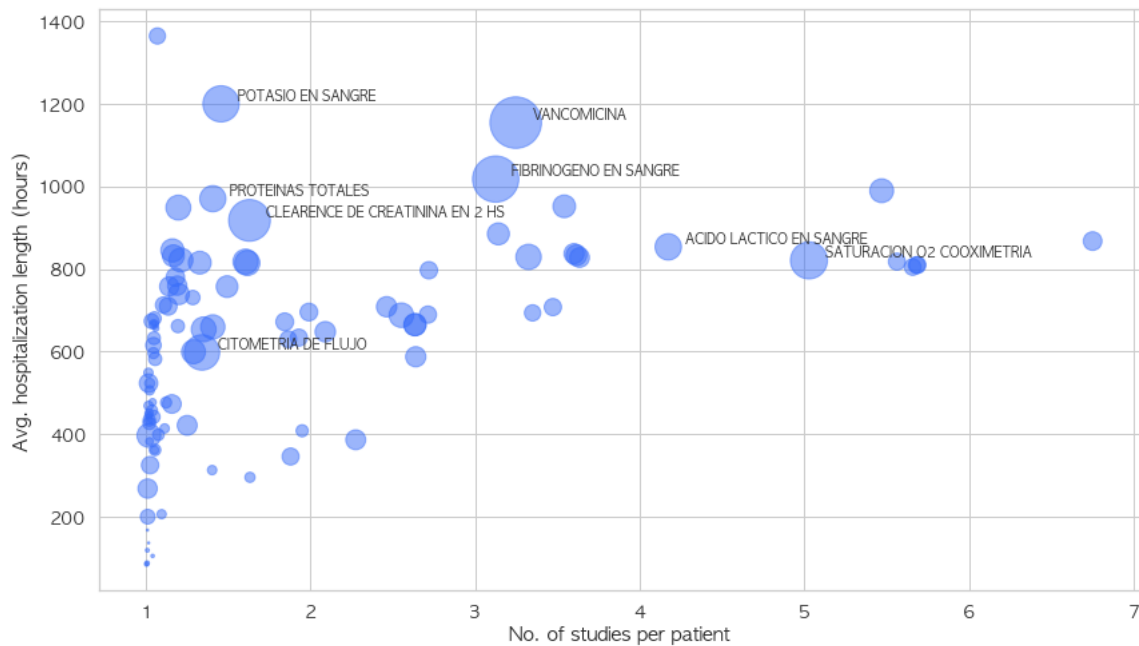


Figure 2.12: Hospitalization length vs. No. of studies per patient by Study type. Circle size represents confidence in the mean hospitalization length estimate (sample mean).

### 2.2.2 ETL

In order to combine the data coming from the hospital's database tables into a single dataset with the necessary schema to address our classification task and ETL process was designed.

ETL technology (that stands for Extract, Transform and Load) is a data integration process used to blend data from multiple sources. During this process, data is extracted from a source system, converted into a format that is appropriate for the analytics task and then stored (loaded) into a destination system. For the purpose of this project, a customized ETL process was designed using Python with an object oriented programming approach.

Figure 2.14 chart explains the information flow in the process ETL and the different tasks performed.

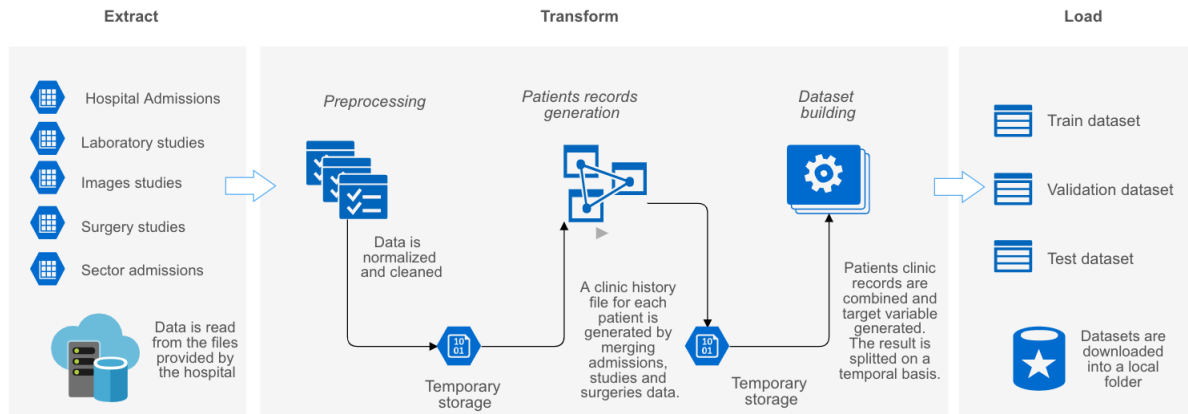


Figure 2.14: ETL pipeline

### Extract

The Extract stage simply consists in reading the data provided by the hospital in its original format. Some of these datasets are stored in `.xlsx` files while others are stored in `.csv` files. Total data extracted adds up to approximately 1 Gigabyte of memory.

### Transform

During the Transform stage different tasks take place. First of all, data is cleaned and normalized. This includes, among other things: renaming columns so that variables names are consistent across the different datasets, converting columns to the appropriate format, dropping some columns due to irrelevance or emptiness, generating some new columns from the already available columns (for example, by combining date and time columns, datetime variables are created which might be useful for later tasks) and checking for inconsistencies among the data (for example patients that have two different birth dates). The output of this job is temporarily stored in `.parquet` files (binary, faster to read file format).

In the second place, the process of generating patient clinic records occurs. This is achieved by iterating over the list of unique patients and gathering the information from the different datasets related to the patient at stake. To deal with this task, two Python classes were created:

- class `Patient`:

This class is aimed to enclose attributes and methods related to a patient. The class is initialized by providing a valid *patient-id*. Each patient has the property `admission_history` which consists of a dictionary of his admissions accessed through their *admission-id*.

By calling the method `load_patient_data()` and indicating the corresponding

database where data is supposed to be extracted from, this class will gather and store as attributes the patient's data regarding admissions, laboratory studies, images studies, surgeries and sector.

The class also contains a property called `admission_history` which consists of a dictionary of `Admission` class objects indexed by their *admission-id*.

The method `get_historic_records()` is of extreme importance. By calling it, we will be able to generate a `Pandas dataframe` with a row for each day the patient spent at the hospital and all the corresponding data upto that point in time.

- class `Admission`:

This class is aimed to enclose attributes and methods related to a patient's admission. The class is initialized by providing a valid *admission-id* and its corresponding `Patient` class object. Thus, both classes are related to each other in a complementary way.

In a similar way to `Patient` functionality, the method `load_admission_data()` will gather from the `Patient.data` attribute all data regarding a particular patient's admission and store them as attributes.

In a similar way to `get_historic_records()`, the method `get_admission_records()` will generate one record per day the patient was hospitalized during a certain admission, with all the patient's data upto that point in time. This includes processing all the patient's surgeries, laboratory studies, images studies and sectors upto each day the patient was hospitalized during the admission.

This last step is extremely important as it guarantees that the model's data will be *point-in-time correct* and so the model will not be able to see information during training that does not replicate the conditions in which it will be used (i.e. incur in *data leakage*). For example, if a certain patient was hospitalized a given date, and after a week in the hospital he underwent an unscheduled surgery, we cannot use the surgery information to predict patient's discharge during the first week hospitalized as, although we have that information in our laboratory settings, data from the future will not be available when using the model in a real-life scenario.

By taking advantage of these two classes, we have built a process that basically goes as it follows (See Algorithm 1):

**Algorithm 1** Patient records generation

---

```

1: for patient in patients do
2:   patient_records = []
3:   for patient in patient.admissions do
4:     admission_records = admission.get_admission_records()
5:     patient_records.append(admission_records)
6:   end for
7:   patient_records.export()
8: end for

```

---

Here there is an example of how the output of this process, for a certain patient, looks like:

admission_id	patient_id	date	hosp_day_number	labos_count	labos_cumulative	...	surgery
507351-7	1000021-0	2018-05-20	0	17	17	...	
507351-7	1000021-0	2018-05-21	1	5	22	...	
507351-7	1000021-0	2018-05-22	2	0	22	...	
507351-7	1000021-0	2018-05-23	3	0	22	...	
512368-7	1000021-0	2018-07-18	0	13	13	...	NEFRECTOMIA
512368-7	1000021-0	2018-07-19	1	12	25	...	
512368-7	1000021-0	2018-07-20	2	0	25	...	
512368-7	1000021-0	2018-07-21	3	3	28	...	
512368-7	1000021-0	2018-07-22	4	5	33	...	
512368-7	1000021-0	2018-07-23	5	1	34	...	

Figure 2.15: Mock view of a patient's clinic records. Some columns have been collapsed

*In this case, the patient 1000021-0 had two admissions to the hospital. The first of them was in May 2018 (admission id 507351-7) and the following one was in July the very same year (admission id 512368-7). The first time, he was hospitalized for 4 days and the second time, he was hospitalized 6 days, thus his full clinic records consist of 10 rows. The first time, he had 17 laboratory tests requested on the first day of admission and 5 more tests the following day, which gives a cumulative number of laboratory studies of 22 tests during his first admission. During the second admission, he had a surgery on the first day of admission, and then he had laboratory studies performed almost every day until he left the hospital.*

This stage of the ETL processes more than 50,000 patient clinic records which represented around 500,000 rows. Each patient's records is saved as a `.parquet` file named after the corresponding *patient-id*.

## Load

The third and last step of the transform stage is the one in charge of building the dataset that will serve as the main input for the machine learning model. In order to

do this, the clinic records from the different patients generated in the previous process are concatenated and sorted by date. Patient static variables, as for example birth date and gender, are also included in this step. Finally, the target variable is defined: it consists in a boolean column that will take the value 1 (**True**) if the patient was discharged at that day of his admission, 0 (**False**) otherwise. For the example above, the result is shown in Figure 2.16.

admission_ id	patient_id	date	hosp_day_ number	labos_count	labos_ cumulative	...	surgery	gender	birth_date	discharge
507351-7	1000021-0	2018-05-20	0	17	17	...		M	1961-10-05	0
507351-7	1000021-0	2018-05-21	1	5	22	...		M	1961-10-05	0
507351-7	1000021-0	2018-05-22	2	0	22	...		M	1961-10-05	0
507351-7	1000021-0	2018-05-23	3	0	22	...		M	1961-10-05	1
512368-7	1000021-0	2018-07-18	0	13	13	...	NEFRECTOMIA	M	1961-10-05	0
512368-7	1000021-0	2018-07-19	1	12	25	...		M	1961-10-05	0
512368-7	1000021-0	2018-07-20	2	0	25	...		M	1961-10-05	0
512368-7	1000021-0	2018-07-21	3	3	28	...		M	1961-10-05	0
512368-7	1000021-0	2018-07-22	4	5	33	...		M	1961-10-05	0
512368-7	1000021-0	2018-07-23	5	1	34	...		M	1961-10-05	1

Figure 2.16: Mock view of the dataset. Some columns have been collapsed.

Once patients' records are altogether and sorted and their target variable defined, the dataset is splitted in three subsets: one of them will be used for model training, another one will be used for model validation and the last one will remain unused for testing purposes once the final model is chosen. In order to decide how to split the data, three considerations were taken into account:

- The first one has to do with replicating the deployment scenario in our validation schema: the model will learn from past data, but its predictions will be applied to the future. Thus our training-validation split criteria should be temporal rather than randomized. In that way, we make sure that we train the model with previous data than the one we validate the model with.
- Moreover, we considered that patients flow within the hospital is quite fluctuant during the year, with some seasonal components. Because we want our model to learn those patterns it is necessary that each subset dataset includes at least a year of data.
- We only had available two complete years of data, thus the most obvious approach of a year for training, a year for validation and a year for testing was not an option.

As a result, we ended up with a validation schema that combined a temporal split with a randomized one. We took a year-long period for training and the chronologically



consequent year-long period for validation and testing. Within this second year-long subset, we performed a random split for validation and testing with a 50%-50% distribution.

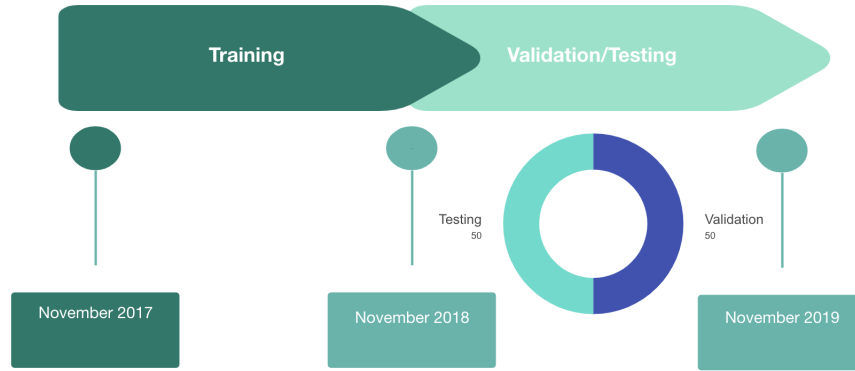


Figure 2.17: Validation schema

## 2.3 Machine Learning Techniques

### 2.3.1 Supervised Learning Algorithms

#### Decision trees

In terms of machine learning, there are different algorithms available to approach a classification problem as the one concerning us. Decision trees constitute a family of supervised-learning algorithms that implement a *divide and conquer* strategy in a non-parametric way by building a hierarchical data structure. Despite their relative simplicity, decision trees are the core of many more complex and sophisticated supervised learning algorithms that remain an industry standard for many classification as well as regression tasks, especially when dealing with tabular, categorical data.

A decision tree can be defined as a composed of internal decision *nodes* and terminal *leaves*. Each decision node  $m$  implements a function  $f_m(x)$  with discrete outcomes labeling the *branches*. Each leaf node has an *output label*, which in the case of classification is the class label and in regression is a numeric value. A leaf node defines a localized region in the input space where instances falling in this region have the same labels (in classification), or very similar numeric outputs (in regression).

Given a certain input, the test is applied at each node, and one of the branches is taken based on the outcome of this evaluation. The process is repeated recursively until arrival at a leaf node. At that point, the value of the leaf constitutes the output value for the input.

Decision trees are a non-parametric method, as they don't assume any parametric form for the classes distributions nor a fixed structure for the tree. Leaves and nodes

are built during the training process depending on the complexity of the problem inherent in the data.

In classification tasks, splits into branches are defined based on their goodness for the learning goal which is quantified by an impurity measure. That is, data is splitted into branches so that the resulting nodes are as pure as possible. A common measure for impurity is *entropy* defined as:

$$I_m = - \sum_{i=1}^K p_m^i \log_2 p_m^i \quad (2.1)$$

where  $p_m^i$  represents the probability of class  $i$  given node  $m$ .

In a binary setting (but this result can be generalized to  $n$  classes) *entropy* is a concave function with minimum at  $p = 0$  and  $p = 1$ , this is, when the node is perfectly pure.

The learning process for decision trees consists of iterating all the nodes which are not yet pure, evaluating each possible split position and choosing the one that has the minimum *entropy* (see Algorithm 2). This is the basis of the *Classification and Regression Tree (CART)* algorithm (Breiman et al. 1984) and its extensions.

Among the main advantages of working with decision trees, lay interpretability and simplicity. Unfortunately, the basic decision tree algorithm, as explained above, is not as powerful as other classification approaches in terms of its predictive accuracy, besides the fact that it can be very non-robust.

In order to address these issues, the Machine Learning community has developed a series of methods known as *ensemble methods* that take advantage of learning decision trees by combining them in a way that increases their performance and decreases their bias.

---

**Algorithm 2** Decision tree algorithm

---

```

function GENERATETREE(X)
2:   if node_entropy(X) <  $\theta$  then
       leaf  $\leftarrow$  create_node()
4:   return leaf
   else
6:     i  $\leftarrow$  FindBestSplit(X)
       for each branch of  $x_i$  do
8:         find  $X_i$  samples falling in branch
           GenerateTree( $X_i$ )
10:    end for
       end if
12: end function

   function FINDBESTSPLIT(X)
14:   MinEnt  $\leftarrow$  Max
       for attribute in X.attributes do
16:     for each possible split do
           split  $\leftarrow$  GenerateSplit()
18:     e  $\leftarrow$  split.entropy()
           if e < MinEnt then
20:             MinEnt  $\leftarrow$  e
               BestSplit  $\leftarrow$  split
22:           end if
       end for
24:   end for
       return BestSplit
26: end function

```

---

**Ensemble methods**

Ensemble methods are conceived as learning algorithms that construct a set of classifiers and then classify new data points by taking a weighted vote of their predictions (Dietterich, 2000). The main discovery is that ensembles are often much more accurate than the individual classifiers.

In his article, *Ensemble Methods in Machine Learning*, Dietterich T. emphasizes the three main reasons why this is the case:

The first of them is statistical, he explains. A learning algorithm can be viewed as searching a space  $H$  of hypotheses to identify the best hypothesis in the space. The statistical problem arises when the amount of training data available is too small compared to the size of the hypothesis space. Without sufficient data, the learning algorithm can find many different hypotheses in  $H$  that all give the same accuracy on the training data. By constructing an ensemble out of all of these accurate classifiers, the algorithm can average their votes and reduce the risk of choosing the wrong clas-

sifier.

The second reason is computational. Many learning algorithms work by performing some form of local search that may get stuck in local optima. In the particular case that concerns us, decision tree algorithms employ a greedy splitting rule to grow the decision tree: they evaluate splits one at a time and decide which split to choose independently on the consequent possible splits. Notice that, actually, optimal training of a decision tree is NP-hard (Hyafil and Rivest, 1976; Blum and Rivest, 1988). Thus an ensemble constructed by running the local search from many different starting points may provide a better approximation to the true unknown function than any of the individual classifiers.

The last issue is representational. In many machine learning applications, the true function  $f$  cannot be represented by any of the hypotheses in  $H$ . Even with very flexible algorithms like decision trees, as the training sample is finite, they will explore a finite set of hypotheses and stop searching when they find one that fits the training data. Forming weighted sums of hypotheses from  $H$  can actually expand the space of representable functions.

In terms of how the ensemble is constructed, different techniques have been developed. Some of them work particularly well with decision trees classifiers, more specifically: *bagging* and *boosting*.

- Bagging:

Bagging is an ensemble technique that takes advantage of *bootstrapping* to improve statistical learning methods such as decision trees by reducing its variance. We recall from basic statistics theory that, given a set of independent observations  $Z_i, Z_1, \dots, Z_n$ , each with variance  $\sigma^2$ :

$$\text{Var}(\bar{Z}) = \sum_{i=1}^N \sigma_i^2 = \sigma^2/N \quad (2.2)$$

Thus a natural way of reducing variance (and hence increasing the prediction accuracy of a statistical learning method) is to take many independent training sets from the population, building a classifier and then averaging their predictions. An approximation to this idea is to generate bootstrapped training datasets, train a classifier with each bootstrapped training set and then averaging their predictions, which is exactly what bagging does:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x) \quad (2.3)$$

An improvement over the plain bagging technique is the one applied by the well known *Random Forest* algorithm. To prevent the different bagged trees from

looking quite similar in presence of a very strong predictor, *Random Forest* algorithm forces each classifier to consider only a random subset of the predictors, thus decorrelating the trees. This approach has proved particularly efficient in preventing *overfitting* as well as speeding up training. Besides this, as trees are trained independently from each other, parallel processing can be applied accelerating the training process even more.

*Random Forest* is one of the learning algorithms that we have experimented with in our model. The most important hyperparameters in its configuration are:

- $B$ : the number of classifiers (learning trees) to fit. In bagging, large number of classifiers usually do not derive in overfitting, thus the optimal choice of  $B$  will rely upon the trade off between increasing performance and fastening training.
  - $m$ : the sample number of the features to use when building each tree. The smaller the number of features used, the less correlated the different classifiers will be. However, this will sacrifice some performance as well. A balance should be found with regard to this parameter value.
- Boosting:

A quite different approach to improve performance of decision trees by taking advantage of ensembles is boosting. Like bagging, boosting is a general approach that can be applied to many statistical learning methods. Here we restrict our discussion of boosting in the context of decision trees.

Instead of training trees independently from each other, boosting proposes a sequential training of the different classifiers: each tree is grown using information from previously grown trees. The idea behind this procedure is to prevent fitting a large decision tree which is likely to overfit the training data. Instead, the boosting algorithm learns slowly by building relatively small trees which aim to predict the residuals of the previous classifier rather than the outcome  $Y$  (See Algorithm 3), thus it focuses on improving the decision function  $\hat{f}$  in those areas where it is not performing well.

This slow learning process has demonstrated to excel in terms of performance, dominating most *Kaggle*'s and other data science competitions.

