Machine Learning

Project Report

Team 50

MDSAA - DS

Authors:

Mateus Amaral (20230595) mbaptistaamaral@gmail.com

Duarte Mendes (20230494)

2023/2024 - 1º Semestre

Contents

Abstract

Context:

Hospital readmissions pose a significant challenge in healthcare management, reflecting both the lack of care given to patients, and the economic burden to national health services associated therewith. When a patient returns to the hospital after a short period of time (30days), not only does it indicate potential flaws in the initial treatment or discharge process, but it also raises concerns about the overall effectiveness of the healthcare system and the well-being of the patient.

Methods:

For our study, we used a database containing 71233 patients (53.7% female, 46.3% male, and 56.37% 65 years or older) to train our models, of which 11.16% were readmitted 30 days after being discharged and 34.93% after more than 30 days thereafter. To assess the quality of our predictions, we employed a test database containing 30530 patients, without their readmitted status. As labels for our data, we used the variables readmitted_binary (if a patient was readmitted in the last 30 days, yes or no) and multiclass (30days, 30days, No).

Results:

We discovered that patients who had been to the hospital several times before, who had their weight measured (diabetes indicator), who had a lot of diagnosis and who didn't have a healthcare insurance provider, where much more likely to be readmitted. The results were far from ideal, with the best predictions for the binary variable where achieved with Extreme Gradient Boosting (XGBoost), which yielded an f1 score of 0.364, being able to predict correctly 29% of the readmissions, while misclassifying 50% of the patients that actually were readmitted.

Conclusion:

The findings of our study underscore the complexity of factors influencing hospital readmissions. The significant correlation between previous hospital visits, diabetes indicators (such as weight measurement), number of diagnoses, and lack of healthcare insurance with higher readmission rates highlights key areas for intervention. Although the predictive accuracy of our models is extremely poor and can't be used for now in a clinical environment, it nevertheless is useful in that it points to the necessity for more personalized and comprehensive patient data tracking, so that forecasts can be improved.

Report Structure

The report is organized into four sections: main problems and solutions, binary and multiclass forecast and conclusion.

1. The first section contains:

   - **Main challenges in data preprocessing:** This subsection will address the complexities encountered during the initial stages of data handling, such as dealing with missing values (NaNs), strategies for imputing missing data, and the intricacies of feature engineering.
   - **Simplifying the data:** Here, we explore the application of feature elimination techniques, such as RFE and regularization as a method to enhance model performance by iteratively removing less significant features. This method eases interpretation and reduces overfitting.
   - **Data imbalance:** This part will foccus on the issues posed by imbalanced datasets, particularly how disproportionate representation of classes can skew model training and affect prediction accuracy. We will discuss the methods implemented to mitigate this imbalance and their impacts, like random under-/oversampling and simple sampling.
   - **Data Scaling:** The necessity and techniques of data scaling will be covered, outlining how standardizing or normalizing data can influence model training and performance, especially in algorithms sensitive to the scale of input features.
   - **Model Testing:** In the context of these problems (imbalance and scaling) we explain how the models were tested, as well as the use of cross validation to reduce overfitting and to make sure the models generalize well.

2. The following sections will be dedicated to specific aspects of binary and multiclass preprocessing, sampling, and the models used:

   - **Preprocessing for Binary and Multiclass Prediction:** This section will compare the unique preprocessing steps adopted for binary versus multiclass models.
   - **Models Used in Prediction:** A detailed discussion of the various models employed, including logistic regression, decision trees, random forests, and ensemble algorithms like XGBoost, along with their configuration, tuning, and performance metrics.
   - **Model Evaluation and Validation:** This subsection will elaborate on the techniques used for model evaluation, such as cross-validation, and the metrics employed to assess model performance, ensuring the robustness and reliability of the predictions.
   - **Interpretation of Model Outputs:** Here, we will interpret the results provided by the models, discussing their implications for healthcare strategies, and how they contribute to understanding and preventing hospital readmissions.

3. **Future Directions and Research Opportunities:** The report will conclude with a discussion on potential future research avenues, including the exploration of advanced modeling techniques, integration of additional data sources, and application in different healthcare contexts.

1 Introduction

1.1 Background and Significance

Hospital readmissions are a critical indicator of healthcare quality and effectiveness. With significant implications for patient well-being and healthcare economics, understanding and predicting these readmissions is of utmost importance. This report dives into the complexities of forescasting readmissions using advanced machine learning algorithms. By analyzing patient data and employing various mathematical models, we aim to uncover patterns and factors contributing to readmissions. The insights gained are not only crucial for improving healthcare outcomes but also for optimizing resource allocation and policy-making in healthcare systems. This study, therefore, serves as a crucial step towards enhancing patient care and reducing the burden of readmissions.

1.2 Objectives of the Study

The main goal of this project is to use various techniques of data pre-processing, manual feature engineering, and a combination of models, so as to be able to predict well enough if a patient is going to be readmitted, and if so, in the next 30 days or not. We aim to minimize as much as possible Type II errors, meaning we prioritize reducing false negatives in our predictions. In the context of hospital readmissions, a Type II error would mean incorrectly predicting that a patient would not be readmitted when, in fact, they are at risk. This is crucial as failing to identify at-risk patients can lead to a lack of necessary interventions and support, potentially resulting in adverse health outcomes. That is why the main metric used will be the f1 score, because it provides a balanced measure of the model's precision and recall. The f1 score is particularly useful in situations where an imbalance between false positives and false negatives carries significant consequences.

2 Data Preprocessing

Before we even started analysing the missing values in the dataset, we noticed that there was a non-variant feature, which was immediately dropped (country). There was another totally uncorrelated, that was soon after dropped as well (bpm).

2.1 Changing feature names for greater clarity

Feature names were changed in the beginning of the work, so as to make it easier to read them, as well as to rapidly be able to distinguish between a categorical variable (c_variable), numerical (n_...) or binary (c_...)

2.2 Strategy for High Missing Value Proportions

For features with a high proportion of missing values (over 40%), we adopted a strategic approach. We considered three