The main hyperparameters in the boosting algorithm are:

- $B$ : the number of classifiers trained, which, unlike what happens with Random Forest, can derive in overfitting if it is too large.
- $\lambda$ , the shrinkage parameter or learning rate, which controls the speed at which boosting learns. Very small values of  $\lambda$  will slow the training and also require a larger number of classifiers to achieve good performance.

- $d$  which constitutes a regularization parameter controlling the maximum number of splits in a single tree (i.e. the depth of the tree).

---

**Algorithm 3** Boosting Algorithm

---

```

function BOOSTDECISIONTREES( $X, B, \lambda, d$ )
     $\hat{f}(x) \leftarrow 0$ 
3:   for all  $i$  in  $X$  do:
         $r_i \leftarrow y_i$ 
    end for
6:   for  $b$  in  $1, 2, 3, \dots, B$  do
         $\hat{f}^b \leftarrow \text{GenerateTree}(X, d)$ 
         $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$ 
9:    $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$ 
    end for
    return  $\hat{f}(x)$ 
12: end function

```

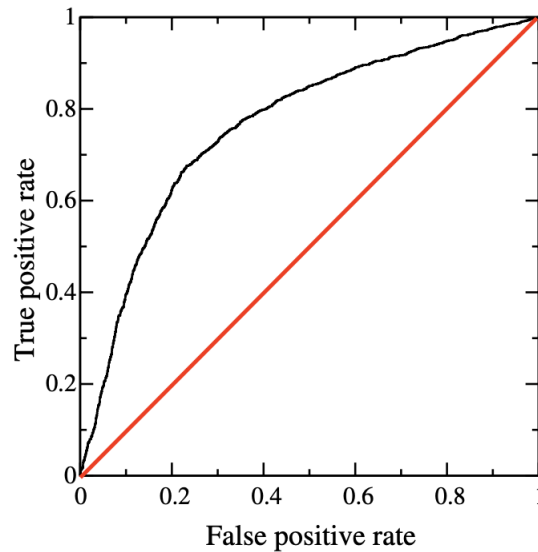
---

### 2.3.2 Machine Learning Model Evaluation

#### Performance Metric

Our metric of choice for the assessment of the different machine learning models experimented in this project has been *Area under the ROC curve*. Area under the ROC curve (*AUC-ROC*) is a widely known performance metric in the Machine Learning community that has been gaining more and more adherence in recent years because of its properties that make it especially useful for domains with skewed class distribution or unequal classification costs.

As its name indicates, this metric is obtained by calculating the area under the *Receiver Operating Characteristics* (ROC) curve. The Receiver Operating Characteristics curve is a graphical plot that illustrates the potential of a binary classifier as its discriminant threshold varies. The curve is obtained by plotting the different combinations of *TPR* (true positive rate) on the  $y$ -axis, and *FPR* (false positive rate) on the  $x$ -axis, derived from using a certain discriminant threshold. By iterating over the threshold space  $[0, 1]$  and plotting its results, a curve can be drawn. The closer the curve is to the diagonal, the lower the degree of separability that the classifier provides. At the extreme, a random classifier's ROC curve would actually be the diagonal where  $TPR = FPR$ ).

Figure 2.18: Example of *ROC* Curve

While interesting in graphical terms, to compare classifiers it is necessary to reduce performance measure to a single scalar value and that's where *AUC-ROC* comes to the scene (Fawcett, 2005). Since it's a portion of an area of a unit square it lies between 0 and 1: the larger its value, the better the classifier. The *AUC-ROC* has an important statistical property: its value is equivalent to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance.

In the paper *The use of the area under the ROC curve in the evaluation of machine learning algorithms*, Bradley (1996) presented the results of a series of experiments analyzing the performance of six different machine learning algorithms using both *AUC-ROC* and the more conventional *accuracy* metric.

*AUC-ROC* demonstrated to have increased sensitivity in Analysis of Variance test, a standard error that decreased as both *AUC-ROC* and the number of test samples increased, independency from decision threshold and invariant to a priori class probabilities.

In the context of our patient discharge prediction task, we are presented with a highly imbalanced dataset (See Figure 2.20) where positive cases (discharges) account for only 13% of the data-points. This is due to the fact that most patients stay more than 2 days hospitalized, as the density distribution of admission length illustrates in Figure 2.19. This characteristic of the data requires the use of a classification performance metric robust to class imbalance, such as *AUC-ROC*.

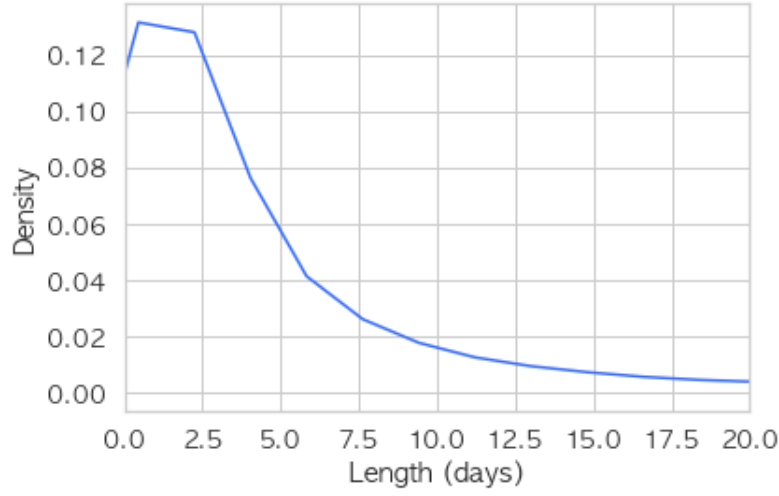


Figure 2.19: Empiric distribution of Hospitalization length

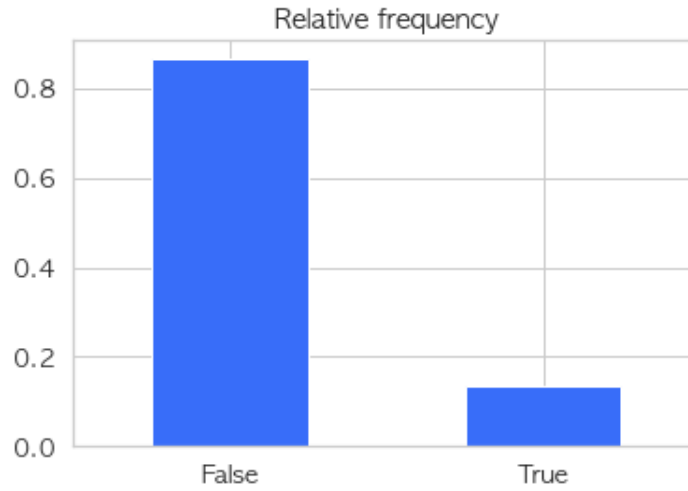


Figure 2.20: Empiric distribution of class probabilities

### Hyperparameters Tuning: a Bayesian Optimization approach

Hyperparameters are important for machine learning algorithms since they directly control the behaviors of training algorithms and have a significant effect on the performance of machine learning models (Wu et al, 2019, pp.26).

The aim of hyperparameter optimization in machine learning is to find the hyperparameters of a given machine learning algorithm that return the best performance as measured on a validation set (Koehrsen, 2018). In analytical terms, this problem can be represented by the following equation:

$$x^* = \arg \min_{x \in \mathcal{X}} f(x) \quad (2.4)$$

where  $f$  represent the objective score to minimize (in our case, 1- *AUC ROC* score) evaluated on the validation set and  $x^*$  the set of hyperparameters values that mini-



mizes this function. The process of solving this problem is known as *fine tuning* and it constitutes an NP-hard optimization problem itself.

There are many approaches to the fine tuning process. Some of the most extended ones make use of a *brute-force* search (grid search) or a *random search* from a hyperparameter subspace in order to find which is the combination of hyperparameter values that minimizes  $f$  within that particular subspace. The main limitation with this type of methods is that they are highly inefficient, as they spend a significant amount of training time evaluating irrelevant combinations of hyperparameters. Moreover, they are constrained to search into the subspace pre-defined by the user, whose choice is not at all trivial.

As a consequence, there is great appeal for automatic approaches to fine tuning that could optimize the performance of any given learning algorithm to the problem at hand (Snoek et al, 2012). Among this type of methods (known as *autoML*) lies the Bayesian Optimization approach to hyperparameter tuning that has been applied in the current project.

In general terms, Bayesian optimization typically works by assuming the unknown function  $f$  was sampled from a Gaussian process and maintains a posterior distribution for this function as observations are made. In the case of hyperparameter tuning, the observations are the results of running learning algorithm experiments with different hyperparameters. To pick the hyperparameters of the next experiment, this method will optimize the expected improvement (EI) or the Gaussian process upper confidence bound (Snoek et al, 2012).

In other words, the method works by building an estimation of the probability function of the score given certain hyperparameters values. Then, it analytically optimizes this function and finds the combination of hyperparameters that minimizes the score surrogate probability, it applies these hyperparameters to the true objective function (by running the experiment) and it updates the probability estimated function incorporating these last results. This process is repeated until certain stopping criteria is met.

By doing this, Bayesian optimization methods are able take advantage of the knowledge derived from the previous experiments in order to make an informed decision on the following combinations of hyperparameters to experiment with. By evaluating hyperparameters values that appear more promising from past results, these methods can find better model settings than random or grid search in fewer iterations.

## Cross Validation

Overfitting is one of the main challenges of Machine Learning. Because inductive bias is at the core of any learning application, one should keep always an eye on the model ability to generalize their results.

*Cross-validation* is the process of measuring the generalization ability of different models by testing them on new data, unseen during training, and then choosing the most accurate one on this set of data. There are different techniques available to perform cross-validation. In the current project we have made use of two of them in a complementary way: the *validation-set* approach and *k-fold* cross-validation.

- The *validation-set* approach

This is a very simple strategy that involves dividing the available set of observations into two parts: a training set and a validation/hold-out set. The model is fitted using only the training set, and the trained model is used to predict responses on the *validation-set*. The main drawback regarding using these approach has to do with the fact that only a subset of the observations are used to train the model (less data will mean the model learning will be poorer) and the performance is measured only on a subset of the observations as well (thus, the measured error can more variable, as it's calculated with less data). However, these drawbacks fall behind when the available data is sufficiently large.

As in our problem data scarcity is not an issue (because we have at hand more than two-years daily patient data), the *validation-set* approach seems a reasonable technique to be applied. Cross validation using a *validation-set* has been used to compare the performance of the different learning algorithms in our experiments.

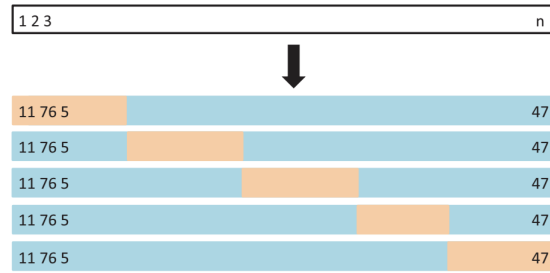


Figure 2.21: Cross validation: Train-Test split

- *K-fold* cross validation approach

This approach involves dividing randomly the set of observations into  $k$  groups of the same size. The model is fitted  $k$  times, each of them using  $k - 1$  folds for training and the remaining left-out one for validation, while rotating the one fold that is left out. By the end of the process there are  $k$  different estimates of the model performance which are averaged to estimate the *k-fold* cross validation performance.

We will use this approach to compare performance among different combinations of hyperparameters for the same machine learning algorithm. There is one main reason why this makes sense: when optimizing hyperparameters, a significant number of models should be fitted. If all of them are trained and validated with the same static train-validation split we might end up overfitting to the validation data.

Figure 2.22:  $k$  fold cross validation

By combining this set of strategies, we aimed to be able to capture the best of each of them. On the one hand, having a validation, large-enough set hold out from training would enable us to get a sufficiently robust measure of models' generalization ability to compare among different algorithms' performance. On the other hand,  $k$ -fold cross validation would prevent us from overfitting the validation data when fitting a large number of models during the hyperparameter tuning process.

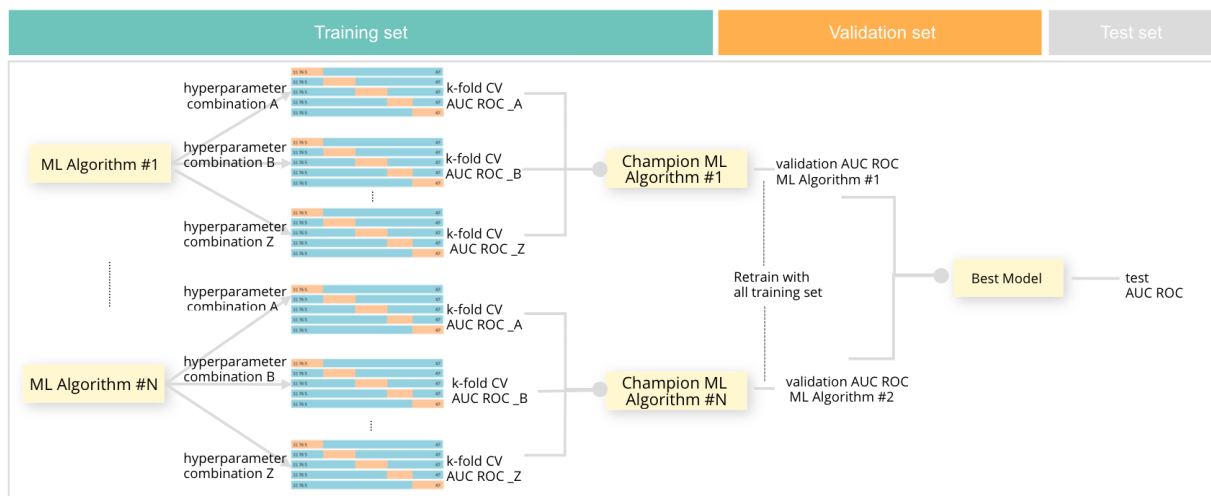


Figure 2.23: Cross validation strategy

### 2.3.3 Feature Engineering Techniques

Feature engineering is an important but labor-intensive component of machine learning applications (Bengio et al, 2013). Most machine learning performance is heavily dependent on the representation of the feature vector. As a result, much of the actual effort in deploying machine learning algorithms goes into the design of preprocessing pipelines and data transformations (Bengio et al, 2013). To make use of feature engineering a model's feature vector is expanded by adding new features that are calculations (ratios, differences, logs) based on the other features or transformations of them (Heaton, 2017).

In the current project, different techniques have been applied with the objective of getting the most out of the available data to predict discharge probability.

- Categorical features encoding

As anticipated in the exploratory analysis, the data at hand contains several categorical variables, some of them with a really high cardinality. As categories of a categorical variable are usually not numeric, it is necessary to encode them in a certain way so that the machine learning algorithm can take advantage of them.

One of the most simple and extended approaches, known as *One-hot Encoding*, is to generate one binary variable for each category within a categorical feature. Thus a categorical variable with  $k$  possible categories is encoded as a feature vector of length  $k$  (Zheng and Casari, 2018). One advantage of this method, besides its simplicity, is that it deals very well with missing values within the categorical feature.

As described in Section 2.2, the dataset in consideration has some features with structural missing values, thus encoding the categorical features like this is a way of assessing this issue while not losing the information derived from the very same fact that there is no value for that feature for a certain observation. We have implemented this method applying the `OneHotEncoder` from `ScikitLearn`'s preprocessing module.

The downside of *One-hot Encoding* is the large and sparse output matrix that this encoder generates when the number of categories is large. However the machine learning algorithm used for training our models deals very well with this huge data space, mainly because it applies a technique called *Exclusive Feature Bundling* that will be explained in detail manner in Section 2.4.

- Text embeddings

In our dataset, we were faced with some text-features related to the patient's diagnosis, types of interventions, etc. In order to be able to add these features to a machine learning model, some transformation is needed. There are different techniques available to deal with this, our choice has been *tf-idf* transformation.

*Tf-idf* stands for *term frequency - inverse document frequency* and it is a metric that represents how frequent is a certain token within a document with respect to the frequency of the very same token across the whole corpus. More specifically, *tf-idf* calculation goes as it follows:

$$\begin{aligned} \text{bow}(w, d) &= \# \text{ times word } w \text{ appears in document } d \\ \text{Tf-idf}(w, d) &= \text{bow}(w, d) * N / (\# \text{ documents in which word } w \text{ appears}) \end{aligned}$$

where  $N$  is the total number of documents in the corpus (dataset).

The advantage of *tf-idf* transformation in comparison with other more simplistic approaches to text encoding, is that it provides a better representation of a particular token's importance, as it not only considers the frequency of a term but also its relationship with a certain document relative to the whole corpus.

In order to be able to generate this type of representation, some preprocessing needed to be made to the text features in our dataset, in particular *tokenization* and *counting*. We have implemented these by making use of `Scikit Learn`'s `Tf-IdfTransformer` which performs both, the preprocessing and the transformation tasks, together.

However, certain adjustments were made to customize this feature processor for the particular problem at stake. More specifically, a customized list of *stopwords* was generated to this aim. Besides, *bigrams* and *trigrams* were included in the *tokenization* process, as some medical terms might consist of more than one word, or some words must be considered within its context to get its full meaning.

- Missing values

As explained in Section 2.2, some features of the dataset at hand present missing values. In some cases, they were directly filtered out from the model because of the little information they contained. However, some others were features with a strong predictive potential, as far as our exploratory analysis suggested. While the proportion of missing values was not something to worry about, the machine learning algorithms we are working with do not accept missing values, thus some preprocessing was needed.

In the case of categorical features this issue was already addressed by the *One-hot-encoding* approach. As for numerical features, we have followed an *imputation* strategy and filled out with zeros the missing-values spaces. We have done this by applying the `SimpleImputer` processor from the `sklearn.Impute` module.

## 2.4 Software

### Project Source Code

This project has been developed entirely using Python Programming Language. Custom software has been developed to approach problem specific tasks and features. This code is available at the following *Git* repository under the module named `thesis_lib`:

[github.com/josedallavia/A-Machine-Learning-Approach-for-Prediction-of-Hospital-Bed-Availability](https://github.com/josedallavia/A-Machine-Learning-Approach-for-Prediction-of-Hospital-Bed-Availability)

## Algorithms Implementations

In terms of the ensemble methods implementations, our algorithm of choice has been *LightGBM* (in its Python's library version). There several advantages of this implementation of the aboved explained tree-based learning algorithms, in particular: faster training speed and higher efficiency, lower memory usage, better accuracy, support for parallel computing and its capability in handling large-scale data.

*LightGBM* innovative approach relies on the techniques used for reducing the number of data instances and the number of features to speed up the training process of traditional gradient boosting in a smart way so that performance does not seem affected. To this aim, the algorithm relies on the application of both *Gradient Based One-Side Sampling (GOSS)* and *Exclusive Feature Bundling (EFB)*. The first technique (*GOSS*) excludes a significant proportion of data instances with small gradients and only uses the rest of them to estimate the information gain of the possible splits, thus obtaining quite accurate estimations with a much smaller data size. The second one (*EFB*), bundles mutually exclusive features using a greedy but effective approach and thus reduces the dimension of the data space. This results especially useful for datasets with high cardinality categorical features, typically approached with *One-hot-encoding*, generating huge sparse matrices.

By applying these variants with respect to the conventional gradient boosting decision tree, *LightGBM* has demonstrated to speed up the training process over twenty times while achieving almost the same accuracy (Ke et al, 2017, p. 3146).

## Other Python *open-source* libraries

We have also made extensive use of some *open-source* Python libraries for Machine Learning and data related tasks:

- *Matplotlib*
- *Numpy*
- *Pandas*
- *Hyperopt*

## Other software

Furthermore, additional *open-source* software has served its purpose in this project:

- *Jupyter Notebooks* for exploration and prototyping.
- *Git* for version control.
- *Apache Parquet* for data storage.
- *PyCharm* for coding.

# Chapter 3

## Results

### 3.1 Model Experiments

In order to predict patient discharges, we have experimented with different approaches to the data and supervised learning algorithms. Results of these experiments are presented in Table 3.1.

In order to understand which features were the most important in each modelling approach, feature importance barcharts were plotted. Feature importance is calculated by the *LightGBM* package based on how many times each feature was used to branch the different decision trees within the ensemble.

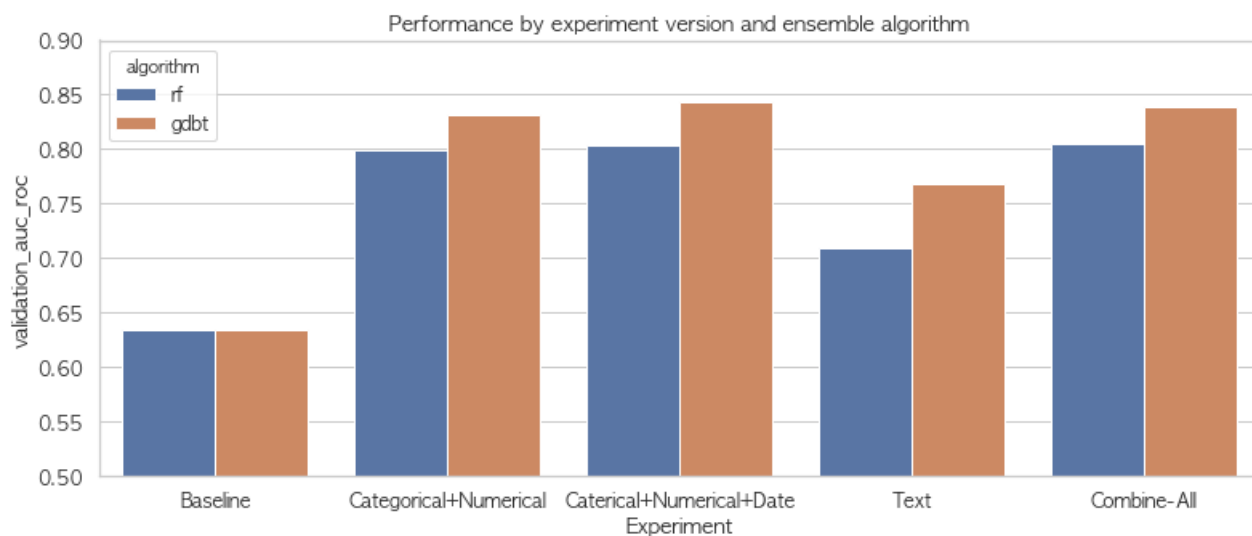


Figure 3.1: Experiments performance

Table 3.1: Modelling experiments results

Experiment set	Model description	Algorithm	No. of features	Training time (secs)	AUC-ROC Training	AUC-ROC Validation	Overfitting
1	baseline	gradient boosting	2	15.2	0.6482	0.6344	2.1%
1	baseline	random forest	2	19.1	0.6456	0.6346	1.7%
2	categorical + numerical features	random forest	59	44.4	0.8062	0.7987	0.9%
2	categorical + numerical features	gradient boosting	59	38.8	0.8484	0.8309	2.1%
3	date features only	gradient boosting	4	13.7	0.6286	0.6131	2.5%
3	date + categorical + numerical features	gradient boosting	63	37.0	0.8591	0.8432	1.8%
3	date + categorical + numerical features	random forest	63	52.3	0.8111	0.8043	0.8%
4	text only features	gradient boosting	4	28.7	0.7580	0.7201	5.0%
4	text + sequence features	gradient boosting	11	45.5	0.7984	0.7678	3.8%
4	text + sequence features	random forest	11	52.9	0.7266	0.7102	2.3%
5	combine all features	gradient boosting	59	94.7	0.8569	0.8392	2.1%
5	combine all features	random forest	59	87.4	0.8111	0.8048	0.8%

Default hyperparameters configurations was:

- For *RandomForest* : { num\_iterations : 100, max\_depth : -1, learning\_rate : 0.1, bagging\_fraction : 0.7, bagging\_freq : 1 }
- For *GradientBoosting*: { num\_iterations : 100, learning\_rate : 0.1, max\_depth : -1, subsample: 1.0 }



Modelling experiments could be summarized into different groups.

- Experiments set #1: Baseline

These were models trained with very little data (more specifically only a categorical variable: *patient\_gender*, and a numerical variable: *patient\_age*). This approach was applied to both *RandomForest* and *GradientBoosting* algorithms. The objective of these models was not to produce an accurate classifier but rather to validate the pipeline built and, simultaneously, to establish a lower bound for the performance of the following, more complex models to be evaluated against.

Performance achieved by these experiments was 0.6344 and 0.6346 *AUC-ROC* score on validation data for *GradientBoosting* and *RandomForest* classifiers. Training time was 15.2 secs for *GradientBoosting* and 19.1 secs for *RandomForest*. Overfitting, measured relatively to the training set *AUC-ROC* score was 2.1% and 1.7% respectively.

- Experiments set #2: Categorical-&-Numerical-features-only models

This set of models included all numerical features available in our dataset with a scaling transformation and all the categorical features with a *One-hot encoding* representation. This made up to more than 28 K features (due to the high cardinality of the categorical features). Both ensemble classifiers *RandomForest* and *GradientBoosting* were tested with this data.

*RandomForest* achieved a performance of 0.7987 *AUC-ROC* score on validation set, with a relative overfitting of less than 1%. Training time was 44.4 secs. *GradientBoosting* hit an *AUC-ROC* score on validation of 0.8309 by incurring in 2.1% overfitting. Fitting time for this classifier was 38.8 secs.

For both models, among the top features we found *hospital\_day\_number* (this is, how many days the patient has been hospitalized so far) and *patient\_age*. Other important features had to do with the cumulative number of different services (studies, surgeries, images) the patient had received so far.

- Experiments set #3: Include-date-features models.

This set of models considered some date-related features, not included in the original dataset but extracted from some of its variables. In particular for categorical features we built: *date\_weekday* and *date\_month* were extracted from the variable *date* (which represented the date a particular data point referred to), *admission\_weekday* and *admission\_month* were extracted from the variable *admission\_date* (which represented the date in which the patient was admitted to the hospital). Different experiments were run with this set of extracted features, either including or not the previous categorical and numerical features.

The classifier fitted only with the date-related features achieved a performance of 0.6131 *AUC-ROC* score on validation data with a 2.5% overfitting. The ob-

jective of this model was to assess whether or not the date features themselves had certain predictive value.

When added to the previous set of categorical and numerical features performance was 0.8432 *AUC-ROC* score for *GradientBoosting* algorithm and 0.8043 for *RandomForest*. Training time was 37 and 52.3 secs respectively, and the number of features ascended to 63.

Among the features of these models *hospital\_day\_number* and *patient\_age* remained of critical importance. The date feature *date\_weekday\_5* showed up within the top fifteen features for both classifiers, reflecting that whether or not it was Friday (i.e weekday # 5) was important to predict patient discharge.

- Experiments set #4: Include text-features.

The fourth set of experiments dealt with the introduction of some text-features.

In the dataset there were some features that were explicitly text, as for example the variables related with the patient's uncodified diagnosis during different moments of their admission (such as *presumptive\_diagnosis* or *administrative\_diagnosis*).

Furthermore, there were variables whose values were sequences of tags (i.e. sequences of categories). For example: *labos\_studies\_names* consisted of a list of the different laboratory studies that were requested to the patient up to that day. We decided to treat this and other similar variables as text, using *tf-idf* with *bigrams* or *trigrams* to generate their embeddings. Thus, if there was some pattern in the data regarding some particular sequence being an indicator of the upcoming discharge or not discharge, we would be able to exploit it.

By doing this, performance reached an *AUC-ROC* score of 0.7201 on validation data using only four features related to the patient diagnosis. The classifier took 28.7 secs to be fitted and fell into 5% of overfitting relative to training performance.

On the second iteration of models with text-features, where sequences of categories features were treated as text, performance was even more promising, reaching a 0.7678 *AUC-ROC* score on validation with *GradientBoosting* ensemble. This training took 45.5 secs and overfitting was reduced to 3.8%. *RandomForest* instead reached a 0.7102 validation *AUC-ROC* score and spent 52.9 secs on training. Overfitting in this case was 2.3%.

- Experiments set #5: Combine-all features.

In experiment set #5 the different features coming from the above described experiments were combined into a single model. This final model included *one-hot-encoded* categorical variables, numerical variables, date-related variables like day of the week and month of the year that let us incorporate some seasonal/cyclical components to the model and text-features regarding both uncoded diagnosis and sequences of studies names, surgeries types, etc. These made up to a total of 59 features.

Performance achieved was 0.8392 and 0.8048 *AUC-ROC* score on validation set for *GradientBoosting* and *RandomForest* ensembles respectively. *GradientBoosting* took 94.7 secs for training while *RandomForest* did it in 87.4. Overfitting was 2.1% in the first case and, 0.8% in the second one.

In both models, among the top features, we found variables coming from the different above listed experiments. *Hospital\_day\_number* and *patient\_age* remained being the most important ones, just like in the very first iterations, but some date features (*date\_weekday\_5* specifically) and some text-features, for example tokens like *eme* (referring to emergencies) or *cesarea* (caesarean operation in Spanish) demonstrated to have contributed significantly to the model performance as well.

## 3.2 Model Selection

After choosing the most promising models from the experiments reported in the previous section, hyperparameters were optimized for both ensemble types, using the Bayesian Optimization approach described in Section 2.3. Table 3.2 summarizes the results of the hyperparameter fine tuning process.

Table 3.2: Model Selection: Comparative results

Metric	Configuration	Gradient boosting	Random forest
<b><u>Hyperparameters</u></b>			
	Default	{num_iterations : 100, learning_rate : 0.1, max_depth : -1, subsample: 1.0 }	{num_iterations : 100, max_depth : -1, learning_rate : 0.1, bagging_fraction : 0.7, bagging_freq : 1}
	Optimized	{num_iterations : 500, learning_rate : 0.06, max_depth : 20, subsample : 0.71}	{num_iterations : 300, max_depth : 6, learning_rate : 0.015, bagging_fraction : 0.5, bagging_freq : 2}
		{num_iterations : +400, learning_rate: +0.5, max_depth: enabled +20, subsample: -0.29}	{num_iterations : +200, learning_rate: -0.085, max_depth: enabled +6, bagging_fraction: -0.2, bagging_freq: +1}
<b><u>Training time (sec)</u></b>			
	Default	94.7	87.4
	Optimized	144.9	132.0
		50.2	44.6
<b><u>AUC-ROC validation</u></b>			
	Default	0.839	0.805
	Optimized	0.844	0.804
	(%)	0.61%	-0.08%
<b><u>Overfitting (%)</u></b>			
	Default	2.1%	0.8%
	Optimized	4.3%	0.6%
	(pp)	2.22	-0.14

***RandomForest***

In the case of *RandomForest* algorithm, the hyperparameters optimized were:

- *Bagging fraction*
- *Learning rate*
- *Number of trees*
- *Maximum tree depth*
- *Bagging frequency*

A relevant space of hyperparameters was generated and Bayesian optimization was applied, fitting a total of 200 classifiers. The charts below illustrate the relationship among the different tested values of each hyperparameter tuned and the obtained cross-validation results.

The parameter *bagging fraction* exhibited a negative relationship with the cross validation loss. Most of the iterations were with values between 0.6 and 0.7.

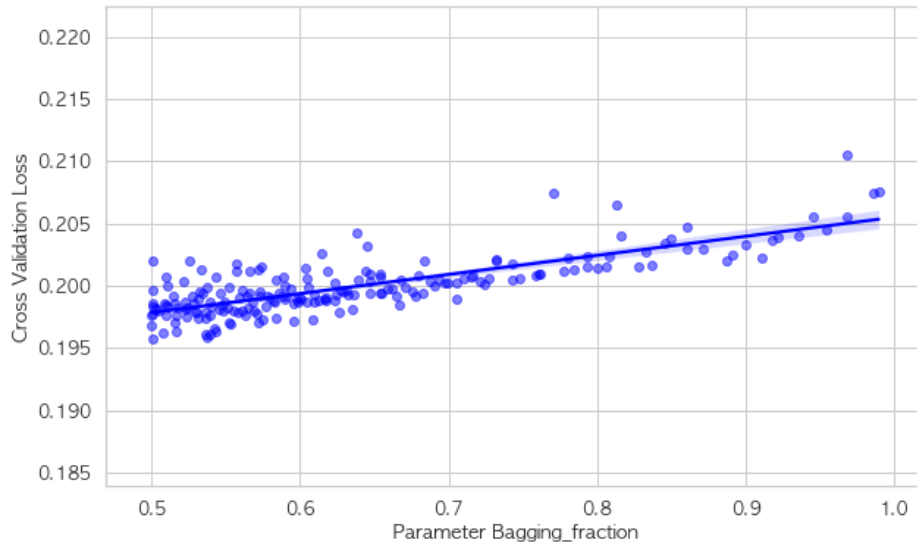


Figure 3.2: Hyperparameters optimization results for *RandomForest* - *bagging fraction* parameter

In case of the *learning rate* parameter, it did not have a great impact on the model performance as can be seen in the Figure 3.3.

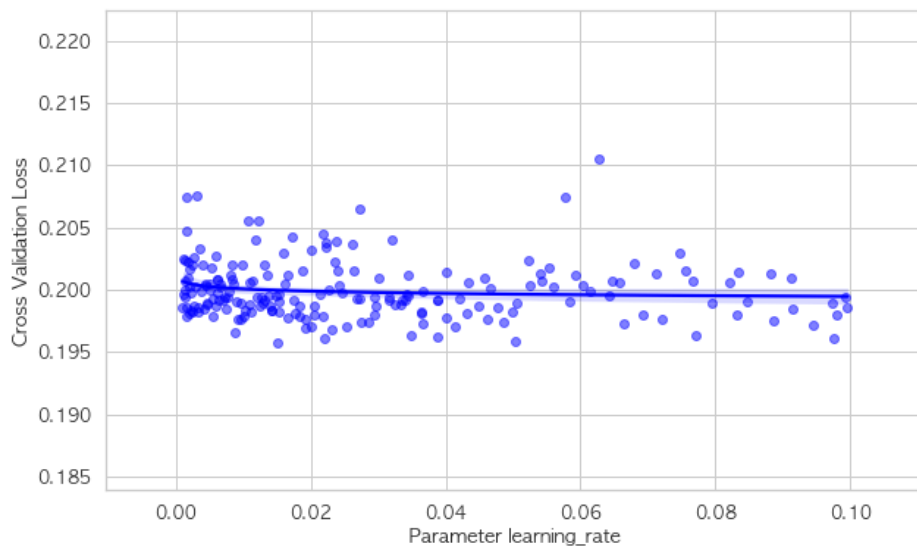


Figure 3.3: Hyperparameters optimization results for *RandomForest* - *learning rate* parameter

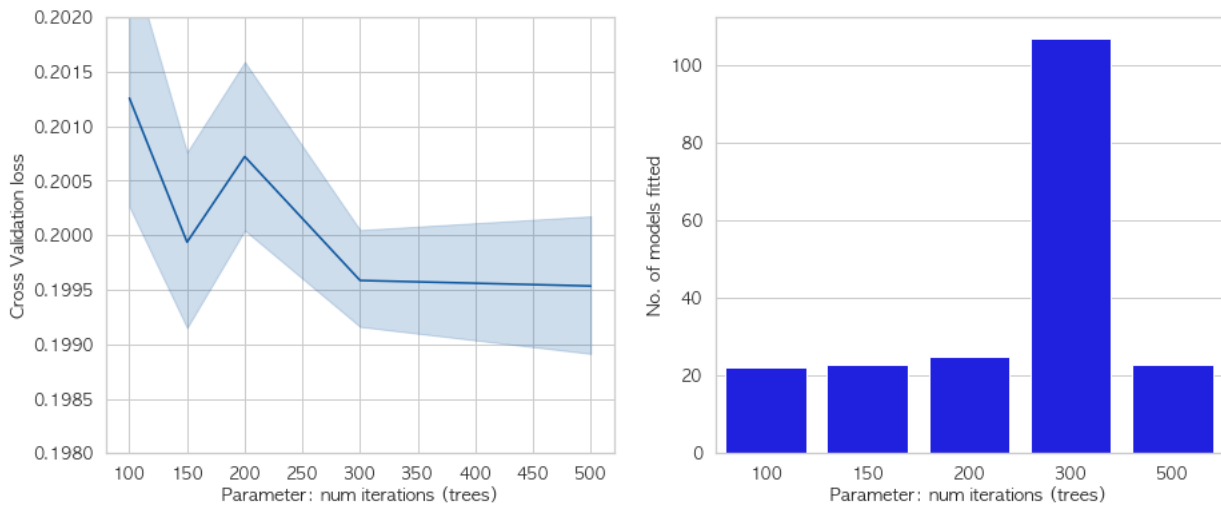


Figure 3.4: Hyperparameters optimization results for *RandomForest* - *number of trees* parameter

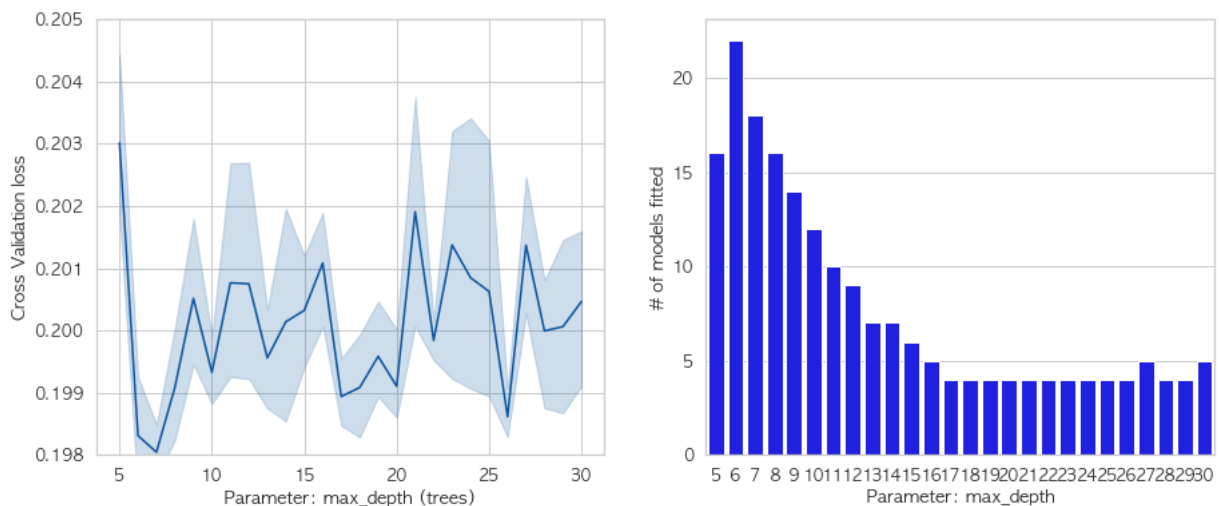


Figure 3.5: Hyperparameters optimization results for *RandomForest* - *maximum depth* parameter

With regard to the *number of trees* parameter, the decrease in the cross validation loss as the *number of trees* increased was notorious, but it turned asymptotic after the parameter value exceeded the 300 trees. The Bayesian Optimization algorithm seems to have noticed this. The right chart in Figure 3.4 shows how 50% of the models fitted by the optimizer (100 out of 200 total fits) were configured with the parameter fixed on 300 trees.

A similar pattern was observed with regard to the *maximum depth* of the *RandomForest* classifier trees but in the opposite direction. Despite the noise, in the left chart of Figure 3.5, it can be seen how the validation loss increases as trees go deeper. Again, this is aligned with the behaviour exhibited by the Bayesian hyperparameter

optimizer. The chart on the right of Figure 3.5 reflects how the lower values of this hyperparameter were the most chosen by the optimizer along its iterations.

The *bagging frequency* parameter exhibited a positive relationship with the cross-validation loss, meaning lower values of the parameter derived in better model performance.

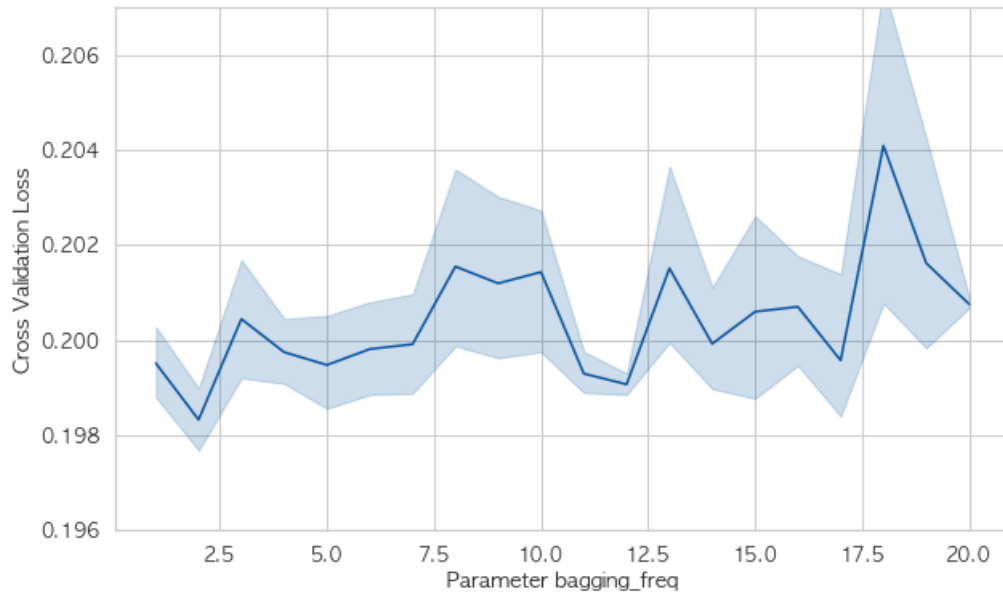


Figure 3.6: Hyperparameters optimization results for *RandomForest* - *bagging frequency* parameter

The best combination of hyperparameters for *RandomForest* model found was:

- *Number of trees*: 300
- *Maximum tree depth*: 6
- *Learning rate*: 0.015
- *Bagging fraction*: 0.5
- *Bagging frequency*: 2

### ***GradientBoosting***

For *GradientBoosting* classification algorithm, the hyperparameters optimized were:

- *Learning rate*
- *Subsample*
- *Number of trees*
- *Maximum tree depth*

In this case, the parameter *learning rate* demonstrated to have a clear impact on the cross validation loss. More specifically, a diminishing asymptotic relationship between the cross validation loss and the *learning rate* was observed, as illustrated by Figure 3.7.

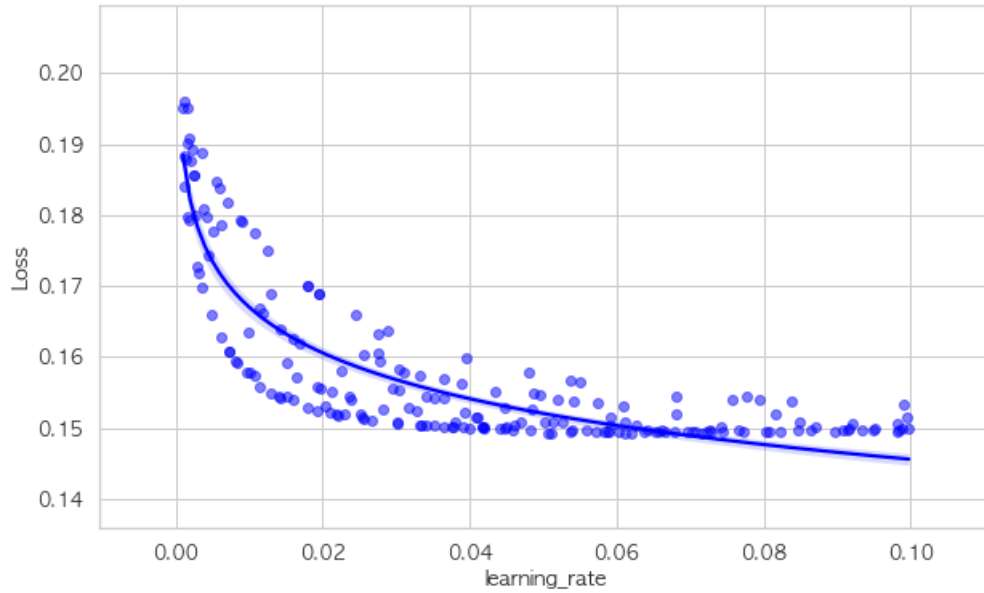


Figure 3.7: Hyperparameters optimization results for *GradientBoosting* - *learning rate* parameter

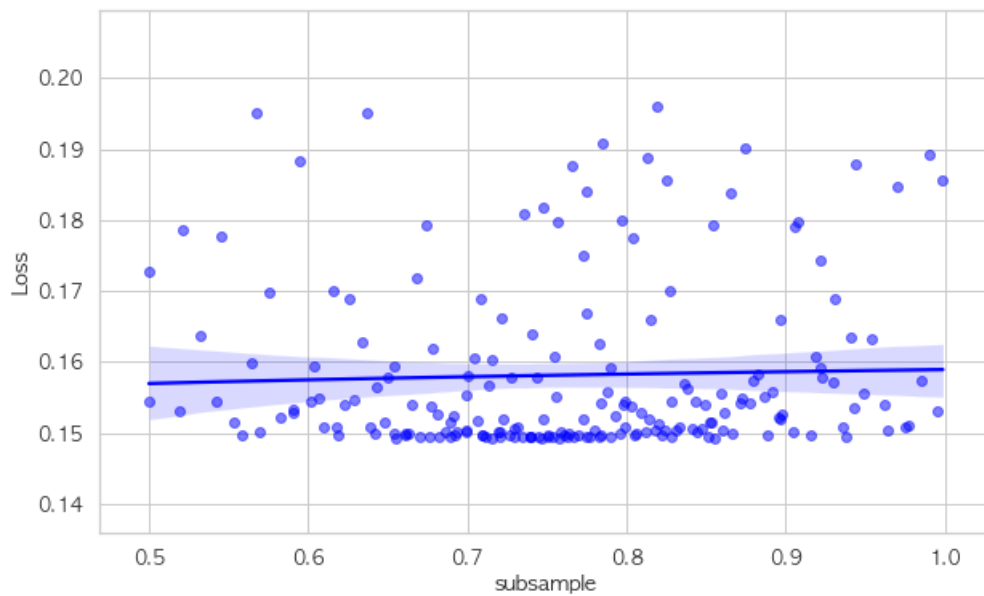


Figure 3.8: Hyperparameters optimization results for *GradientBoosting* - *subsample* parameter



With regards to the *subsample* parameter, there is no clear pattern between the cross validation performance and this hyperparameter's value.

In case of the *number of trees*, the parameter presented a negative relationship with the cross validation loss. This is, the larger the *number of trees*, the better the classifier cross validation performance, as far as these results illustrate. Once again, the hyperparameter optimizer behaviour was aligned with this finding, as it stands out in Figure 3.9.

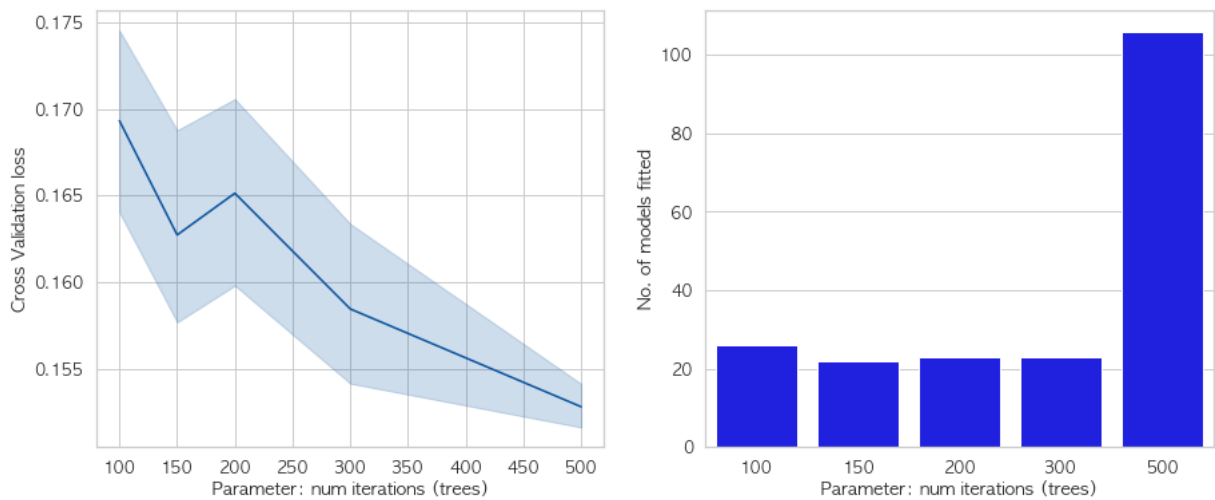


Figure 3.9: Hyperparameters optimization results for *GradientBoosting* - *number of trees* parameter

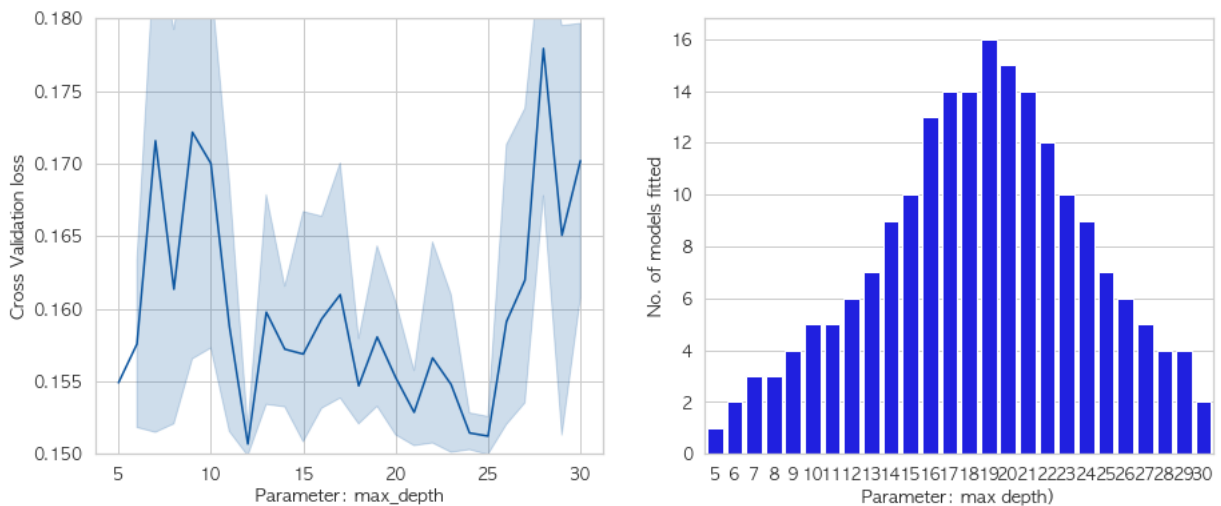


Figure 3.10: Hyperparameters optimization results for *GradientBoosting* - *maximum tree depth* parameter

Finally, with regard to the *maximum tree depth*, the performance exhibited by the classifiers was quite fluctuating. The optimizer prioritized values in the middle of the

proposed range but there was no determinant effect on the classifier performance.

The best combination of hyperparameters found for *GradientBoosting* was:

- *Learning rate*: 0.015
- *Subsample*: 0.71
- *Number of trees*: 500
- *Maximum tree depth*: 20

### ***RandomForest vs GradientBoosting***

The *RandomForest* algorithm trained with the best combination of hyperparameters found during the fine tuning process reported a performance of 0.804 as measured by *AUC-ROC* score on the validation data. The model took a total of 132 seconds on training and it presented a negligible 0.6% of overfitting relative to training performance. Despite the optimization process, an increase in performance was not achieved in comparison with the classifier performance with the default configuration of hyperparameters.

On the other hand, the optimized configuration of *GradientBoosting* classifier exhibited a performance of 0.84429 as measured by *AUC-ROC* score on the validation data, with overfitting representing a only 4% gap between training and validation performance. The classifier training time was 145 seconds. The optimization efforts derived in only a 1% improvement on classification performance relative to the default configuration.

Figures 3.11 and 3.12 illustrate the classifiers performance during training. In both cases, the number of trees presented positive but diminishing returns in terms of performance: as the number of trees fitted grows, the performance gain of fitting one extra tree (incremental gain) diminishes.

While *RandomForest* performance curves are noisy, reflecting the randomness introduced by the algorithm during training (see Figure 3.11), *GradientBoosting* performance curves are pretty smooth aligned with the boosting logic where each tree is grown using information from previously grown trees (see Figure 3.12).

Furthermore, the gap between training and validation performance remains constant with *RandomForest* algorithm across the different iterations (see Figure 3.11), meaning the model is not overfitting the training data. In contrast, for *GradientBoosting* ensemble, once the number of trees gets sufficiently large, the training performance curve quickly takes off from the validation curve (see Figure 3.12), reflecting the model starts to overfit the training data.

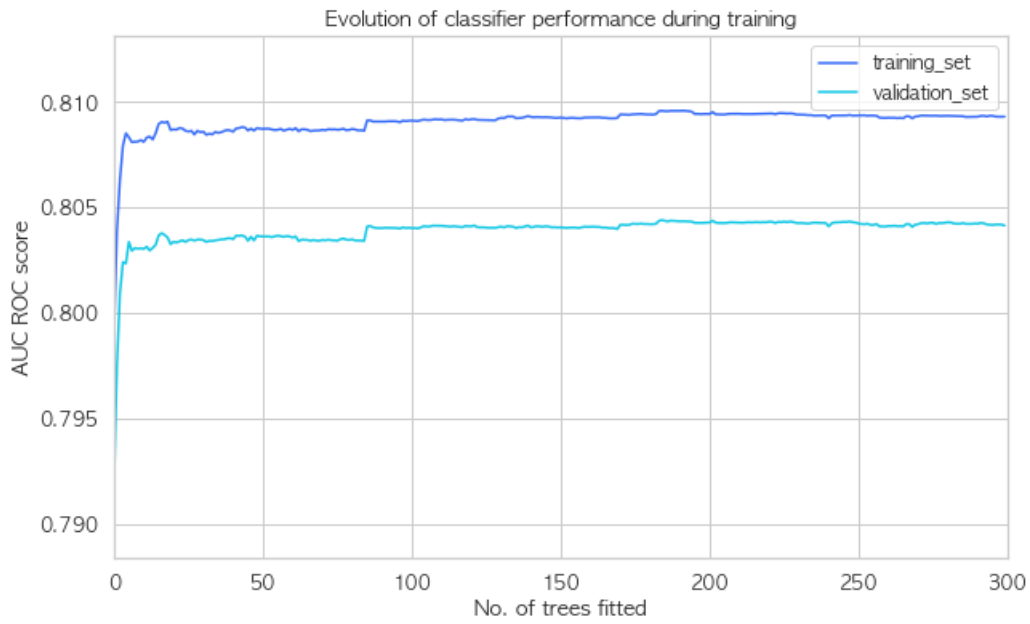


Figure 3.11: Random Forest - Evolution of classifier performance during training

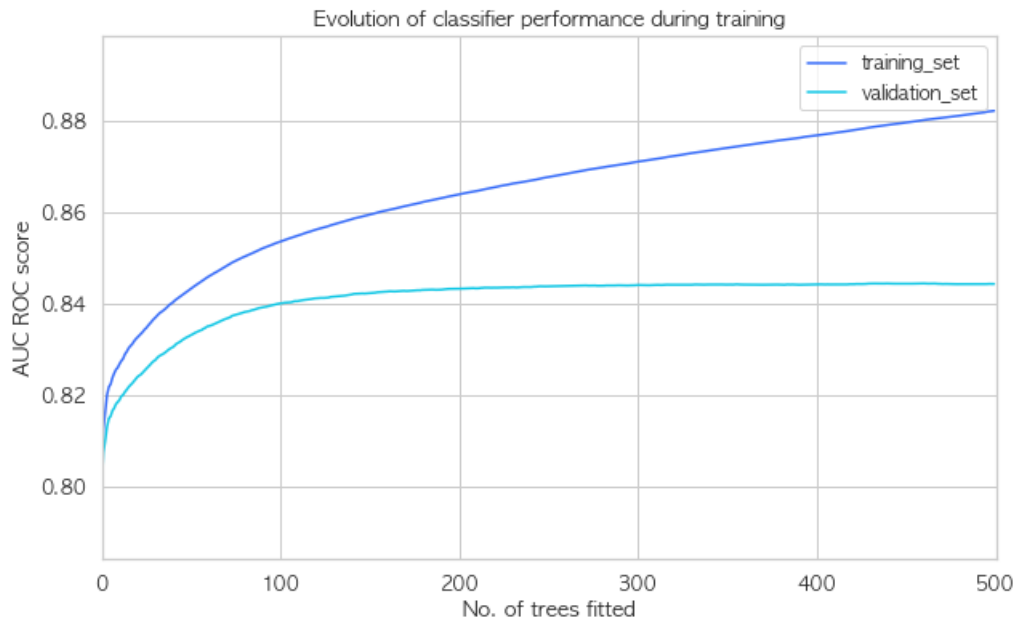


Figure 3.12: Gradient Boosting - Evolution of classifier performance during training

Despite the fact that *GradientBoosting* optimized model exhibits a greater overfitting than the *RandomForest* version, the classifier presents a superior performance both in training and validation set as Figure 3.13 illustrates. This being said, the *GradientBoosting* with optimized hyperparameters has been the classifier of choice for our final model.

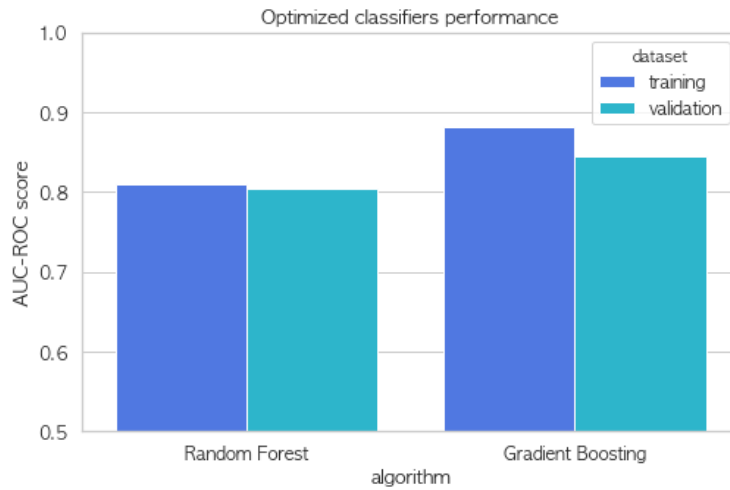


Figure 3.13: Comparative performance of optimized models

### 3.3 Final Model Assessment

Once the final model was chosen, it was evaluated in the test data that had been held back from the training and optimization process until that moment. *AUC-ROC* score on the test dataset was 0.84, approximately the same as validation *AUC-ROC* score. Overfitting measured as the relative difference of testing vs. training performance remained 4%, just as the validation data anticipated.

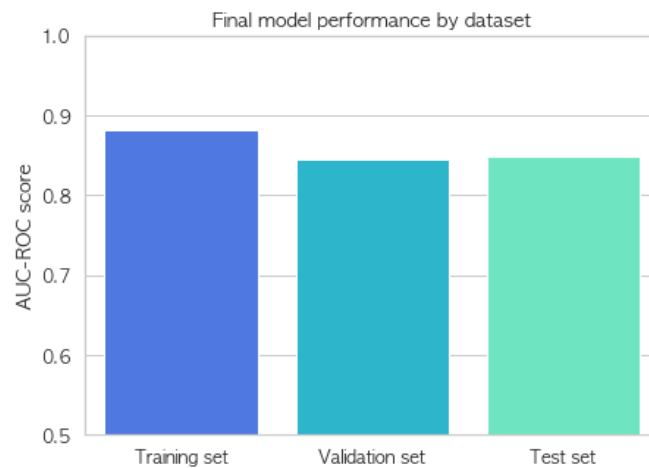


Figure 3.14: Final Model Cross-validation Performance

The *ROC curve* for the final model evaluated on test data was build as a graphical way of assesing the model performance. Figure 3.15 exhibits the relationship among *TPR* (*True Positive Rate*) and *FPR* (*False Positive Rate*) of the final model as measured on testing.

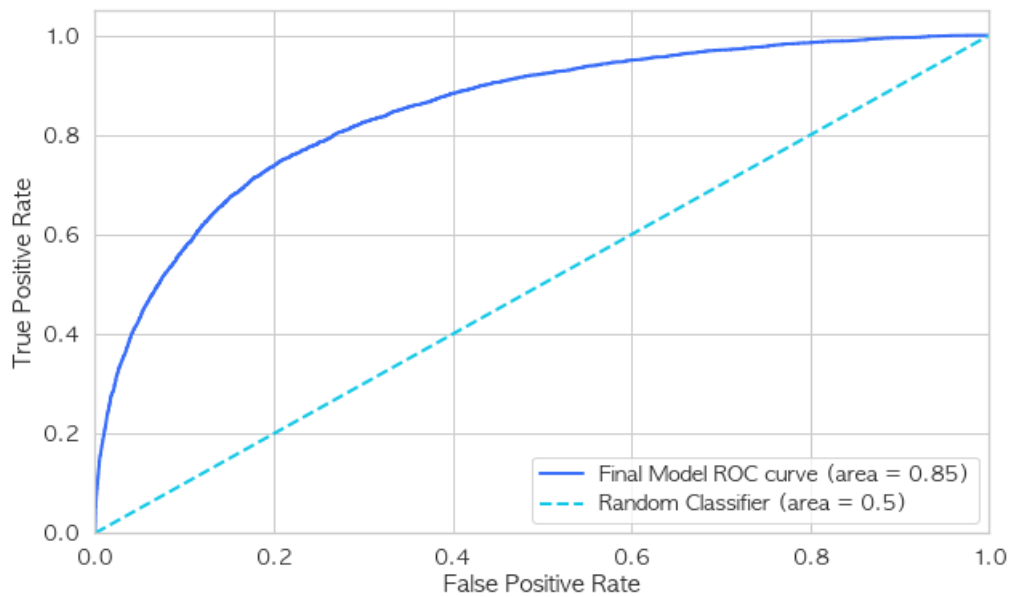


Figure 3.15: Receiver operating characteristic (ROC) Curve on Test set

In terms of the most important features of the final model, *hosp\_day\_number* leads the rank by far with 1,000 splits, as illustrated by Figure 3.16. It is followed by *patient\_age* and a series of numerical features related to the quantity of different services received during the admission (images studies, laboratory tests, surgeries, etc). Some date-related categorical features show up in the top too (for instance, *date\_weekday\_5* and *date\_weekday\_4*). Regarding text features, several tokens are present in the list, for example: *sangre*, *eme*, *arteriales*, *acido* just to name a few. Among the categorical most important features we find *origin* and *entity\_group*.

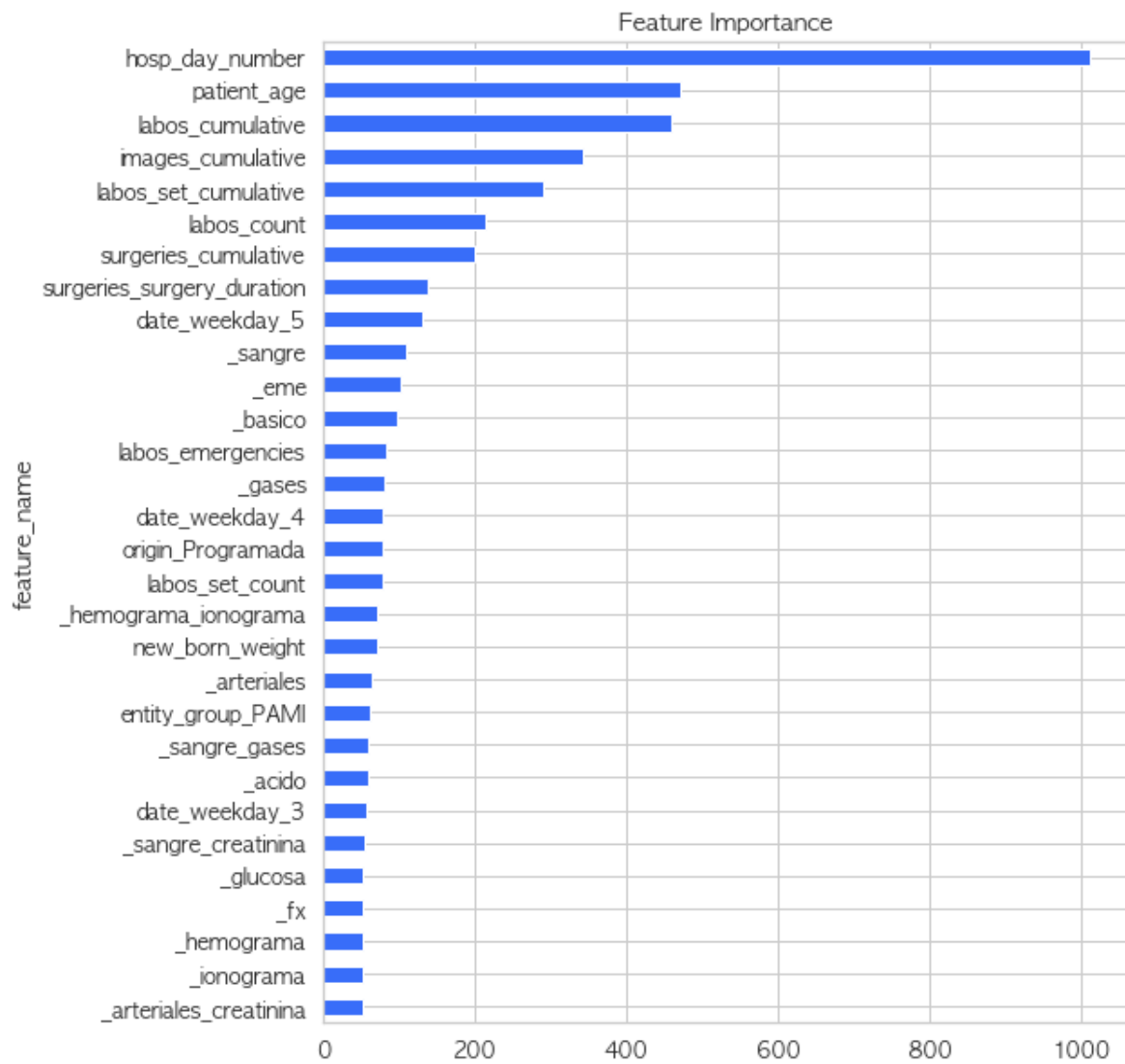


Figure 3.16: Top 30 features in terms of importance for final model

# Chapter 4

## Discussion

The current project has demonstrated how machine learning techniques can be applied to address the needs of a hospital’s Operations Management team by predicting patient discharge based on his profile and medical history.

By experimenting with different modelling approaches, we were able to come up with a model that combines different types of variables. We started by building a baseline that, making use of only two independent variables, was able to achieve a better performance than a random classifier. We found that, by adding more features to the model that needed little preprocessing, performance could be improved more than 25% relative to the baseline. Furthermore, we found that some date-related features had predictive potential and we added them to our model. Last but not least, we were able to incorporate text data related to the patient diagnosis taking advantage of *tf-idf* embeddings. These features, by themselves, were able to classify patient discharge with a 50% better performance than a random classifier. As a result, we ended up with a model containing 59 features, including variables related to patient demographics, his diagnosis, the length and characteristics of his stay at the hospital, etc.

Hyperparameter optimization with the Bayesian approach demonstrated having no such an impact on performance as we expected. Many reasons might account for that. On the one hand, several hyperparameters of *LightGBM* implementation for Python were not fine tuned. While we focused on both *RandomForest* and *GradientBoosting* main hyperparameters due to a matter of time, there are several peculiarities of the *LightGBM* implementation itself that could eventually lead to an improvement in performance if they’re fine tuned too. On the other hand, it could be that there was no much more information for the classifier algorithm to learn from the data as it was, thus more data or a different representation of it was necessary for an impactful performance increase to be accomplished. However, in the case of *GradientBoosting*, fine tuning achieved a 1% improvement in performance. The *learning rate*, the *number of trees* and the *maximum depth*, demonstrated to have all of them a strong correlation with the algorithm performance as illustrated in Section 3.2.

In terms of the ensemble tree-based algorithms, *GradientBoosting* outperformed *RandomForest* in all the experiments run, despite incurring in a little bit more over-

fitting. This result was aligned with our expectations. While there is no evidence supporting a generalized better performance of *GradientBoosting* classifier over *RandomForest*, the very different approaches both ensembles pursue would suggest this kind of result is not surprising. *RandomForest*'s main motivation is to reduce the variance that the plain and classic decision trees classifiers exhibit. It achieves this by building different models, each of them trained with only a random sample of the features available and then averaging them all. Thus the result is a reduction of the model overfitting rather than an increase of performance, just as our results illustrated. Instead, *GradientBoosting* focus is placed on reducing the bias of a decision tree, by sequentially learning new trees leaded by the gradient direction of the previously grown trees. This can eventually introduce a higher variance (i.e. overfitting) but also helps to improve the classifier performance in terms of its prediction error. In our case, the increase in overfitting is more than compensated by the increase in performance, so that in terms of the validation error *GradientBoosting* ends up winning the race, achieving a 0.84 *AUC-ROC* score on the validation data, relative to the 0.80 of the *RandomForest* classifier.

The final classifier performance, as measured on the test data, was 0.84 *AUC-ROC* score, presenting almost no difference with the metric on validation. This suggests our cross-validation strategy was successful in preventing the model from overfitting the validation data. While there is some overfitting to the training data (around 4% gap between in performance) it's not sufficiently large to be something problematic. A certain difference between the performance on the data used for training, in comparison with the data previously unseen by the classifier is always expected, due to the inductive bias that lays at the core of supervised learning.

In terms of which were the most important features for the decisions made by the algorithm's trees, not surprisingly the variable *hosp\_day\_number* leaded the rank in all of the experiments made, meaning that how many days that patient has already been in the hospital is the most important predictor of the probability of discharge. The patient age is another of the highlighted variables in the different experiments. Just as anticipated in the the exploratory stage (see Section 2.2.1), there is a strong relationship between this variable and the target variable, and the model was able to exploit it.

Other variables, also detected during the exploratory stage as potential predictors, showed up in the top 30 most important features as, for example, the *entity\_group* variable indicating the type of medical insurance the patient has. Moreover, many of the tokens extracted from the text preprocessing of diagnosis and related features appeared as the most important ones for the classifier. This last result was somehow unexpected. While we did presume that patient diagnosis was important to predict patient discharge, we did not expect *tf-idf* embeddings to produce as good a representation for them, especially because the *tf-idf* matrix was fitted among a relatively small corpus which was the training data and because there were a lot of medical jargon that we were not sure the tokenizer would be able to deal properly with. However this was not the case, the decision trees of the ensemble were able to make use of some pretty basic but apparently powerful concepts related to the patient diagnosis (for instance:



*glucosa, creatinina, acido, gases*, etc.) to predict patient discharge probability.



# Chapter 5

## Conclusions and Recommendations

### 5.1 Project Achievements

In the current project we posed the question of whether applying Machine Learning to the huge amount of data hospitals' information systems are flooded with, could propel the efficient utilization of scarce resources as beds, by predicting resource availability. To address this question, we worked together with one of the most relevant hospital in Buenos Aires, Argentina in order to develop a classifier that could predict whether a patient would be discharged in the following twenty four hours.

Data was extracted from the different tables within the hospital's database and processed with a customized ETL pipeline that produced a dataset with the necessary characteristics to build a supervised learning model.

Through exploration we were able to find certain patterns in the data that let us understand which information should the model be fed with. The model built takes into account different types of variables as patient demographic characteristics, patient diagnosis data, the history of services received during the admission, and some cyclical components related to the day of the week.

We experimented with different modelling approaches, feature engineering techniques, supervised learning algorithms and hyperparameters configuration. We optimized our classifiers using a Bayesian fine tuning approach. We applied a cross-validation strategy that combined both: *k-fold* and *train-validation-test* split in order to prevent overfitting.

The output of this work is a model that, given a patient at a certain date and his medical history upto that point in time, can predict the odds of the patient being discharged during that day. It does so by using a *GradientBoosting* classifier with an overall performance of 0.84 *AUC-ROC*. This means that, given a randomly chosen positive instance (discharged patient) and a randomly chosen negative one (not discharged patient) the chances that the model will score the positive case higher than the negative one is 84%.

## 5.2 Limitations

The results achieved by this project come together with its limitations. More specifically, there are four aspects of the current project where further work could be done aiming to improve the results achieved.

### Performance

First of all, the performance reached by the classifier, while quite promising, is not perfect. It is likely that accuracy can be improved by either incorporating more data into the model or experimenting with different representations or algorithms for the current data.

In terms of the model data, all the information regarding patient studies results (images, laboratory test results, etc) was left out of this project mainly due to the lack of standardization among the hospital's different data sources that made it more difficult to be processed and the lack of codification of these variables of interest.

With regard to data representation, more can be done in terms of feature engineering. For instance, trying a different encoding for categorical variables. *Feature binning* is a technique that has proved to work particularly well with tree-based algorithms. Another option is to try a different type of embeddings for text features. Unsupervised representation learning could be a way of approaching it. As for its implementation, it can either use some representation learning library for text (as `fastText` or `Word2vec`) which offer already trained representations of words to generate word embeddings. Another option, would be to develop a customized, unsupervised learning model to this aim. However, this would require a big development effort and lots of data to be trained with. Using transfer learning to combine the results of an already pre-trained neural network with the particularities of the problem at stake would probably be a more efficient approach.

Lastly, more hyperparameter fine tuning could be done. As discussed in the previous section, *LightGBM* algorithm configuration can be completely customized by taking advantage of all the hyperparameters available in the Python's API. In the current project, we iterated over the most important hyperparameters only, due to time restrictions. Probably, there is way more to explore on this side.

### Modeling

The way we modeled the classification problem, where each data point corresponds to a patient's day at the hospital has an inherent bias toward patients who have stayed longer at the hospital, as there are more observations per patient.

The algorithm will weight more data patterns coming from the patients who stay longer, because by doing so it will be able to classify properly a higher fraction of the data points. If patterns coming from these patients differ significantly from the characteristics of patients with shorter admission periods, our model will be skewed. While

the longer-stay patients represent a large fraction of the model observations, they represent a small fraction of the hospital admissions, so the hospital might actually be more interested in the performance of the classifier on the shorter admissions' patients.

In order to address this potential issue, two different options are available. One of them is to first clustered the patients into relevant groups for the hospital, and then build a different classifier for each group. Another way would be first balancing the dataset over the patients groups so that the training data has the same fraction of observations from each group of patients, and then fit a single classifier with that data. Both approaches should be tested in order to decide which one is more convenient by measuring the performance in the different groups of patients of interest and contrasting them.

### Probability Calibration

When faced with an unbalanced data classification task, a widely spread practice is to subsample the negative class in order to balance the training data. In our case, *LightGBM* implementation has a specific hyperparameter (*is\_unbalance*) that handles the imbalance data issue. More specifically it uses the values of  $Y$  (the target variable) to automatically adjust weights in an inversely proportional way to class frequencies in the input data.

While both approaches, either using the *is\_unbalance* parameter of *LightGBM* or balancing the dataset before feeding the model, should increase the overall performance of the classifier, they will also result in poor estimates of the individual class probabilities (*LightGBM*, 2020).

Because of this, if we were interested in using the scores predicted by the model as class probabilities instead of just the predicted class, it would be necessary to calibrate posterior probabilities to adjust for the undersampling/underweighting effect. In the paper *Calibrating Probability with Undersampling for Unbalanced Classification*, Dal Pazzolo et al. (2015) proposed a simple approach to adjust the posterior probabilities that could be a good way of dealing with this issue.

### Extending the prediction scope

Finally, in the current project, the focus has been on determining whether a patient will be discharged in the following twenty four hours or not. While the usefulness of this information for the hospital operations management team is not doubted, we are aware it would be even more beneficial to provide a prediction for the remaining length of stay of each inpatient, thus letting management plan ahead for the following days.

In order to do so, the problem at stake should be reframed into a survival analysis task. Survival analysis is a collection of longitudinal analysis methods for examining data having time until the occurrence of a particular event as an outcome variable (Emmert-Streib & Dehmer 2019). In this case, the event of interest would the patient

discharge.

The use of survival analysis in the healthcare industry is a well extended practice (Sá et al. 2007, Carter et al. 2004, Sekula et al. 2013). However, most approaches to survival analysis rely on standard statistical methods rather than machine learning. Following the spirit of this project, we consider it will be interesting to experiment with a machine learning approach to this survival analysis problem.

In the article *Machine learning for survival analysis: a case study on recurrence of prostate cancer* (Zupan et al. 2000), the authors propose a framework to apply machine learning methods to survival analysis. Moreover, deep learning techniques, specifically targeted to this type of problems, haven been developed during the last years. In *Deep Learning for Survival Analysis* (2020), Löschmann and Smorodina explore different deep learning models available for survival analysis (in particular *DeepSurv* and *Cox-nnet*) and provide an open-source implementation of each architecture. In planning to address the problem of the time-to-discharge prediction with machine learning techniques, these resources should be further explored.

### 5.3 Management Recommendations

In terms of how the output of this research project should be used by the hospital management team to improve their decision making process, we have built a forecaster tool that, based on the inference made by the machine learning model, is able to return a set of predictions for the expected discharges in the next twenty four hours, with different certainty levels. This information will be extremely useful for the decision makers as it will provide them not only with an expectation regarding the number of discharges but also a measure of the forecasting error they are taking by assuming each scenario.

According to our conversation with the hospital management team, it would be useful for them to have at hand three forecast levels:

- 95 % ← Management can almost assume that, at least, this level of beds is going to be free by the end of the day. Very conservative, almost no risk of actual discharges being less than this.
- 80 % ← This would be the main reference value for management expectations regarding the lower bound of discharges.
- 70 % ← Management should definitely not expect more than this level of discharges. Highest risk scenario.

In order to generate these forecasts, the scores provided by the model in a particular day (estimated probabilities of discharge) should be transformed into class predictions (either discharge or no-discharge), to then compute the sum of positive classes (discharges) and thus get an estimate of the day's discharges. It is also needed to estimate

the odds of actual daily discharges being at least the predicted.

Usually the classification task is approached as follows:

$$Predicted\_class(x) = \begin{cases} True & \text{if } f(x) \geq t \\ False & \text{if } f(x) \leq t \end{cases}$$

where  $f$  is the model's function (so  $f(x)$  is the model score for a given observation) and  $t$  is an arbitrary threshold.

The main challenge is then to define this threshold, as there is no *a priori* correct value for this parameter and it has severe consequences on the classification output. At the same time, the choice of the threshold has a direct impact on the odds of the discharges being over or under-estimated. Using a very low value for the threshold will rebound in the classifier capturing most of the actual discharges (i.e. high *recall*), but also overestimating them (low *precision*). This would lead to higher chances of a bed shortage scenario. Decision makers would expect more patients to be discharged and then would plan to occupy these beds, but at the end of the day, there would be less beds free than the expected, as actual discharges would not be as many as our forecast predicted.

On the other hand, if the threshold is set too high, the prediction will be quite precise, as only patients with a really high chance of getting discharged are going to be classified accordingly. But we will also more likely end up underestimating the actual number of discharges. This will affect the forecaster quality in a negative way as well. While we do not risk a bed shortage for expecting less discharges than the actual ones, this information will be, for sure, of less value for the decision makers. In the extreme, telling that discharges are expected to be at least zero would be correct 100 % of the times, but it will be of no value for the management.

To address these points, we have run a simulation over each observation (day) in the test data. For each day we forecasted the daily discharges using each of the possible threshold values (going as far as two decimal points) in the interval  $[0,1]$ . Then we compared the actual discharges with the classifier's predicted discharges and we analyzed the proportion of times in which the actual day discharges were at least the predicted.

For a randomly chosen day in the sample, our simulation output is the following:

	threshold	date	actual_discharges	predicted_discharges	observations	predicted_discharges_happened
0	0.0	2019-11-09	11.0	23	23	False
1	0.1	2019-11-09	11.0	20	23	False
2	0.2	2019-11-09	11.0	14	23	False
3	0.3	2019-11-09	11.0	12	23	False
4	0.4	2019-11-09	11.0	10	23	True
5	0.5	2019-11-09	11.0	8	23	True
6	0.6	2019-11-09	11.0	8	23	True
7	0.7	2019-11-09	11.0	8	23	True
8	0.8	2019-11-09	11.0	7	23	True
9	0.9	2019-11-09	11.0	3	23	True

Figure 5.1: Simulation output for a sample day

*As it can be seen, for this day in our sample we have that, during the first four simulation rounds, the forecast overestimates actual discharges, thus conducting to a bed shortage scenario. From threshold 0.4 onwards, actual discharges are at least the predicted. Figure 5.2 illustrates this.*



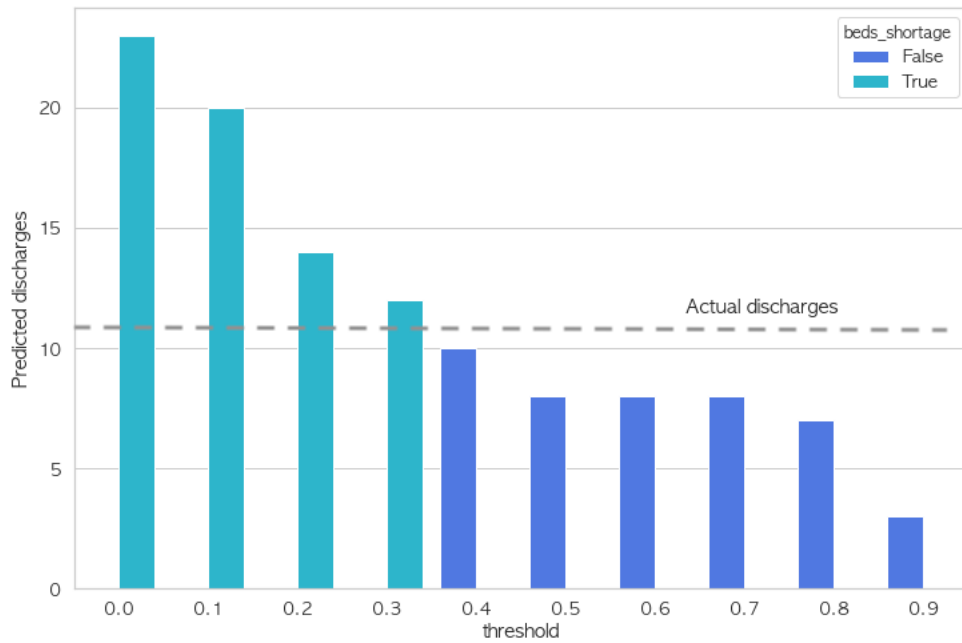


Figure 5.2: Simulation of discharges for a sample day (2019-11-9)

As a result of this simulation, we estimated a function (Figure 5.3) that captures the relationship between the applied threshold and the proportion of days in which the forecast for the lower bound of discharges is correct (i.e. actual discharges are at least the predicted). Based on this function, we determined the threshold values to use for each of the forecast certainty levels that we aim to provide (see the highlighted points in Figure 5.3).

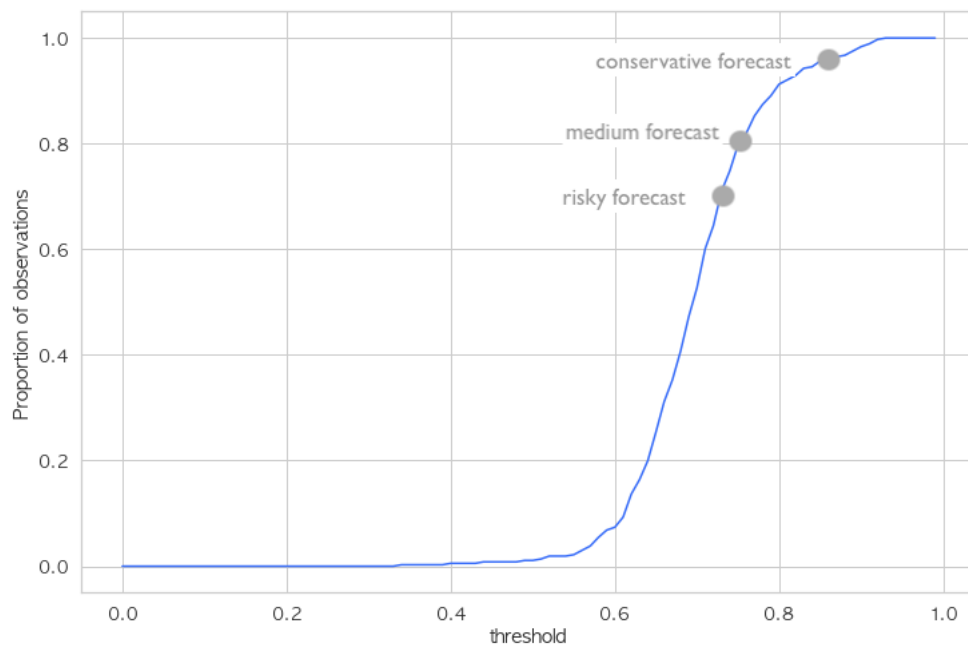


Figure 5.3: Simulation Results: % of observations where actual discharges are at least the predicted discharges by threshold used for classification.

Appendix E contains a table with the simulation output that resulted in the plot of Figure 5.3. This appendix also includes some visualizations illustrating the trade-off between precision and recall derived from the choice of different thresholds and how this impacted on the resulting curve.

To compress these results into a portable solution to be directly used by decision makers, we built a very intuitive user interface that given some date parameters, returns the three different forecasts of discharges for that day, with the associated risk of each of the predictions. (See Appendix F for an example visualization of the forecaster output).

# Appendix A

## List of variables in each dataset

### **Patients' hospitalizations dataset**

For each admission (row), the following data is available:

- Admission id
- Patient id
- Age
- Gender
- Birth date
- Insurance entity
- Entity group
- Admission date
- Admission time
- Admission year
- Admission month
- Origin
- Admission physician
- Admission sector
- Last sector
- Last category
- Isolation
- Last room

- Last bed
- Discharge date
- Discharge time
- Discharge year
- Discharge month
- Discharge reason
- Pre discharge date
- First sector
- Administrative diagnosis
- Diagnosis code
- Presumptive diagnosis
- Discharge diagnosis code
- Discharge diagnosis
- Discharge register date
- Discharge register time
- Discharge physician
- Discharge summary
- Discharge summary physician
- Surgery
- Express hip surgery
- Admission length days
- Responsible sector
- Second responsible sector
- Emergency admission datetime
- Has previous admission
- Previous admission id
- Previous admission date
- Previous discharge date

- Previous sector
- Previous discharge diagnosis
- Discharge ambulance
- Newborn weight
- Newborn gestational age
- High risk TEP
- Low risk TEP
- ARM TEP
- CEC TEP
- Request number
- Request origin
- Request diagnosis
- Request sector
- Notified
- Request user

### **Laboratory studies dataset**

For each laboratory study the following data is available:

- Laboratory study id
- Status
- Study Date
- Study Time
- Study year-month
- Sector
- Admission id
- Patient id
- Admission date
- Discharge date

- Entity id
- Insurance entity
- Entity affiliate id
- Entity group
- Emergency
- Requester name
- Requester role
- Study code
- Study description
- No. of studies

**Images studies dataset**

For each image study the following data is available:

- Image study id
- Status
- Study date
- Study time
- Study year-month
- Sector
- Admission id
- Patient id
- Admission date
- Discharge date
- Entity id
- Insurance entity
- Entity affiliate id
- Entity group
- Emergency

- Requester name
- Requester role
- Study code
- Study description
- No. of images studies

### **Surgeries dataset**

For each surgery the following data is available:

- Operating room
- Surgery date
- Surgery weekday
- Surgery year-month
- Surgery scheduled time
- Admission id
- Surgery type
- G.A.P.
- Surgery id
- Origin
- Patient id
- Gender
- Age
- Discharge type
- Entity description
- Diagnosis
- Scheduled surgery
- Actual surgery
- Surgery physician
- Dependency

- Anesthesia type
- ASA
- Antisepsia
- Prophylactic ATB
- Dosis mg
- Bed request
- Hemotherapy
- Hemotherapy ok
- X-ray
- Cardiologist
- Supplies
- Supplies ok
- Protocol no.
- Service description
- Surgery code
- Sector bed
- Specialization
- Estimated duration
- Surgery starttime
- Surgery endtime
- Surgery delay
- Admission date
- Specialization code
- Entry time
- Exit time Anesthesia starttime
- Anesthesia endtime
- Post-surgery condition
- Admission date



- Admission time
- Discharge date
- Discharge time
- Repeated surgery
- No. of surgeries
- Hips surgery
- Injury condition
- Pre-surgery-prep duration
- Surgery duration
- Post surgery duration
- Surgery-prep duration
- Antibiotic
- Seeding
- No. of assistans
- Anesthetist id
- No. of pregnancies
- No. births
- No. of cesarean
- Nulliparous
- Newborn id
- Newborn admission id
- Newborn weight
- Newborn gestation age
- Newborn alive
- Bact positive
- Scheduled surgery done

**Sectors admissions dataset**

For each admission to a particular sector, the following data is available:

- Admission id
- Patient id
- Sector admission date
- Sector admission time
- Sector code
- Category
- Sector admission datetime

**Hospital sectors dataset**

Each row represents a sector and the following data is available:

- Sector code
- Sector name
- Sector type

## Appendix B

### Data Quality Reports

## Admissions Dataset Quality Report

### Categorical Variables

Getting categorical quality report for: admissions dataset...  
 CATEGORICAL VARIABLES: 62  
 Getting tabular report for categorical variables...

	dataset	variable	count	unique	top	freq	missings	missings %
0	admissions	admission_id	87720	87720	492113-6	1	0	0
1	admissions	patient_id	87720	64156	3727595-3	89	0	0
2	admissions	gender	87720	3	F	48188	0	0
3	admissions	birth_date	87709	28382	1950-01-01	118	11	0.013
4	admissions	insurance_entity	87720	117	OSUTHGRA O.S PERSONAL GASTRONOMICO	27165	0	0
5	admissions	entity_group	87720	7	BRISTOL	31112	0	0
6	admissions	admission_date	87720	1045	2019-07-10	133	0	0
7	admissions	admission_time	87720	26100	14:11:00	78	0	0
8	admissions	admission_year	87720	3	2018	30848	0	0
9	admissions	admission_month	87720	12	Ago	8401	0	0
10	admissions	origin	87715	5	Emergencias	33412	5	0.0057
11	admissions	admission_physician*	86255	1684	3140763216979270967	11246	1465	1.7
12	admissions	admission_sector	87720	35	EME	35215	0	0
13	admissions	last_sector	87720	36	EME	11523	0	0
14	admissions	last_category	87720	3	C	49014	0	0
15	admissions	isolation	1388	1	*	1388	86332	98
16	admissions	last_room	87720	477	442	4797	0	0
17	admissions	last_bed	87720	32	1	62320	0	0
18	admissions	discharge_date	87720	1046	2019-04-12	151	0	0
19	admissions	discharge_time	87720	1439	13:16:00	189	0	0
20	admissions	discharge_year	87720	3	2018	30779	0	0
21	admissions	discharge_month	87720	12	Ago	8454	0	0
22	admissions	discharge_reason	87720	6	Alta Medica	81315	0	0
23	admissions	pre_discharge_date	13880	447	2018-11-22	80	73840	84
24	admissions	first_sector	87720	36	EME	11745	0	0
25	admissions	administrative_diagnosis	87714	20789	NACIDO VIVO NEOM	3934	6	0.0068
26	admissions	diagnosis_code	87720	4881	XX2	12924	0	0
27	admissions	presumptive_dianogsis	87720	4865	No EspecificaCodigo	12924	0	0
28	admissions	discharge_diagnosis_code	87720	5806	765.29	4619	0	0
29	admissions	discharge_diagnosis	87720	5793	37 O Mas Semanas Completas De Gestacion, Recien Nacido	4619	0	0

Figure B.1:

\*Sensitive variables' labels have been encrypted due to confidentiality reasons

30	admissions	date_registered_discharge	86044	1045	2018-08-24	152	1676	1.9
31	admissions	time_registered_discharge	86044	43697	09:27:05	10	1676	1.9
32	admissions	discharge_physician*	85825	811	8972760370031577475	1674	1895	2.2
33	admissions	discharge_summary	87720	2	True	81000	0	0
34	admissions	discharge_summary_physician*	80637	765	8972760370031577475	1509	7083	8.1
35	admissions	surgery	87709	2	False	47132	11	0.013
36	admissions	express_hip_surgery	87720	2	False	87184	0	0
37	admissions	responsible_sector	62718	62	CLINICA MEDICA	21587	25002	29
38	admissions	second_responsible_sector	3000	46	CLINICA MEDICA	1631	84720	97
39	admissions	emergency_admission_datetime	21085	1	1970-01-01	21085	66635	76
40	admissions	emergency_service*	21085	13	-1466667266627811949	7205	66635	76
41	admissions	has_previous_admission	87720	2	False	84057	0	0
42	admissions	previous_admission_id	1978	1978	537309-5	1	85742	98
43	admissions	previous_admission_date	1970	861	2019-02-07	8	85750	98
44	admissions	previous_discharge_date	1978	851	2017-11-10	7	85742	98
45	admissions	previous_sector	1978	36	EME	386	85742	98
46	admissions	previous_discharge_dianosis	1978	666	37 O Mas Semanas Completas De Gestacion, Recien Nacido	94	85742	98
47	admissions	discharge_ambulance	86044	2	False	76988	1676	1.9
48	admissions	PIM2TEP	429	73	0.0	65	87291	1e+02
49	admissions	high_risk_TEP	429	8	0.0	402	87291	1e+02
50	admissions	low_risk_TEP	429	3	0.0	402	87291	1e+02
51	admissions	ARM_TEP	429	2	False	324	87291	1e+02
52	admissions	CEC_TEP	429	2	False	420	87291	1e+02
53	admissions	request_number	20477	20461	43504.700520833	2	67243	77
54	admissions	request_origin	20477	3291	CENTRO DE SALUD PUBLICO	1432	67243	77
55	admissions	request	20310	3680	DOMICILIO	5785	67410	77
56	admissions	request_diagnosis	20467	7951	FX DE CADERA	426	67253	77
57	admissions	request_sector	20470	8	INTERNACION GENERAL	16855	67250	77
58	admissions	notified*	15226	1547	7676720163663508467	2376	72494	83
59	admissions	request_user*	20477	63	4455990325853785004	3541	67243	77
60	admissions	admission_datetime	87720	86751	2017-11-14 12:08:00	3	0	0
61	admissions	discharge_datetime	87720	80101	2018-09-22 13:31:00	5	0	0

Figure B.2:

\*Sensitive variables' labels have been encrypted due to confidentiality reasons

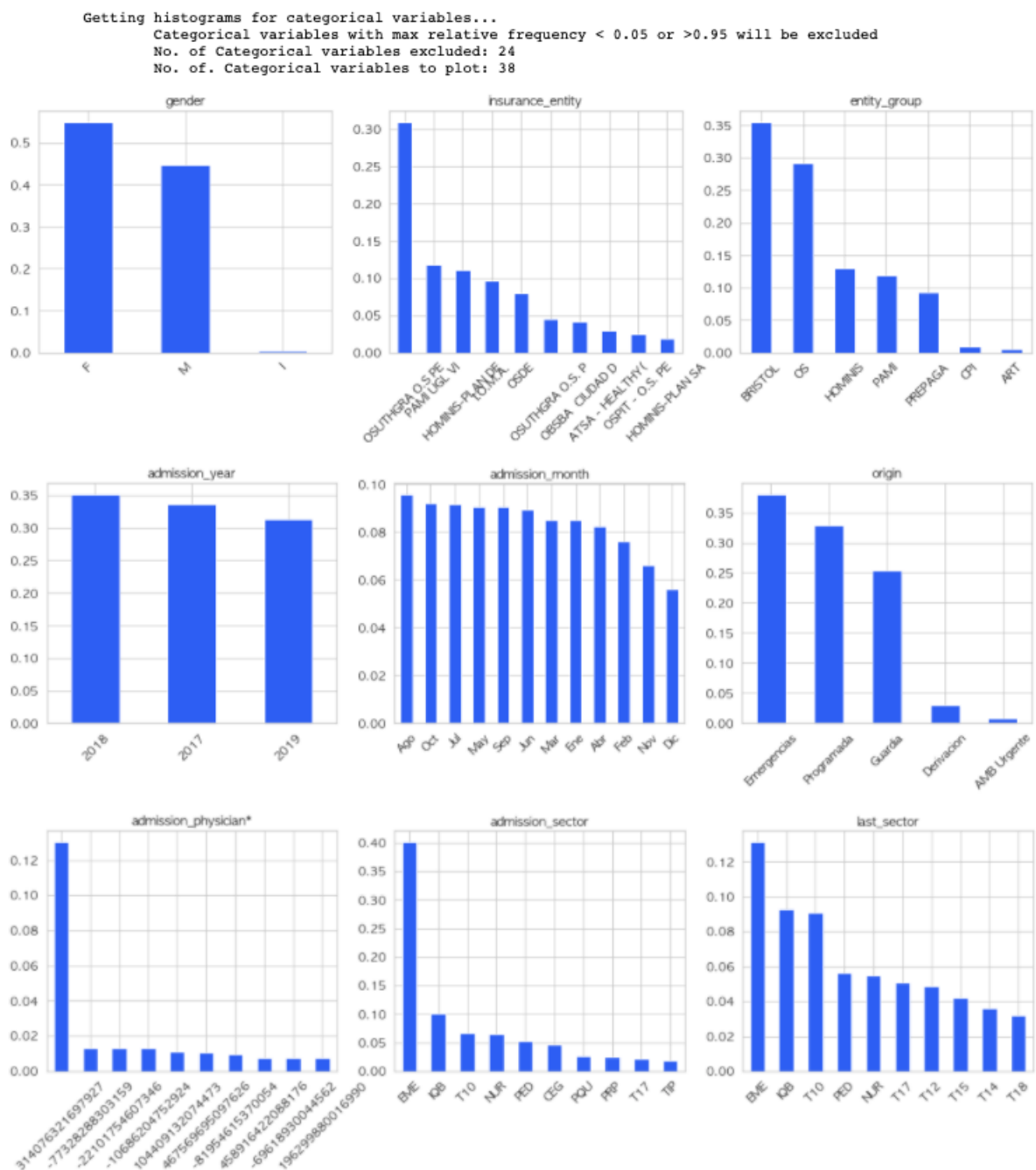


Figure B.3:

\*Sensitive variables' labels have been encrypted due to confidentiality reasons

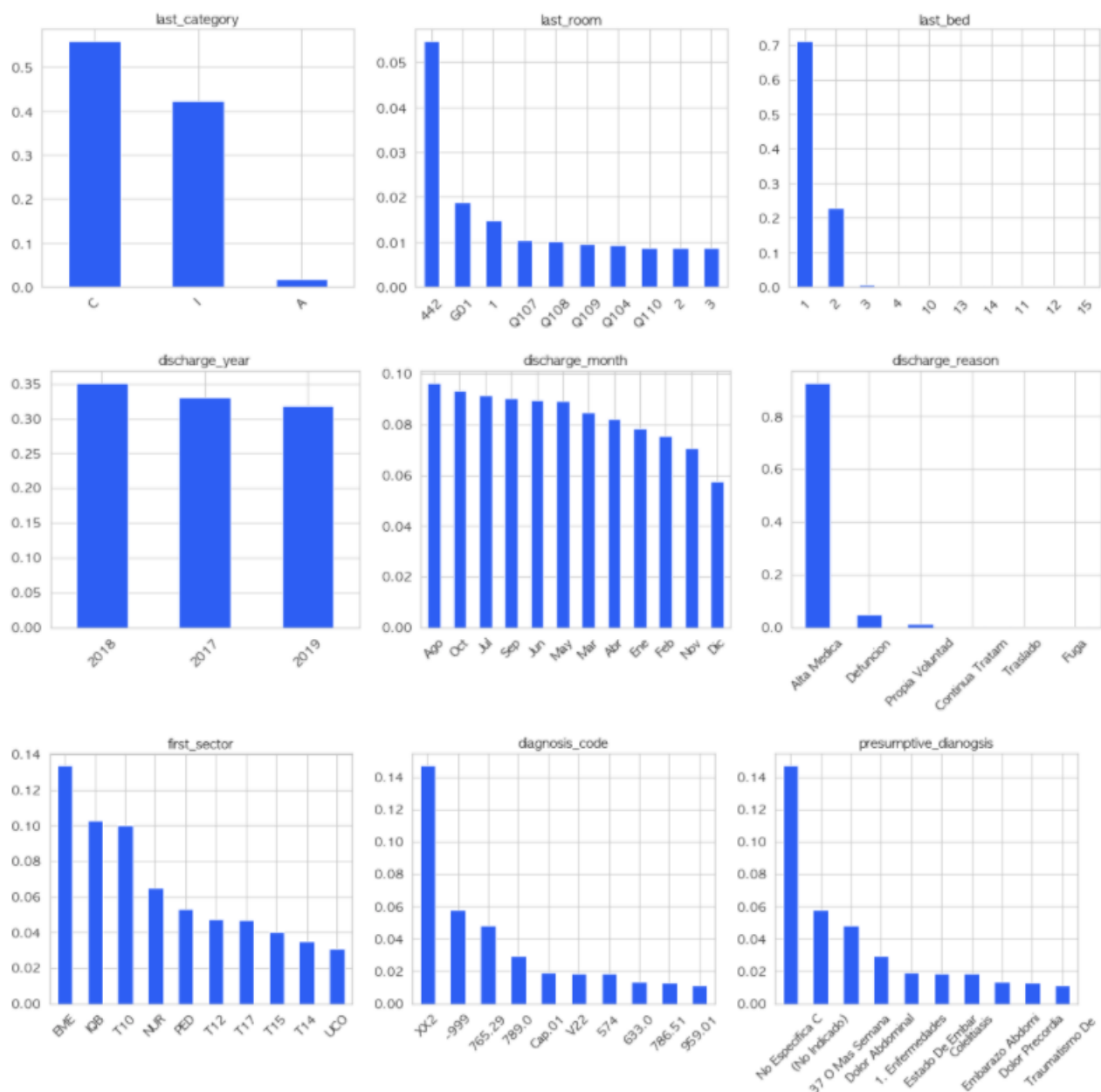


Figure B.4

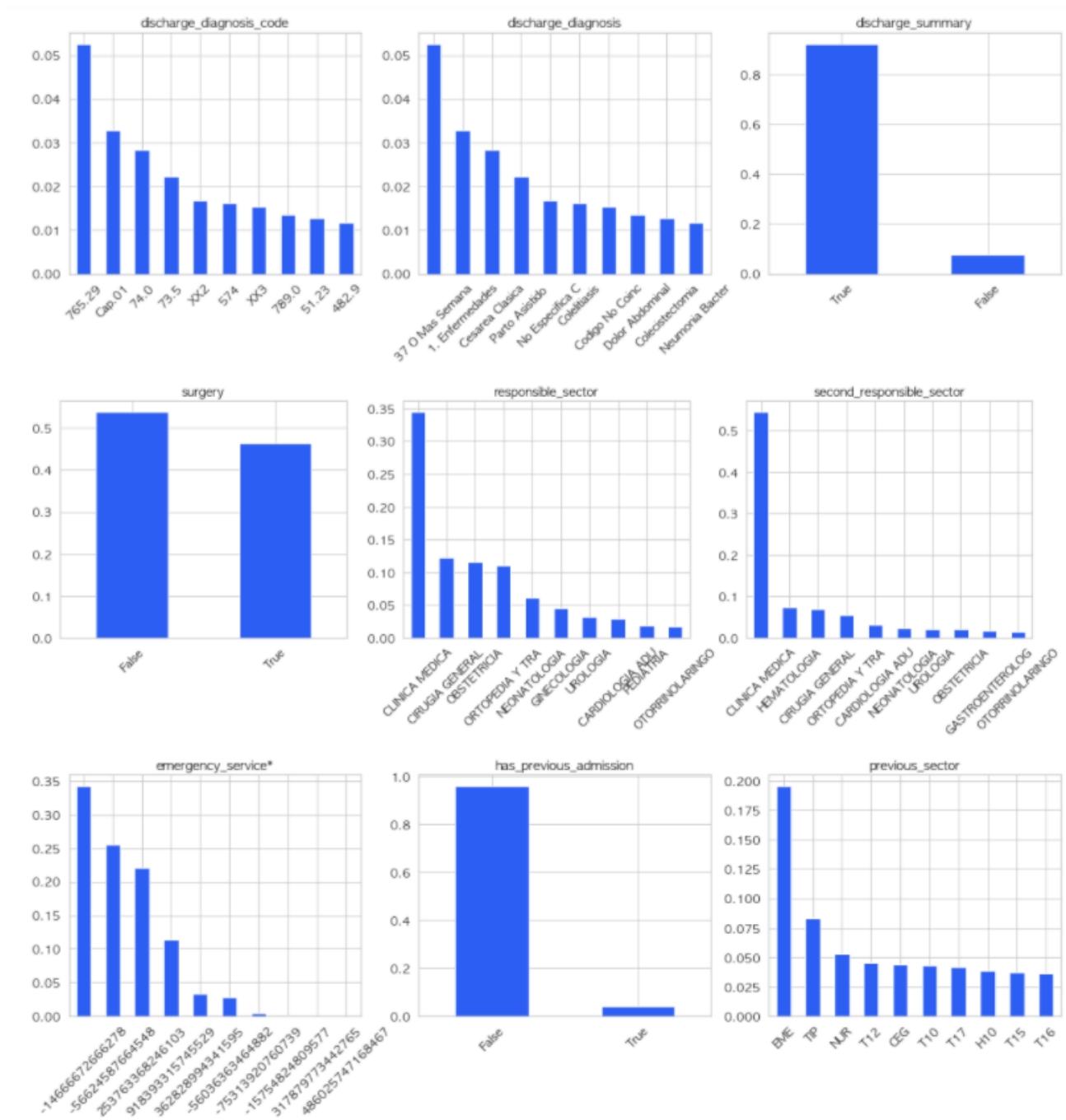


Figure B.5:

\*Sensitive variables' labels have been encrypted due to confidentiality reasons



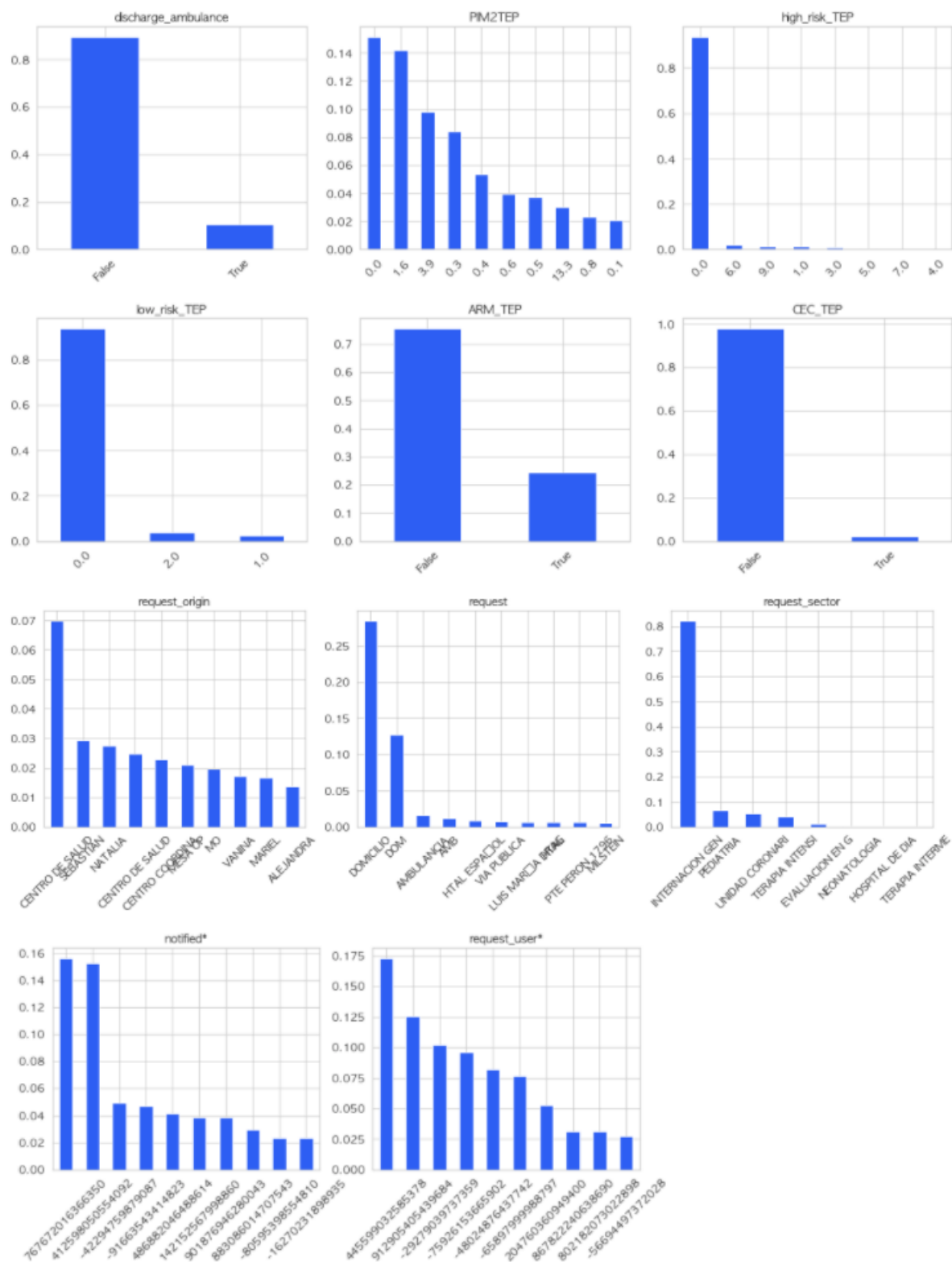


Figure B.6:

\*Sensitive variables' labels have been encrypted due to confidentiality reasons

## Admissions Dataset Quality Report

### Numerical Variables

Getting numerical quality report for: admissions dataset...

NUMERICAL VARIABLES: 4

Getting tabular report for numerical variables...

	dataset	variable	count	mean	std	min	25%	50%	75%	max	missings	missings %
0	admissions	age	87720.0	45.15	27.41	0.0	25.0	45.0	68.0	107.0	0.0	0.00
1	admissions	admission_lenght_days	87720.0	6.01	15.43	1.0	1.0	2.0	5.0	896.0	0.0	0.00
2	admissions	new_born_weight	5484.0	3179.09	800.11	0.0	2920.0	3310.0	3650.0	5530.0	82236.0	93.75
3	admissions	new_born_gestation_age	87720.0	2.25	9.04	0.0	0.0	0.0	0.0	71.0	0.0	0.00

Getting distribution plots for numerical variables...

No. of. Numerical variables to plot: 4

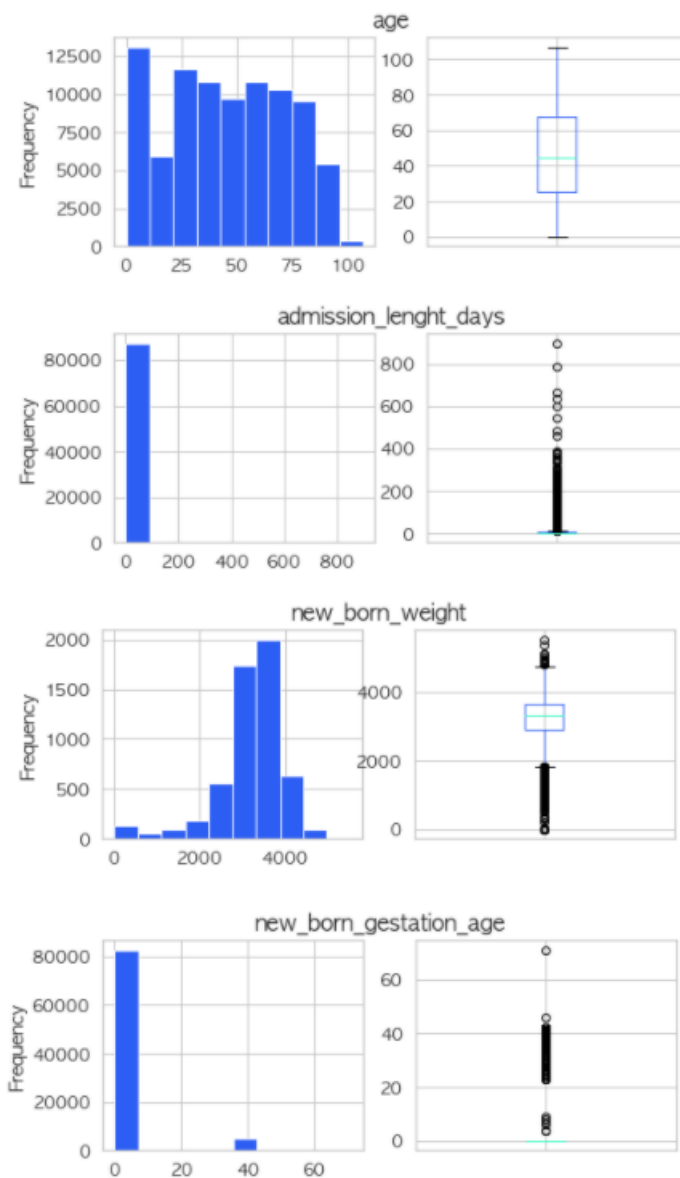


Figure B.7

## Appendix C

### Data Quality Plan

Table C.1: Data Quality Plan

Data Quality Issue	Issue description	Dataset	Features affected	Potential Handling Strategies
Missing values	More than 60% structural missing values	Admissions	Isolation Pre_discharge_date Second_responsible_sector Emergency_admission_datetime Emergency_service Previous_admission_id Previous_admission_date Previous_discharge_date Previous_sector Previous_discharge_diagnosis	Create a missing value category (for example 'N/A')
		Surgeries	New_born_id_related New_born_admission_id New_born_alive New_born_weight	
	More than 60% missing values due to data collection errors	Admissions	Request_number Request_origin Request Request_sector Notified Request_user PIM2TEP High_risk_TEP Low_risk_TEP ARM_TEP CEC_TEP	Discard features

Table C.1 continued from previous page

Data Quality Issue	Issue description	Dataset	Features affected	Potential Handling Strategies
		Surgeries	Bact_positive	
Irregular cardinality	Extremely high cardinality features (valid data)	Admissions	Admission_id Patient_id	Discard features (uninformative for predictive analytics)
		Surgeries	Protocol_no Surgery_id Patient_id Admission_id New_born_admission_id New_born_admission_id	
		Laboratory	Labo_id Labo_pun	
		Images	Image_id Image_pun	
	Continuous features treated as categorical	Admissions	Admission_time Discharge_time Time_registered_discharge Admission_datetime Discharge_datetime	Convert to the appropriate type

Table C.1 continued from previous page

Data Quality Issue	Issue description	Dataset	Features affected	Potential Handling Strategies
		Surgeries	Surgery_scheduled_time Surgery_starttime Surgery_endtime Entry_time Exit_time Anesthesia_starttime Anesthesia_endtime Admission_time Discharge_time	
		Laboratory	Labo_time	
		Images	Image_time	
Unstandardized categories		Admissions	Admission_physician Administrative_dianogsis Presumptive_dianogsis Discharge_diagnosis Discharge_physician Discharge_summary_physician Previous_discharge_diagnosis	Map to a standard set of levels/ Treat as text (represent with embeddings)
		Surgeries	Surgery_physician Scheduled_surgery Actual_surgery	
		Laboratory	Requester_role Requester_name	

Table C.1 continued from previous page

Data Quality Issue	Issue description	Dataset	Features affected	Potential Handling Strategies
Outliers	Large number of outliers	Images	Requester_role Requester_name	
		Admissions	Admission_length_days New_born_weight	Use domain knowledge to determine whether or not they are valid outliers. If invalid correct or assign missing value.
		Surgeries	Dosis_mg Estimated_duration Surgery_delay No_of_surgeries No_of_pregnancies No_of_births No_of_cesarean New_born_weight New_born_gestation_age	If valid, take them into account for the machine learning technique of choice.
		Laboratory	No_of_studies	
Implausible outliers		Images	No_of_studies	
		Admissions	New_born_gestation_age New_born_weight	Assign missing value where applicable





## Appendix D

# Feature importance plots for different model experiments

Feature importance plots for experiments set #2

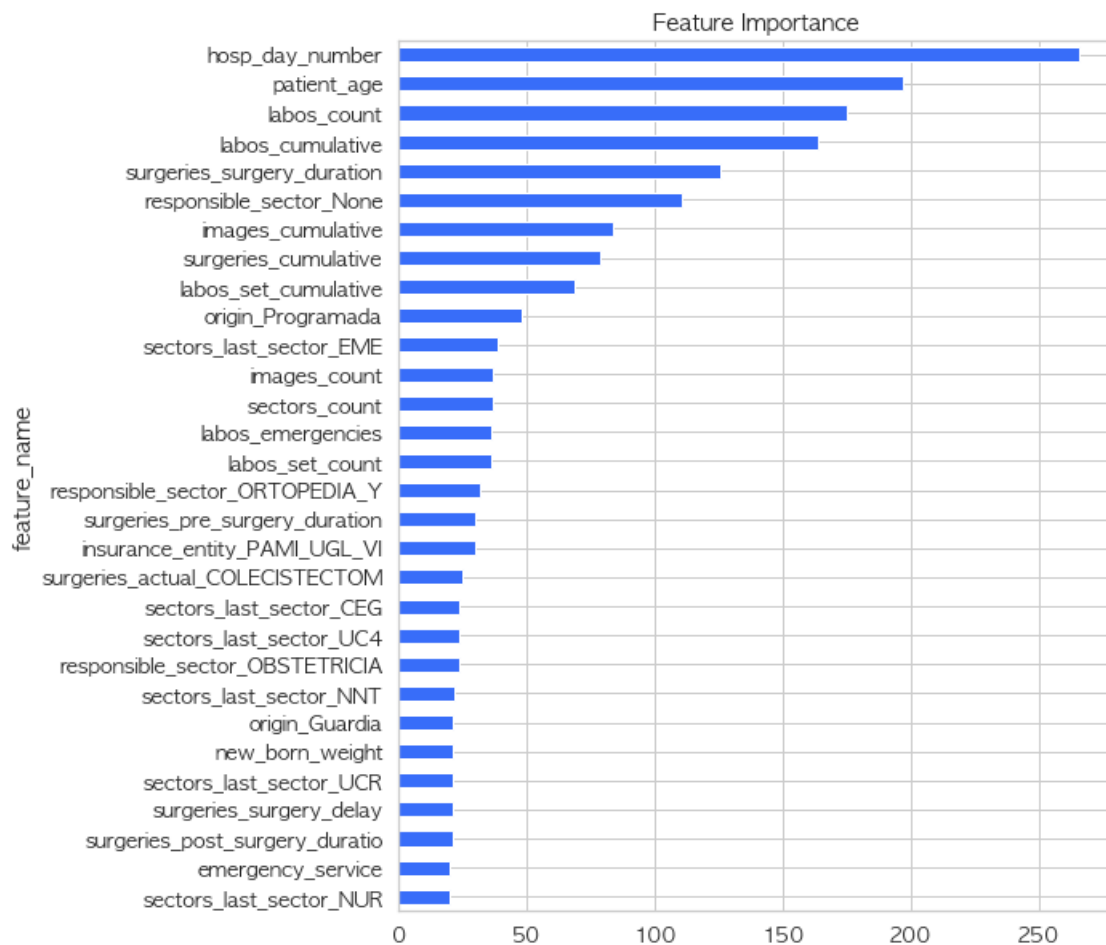


Figure D.1: Feature importance for GradientBoosting algorithm

## Feature importance plots for experiments set #3

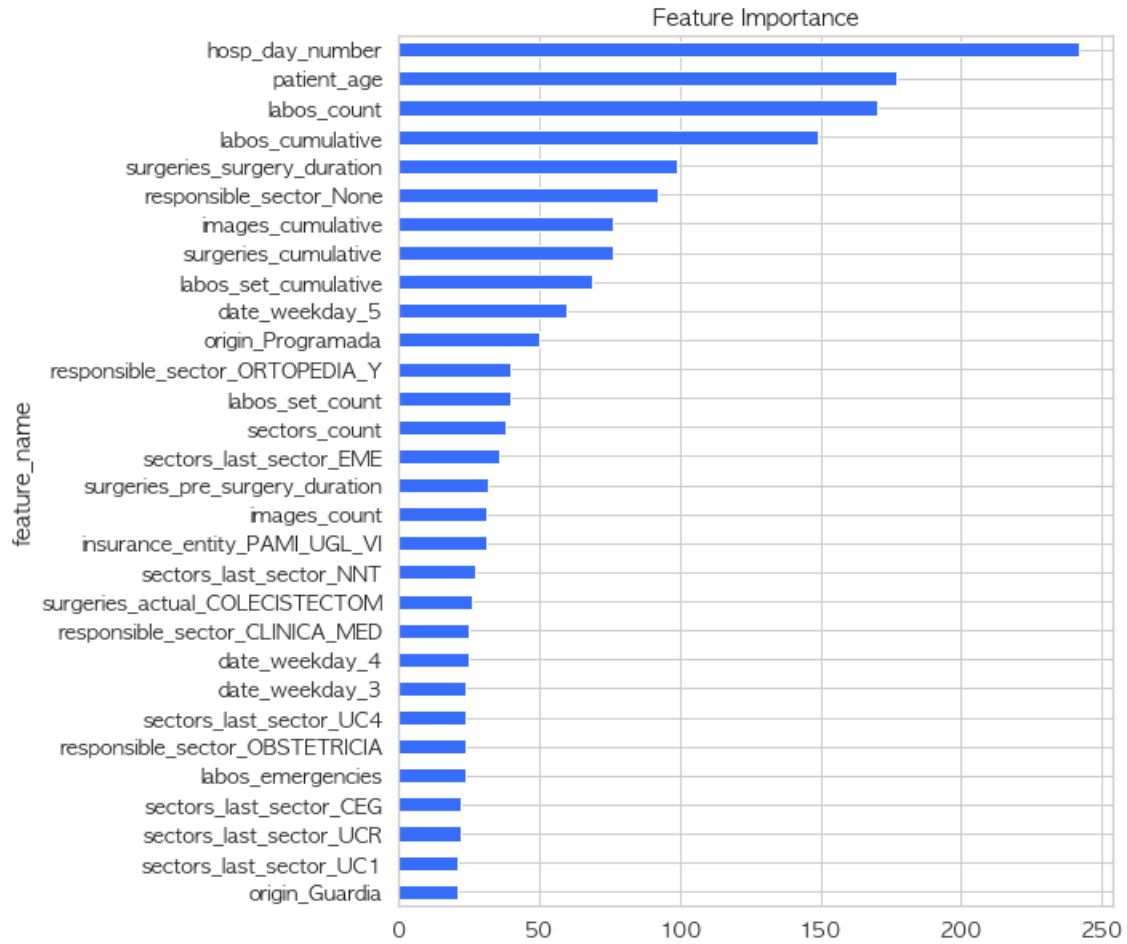


Figure D.2: Feature importance for GradientBoosting algorithm

### Feature importance plots for experiments set #4

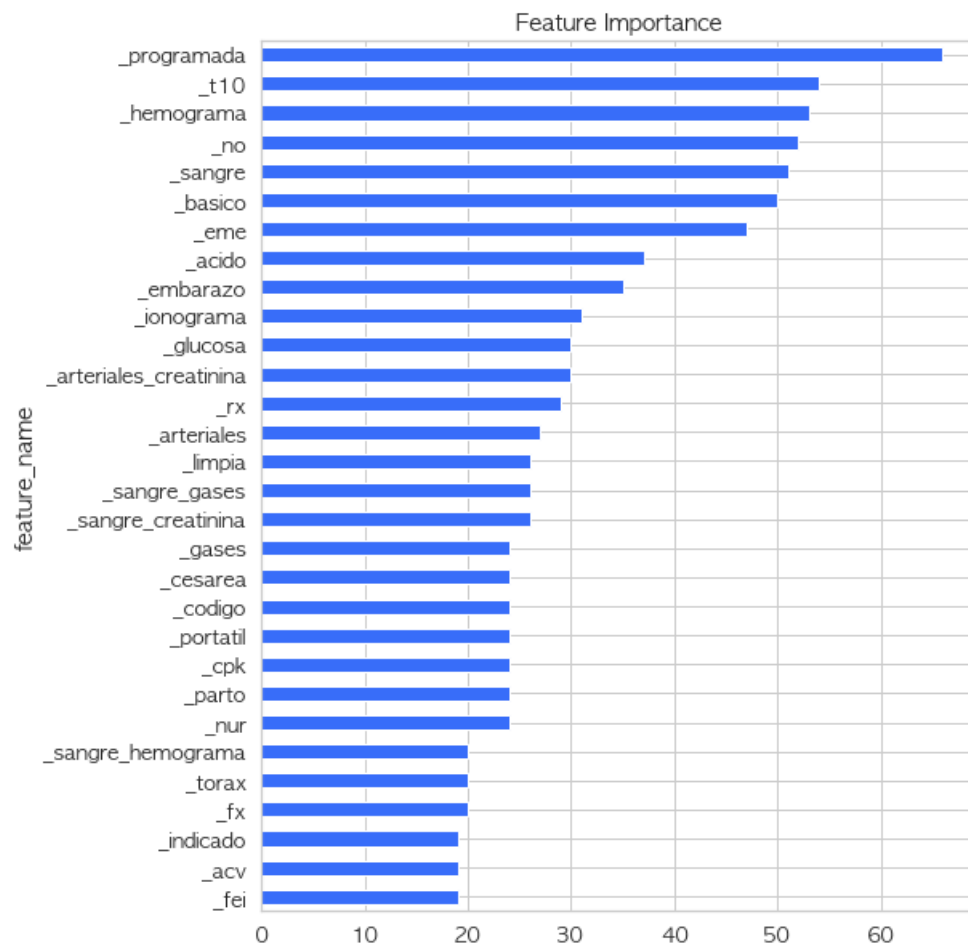


Figure D.3: Feature importance for GradientBoosting algorithm

## Feature importance plots for experiments set #5

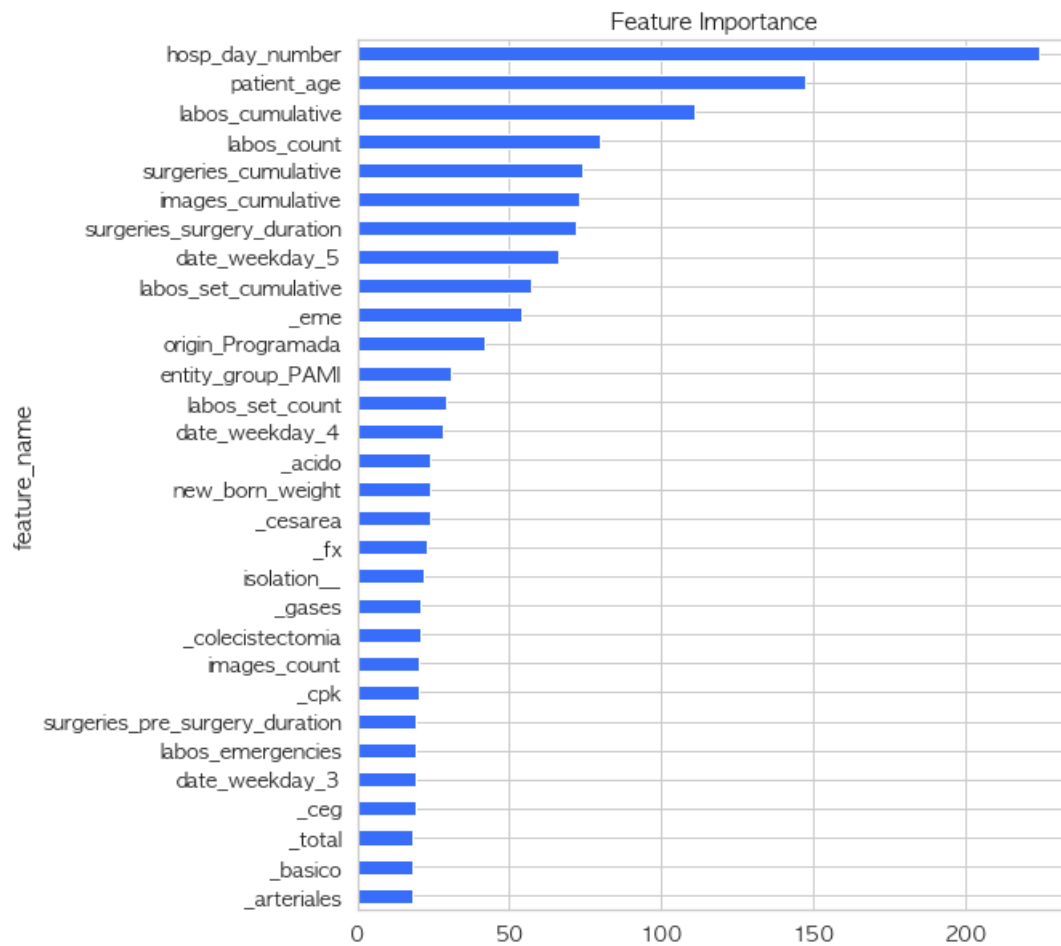


Figure D.4: Feature importance for GradientBoosting algorithm

# Appendix E

## Simulation results

threshold	predicted_discharges_happened	accuracy	recall	precision	confidence
0.00	0.000000	0.141895	1.000000	0.141895	0.000000
0.05	0.000000	0.205921	0.996091	0.150265	0.000000
0.10	0.000000	0.305027	0.983522	0.164324	0.000000
0.15	0.000000	0.406358	0.960765	0.184003	0.000000
0.20	0.000000	0.492045	0.934592	0.206338	0.000000
0.25	0.000000	0.560925	0.908653	0.229690	0.000000
0.30	0.000000	0.617265	0.878971	0.252817	0.000000
0.35	0.002732	0.666192	0.845392	0.276616	0.002732
0.40	0.005464	0.710043	0.813060	0.303804	0.005464
0.45	0.008197	0.745790	0.773561	0.329206	0.008197
0.50	0.010929	0.780624	0.734145	0.363811	0.010929
0.55	0.021858	0.812676	0.682112	0.401080	0.021858
0.60	0.073770	0.836672	0.623930	0.436980	0.073770
0.65	0.254098	0.855510	0.557449	0.476741	0.254098
0.70	0.527322	0.869697	0.489604	0.523316	0.527322
0.75	0.797814	0.878006	0.427694	0.570845	0.797814
0.80	0.912568	0.881606	0.367762	0.615919	0.912568
0.85	0.956284	0.883845	0.318711	0.652348	0.956284
0.90	0.983607	0.879454	0.249936	0.692152	0.983607
0.95	1.000000	0.869965	0.095507	0.584368	1.000000

Figure E.1: Summary of simulation results

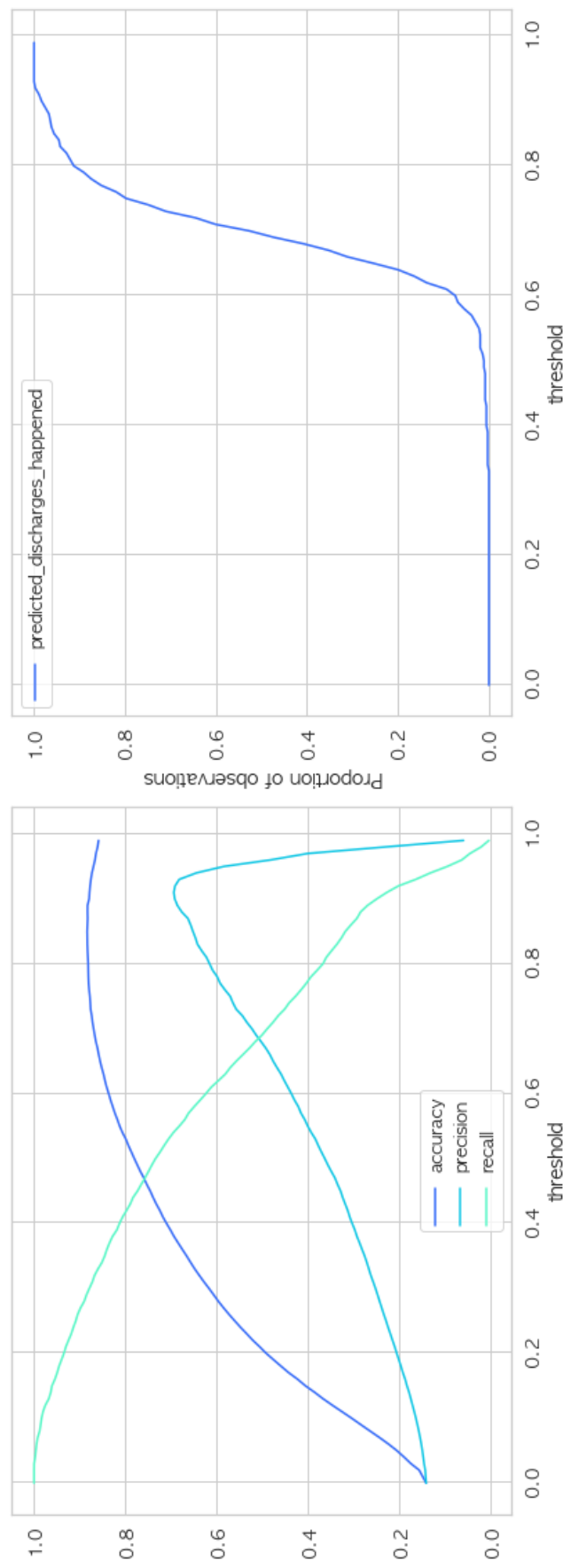



Figure E.2: Precision and recall trade-off for different threshold levels and its impact on accuracy and the proportion of observations in which predicted discharges happened

# Appendix F

## Forecaster output example

 UNIVERSIDAD  
TORCUATO DI TELLA


### Discharges Forecaster

**Instructions:**

Specify the desired date and then click on **Run Forecast**. You will get three estimates, with different confidence levels, of expected discharges for that date..

If there is no data for the specified date an error message will be prompt.

For more information visit visit: [https://github.com/josedallavia/A-Machine-Learning-Approach-for-Prediction-of-Hospital-Bed-Availability'](https://github.com/josedallavia/A-Machine-Learning-Approach-for-Prediction-of-Hospital-Bed-Availability)

Select Date  

**Run Forecast**

Forecasting discharges for date: 2019-10-22  
With 95 % of confidence, number of discharges will be at least 6  
With 80 % of confidence, number of discharges will be at least 9  
With 70 % of confidence, number of discharges will be at least 10

Josefina Dalla Via Monti  
@github.com/josedallavia

Figure F.1





# Bibliography

- [1] Balaji,R. and Brownlee, M., (2018). Bed Management Optimization. Infosys.
- [2] Boaden, R., Proudlove, N.C. and Wilson, M. (1999), An Exploratory Study of Bed Management, *Journal of Management in Medicine*, Vol. 13, No.4, pp. 234-250.
- [3] Bontempi G., Ben Taieb S., Le Borgne YA. (2013) Machine Learning Strategies for Time Series Forecasting. In: Aufaure MA., Zimányi E. (eds) Business Intelligence. eBISS 2012. Lecture Notes in Business Information Processing, vol 138. Springer, Berlin, Heidelberg.
- [4] Bradley, Andrew P. (1997) The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7), pp. 1145-1159.
- [5] Carter, J.C., Blackmore, E., Sutandar-Pinnock, K. and Woodside, D.B., 2004. Relapse in anorexia nervosa: a survival analysis. *Psychological medicine*, 34(4), p.671.
- [6] Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R.: Crisp-dm 1.0: Step-by-step data mining guide (2000), <http://www.crisp-dm.org>.
- [7] CRISP-DM, viewed 14 June 2020, <<http://www.datascience-pm.com/crisp-dm-2/>>
- [8] Dal Pozzolo, A., Caelen, O., Johnson, R.A. and Bontempi, G., (2015). Calibrating probability with undersampling for unbalanced classification, *2015 IEEE Symposium Series on Computational Intelligence*, pp. 159-166. IEEE.
- [9] Davis, G.E. and Lowell, W.E., (1999). Using artificial neural networks and the Gutenberg-Richter power law to "rightsize" a behavioral health care system. *American Journal of Medical Quality*, 14(5), pp.216-228
- [10] Di Piazza, A., Di Piazza, M.C. and Vitale, G., (2016). Solar and wind forecasting by NARX neural networks. *Renewable Energy and Environmental Sustainability*, 1, p.39.
- [11] Dietterich, T.G., (2000, June). Ensemble methods in machine learning. *International workshop on multiple classifier systems*, pp. 1-15. Springer, Berlin, Heidelberg.

- [12] Dumas M. B. (1985). Hospital bed utilization: an implemented simulation approach to adjusting and maintaining appropriate levels. *Health services research*, 20(1), 43–61.
- [13] Emmert-Streib, F. & Dehmer, M., 2019. Introduction to Survival Analysis in Practice. *Machine Learning and Knowledge Extraction*, 1(3), pp.1013–1038. Available at: <http://dx.doi.org/10.3390/make1030058>
- [14] ETL: What is and why it matters, viewed 14 June 2020, <[https://www.sas.com/en\\_us/insights/data-management/what-is-etl.html](https://www.sas.com/en_us/insights/data-management/what-is-etl.html)>
- [15] Ferri, C., Hernández-Orallo, J. and Modroiu, R., (2009). An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1), pp.27-38.
- [16] Green, J and Armstrong, D (1994), "The views of service providers", in Morrell D, Green J, Armstrong D, Bartholomew J, Gelder F, Jenkins C, Jankowski R, Mandalia S, Britten N, Shaw A and Savill R, Five essays on emergency pathways, Kings Fund Institute, for the Kings Fund Commission on the future of Acute Services in London, Kings Fund, London.
- [17] Hao, K., (2018). What is machine learning?. *MIT Technology review* [online]. Viewed 14 June 2020. Available from: <<https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/>>
- [18] J. Heaton,(2016). An empirical analysis of feature engineering for predictive modeling, *SoutheastCon 2016*, Norfolk, VA, , pp. 1-6, doi: 10.1109/SECON.2016.7506650.
- [19] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. and Liu, T.Y., (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, pp. 3146-3154.
- [20] Kelleher, J.D., Mac Namee, B. and D'arcy, A., (2015). *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press.
- [21] Koehrsen, W., (2018). A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning. *Towards data science* [online]. Viewed 14 June 2020. Available from: <<https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter/-optimization-for-machine-learning-b8172278050f>>
- [22] Krollner, B., Vanstone, B.J. and Finnie, G.R., (2010, April). Financial time series forecasting with machine learning techniques: a survey. In ESANN.
- [23] Kutafina, E., Bechtold, I., Kabino, K. et al.(2019). Recursive neural networks in hospital bed occupancy forecasting. *BMC Med Inform Decis Mak* 19, 39 (2019). <https://doi.org/10.1186/s12911-019-0776-1>

- 
- [24] Löschmann, L. & Smorodina, D., (2020). Deep Learning for Survival Analysis. Humboldt-Universität zu Berlin. Viewed 8 August 2020. Available from [https://humboldt-wi.github.io/blog/research/information\\_systems\\_1920/group2\\_survivalanalysis/](https://humboldt-wi.github.io/blog/research/information_systems_1920/group2_survivalanalysis/)
  - [25] Malkomes, G., Schaff, C. and Garnett, R., (2016). Bayesian optimization for automated model selection. *Advances in Neural Information Processing Systems*, pp. 2900-2908.
  - [26] Mekhaldi R.N., Caulier P., Chaabane S., Chraïbi A., Piechowiak S. (2020). Using Machine Learning Models to Predict the Length of Stay in a Hospital Setting. In: Rocha Á., Adeli H., Reis L., Costanzo S., Orovic I., Moreira F. (eds) Trends and Innovations in Information Systems and Technologies. WorldCIST 2020. *Advances in Intelligent Systems and Computing*, vol 1159. Springer, Cham
  - [27] Nguyen, J., Six, P., Antonioli, D., Glemain, P., Potel, G., Lombrail, P., Beux, P.L. (2005). A simple method to optimize hospital beds capacity. *International journal of medical informatics*, 74 1, p. 39-49.
  - [28] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., (2011). Scikit-learn: Machine learning in Python. *The Journal of machine Learning research*, 12, p.2825-2830.
  - [29] Perelman, L. and Barrett, E., (1997). The Mayfield handbook of technical and scientific writing. McGraw-Hill, Inc..
  - [30] Sá, C., Dismuke, C.E. and Guimarães, P., 2007. Survival analysis and competing risk models of hospital length of stay and discharge destination: the effect of distributional assumptions. *Health Services and Outcomes Research Methodology*, 7(3-4), p.109-124.
  - [31] Sekula, P., Dunant, A., Mockenhaupt, M., Naldi, L., Bavinck, J.N.B., Halevy, S., Kardaun, S., Sidoroff, A., Liss, Y., Schumacher, M. and Roujeau, J.C., 2013. Comprehensive survival analysis of a cohort of patients with Stevens–Johnson syndrome and toxic epidermal necrolysis. *Journal of Investigative Dermatology*, 133(5), p.1197-1204.
  - [32] Sitepu, S., Mawengkang, H. and Husein, I., (2018, January). Optimization model for capacity management and bed scheduling for hospital. *IOP Conference Series: Materials Science and Engineering*, Vol. 300, No. 1, p. 012016. IOP Publishing.
  - [33] Snoek, J., Larochelle, H. and Adams, R.P., (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, p. 2951-2959.
  - [34] Tom Fawcett, (2006). An introduction to ROC analysis. *Pattern Recognition Learning* 27, 8 (June 2006), 861–874. DOI: <https://doi.org/10.1016/j.patrec.2005.10.010>

- [35] Turgeman, L., May, J.H. and Sciulli, R., (2017). Insights from a machine learning model for predicting the hospital Length of Stay (LOS) at the time of admission. *Expert Systems with Applications*, 78, pp.376-385.
- [36] Walczak, S., Pofahl, W.E. and Scorpio, R.J., (1998, May). Predicting Hospital Length of Stay with Neural Networks. *FLAIRS conference*, pp. 333-337.
- [37] What is the CRISP-DM methodology?, viewed 14 June 2020, <<https://www.sv-europe.com/crisp-dm-methodology/>>
- [38] Wu, J., Chen, X.Y., Zhang, H., Xiong, L.D., Lei, H. and Deng, S.H., (2019). Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1), pp.26-40.
- [39] Y. Bengio, A. Courville, and P. Vincent, (2013). Representation learning: A review and new perspectives, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828.
- [40] Zheng, A. and Casari, A., (2018). *Feature engineering for machine learning: principles and techniques for data scientists*. O'Reilly Media, Inc.
- [41] Zupan, B., Demšar, J., Kattan, M.W., Beck, J.R. and Bratko, I., 2000. Machine learning for survival analysis: a case study on recurrence of prostate cancer. *Artificial intelligence in medicine*, 20(1), pp.59-75.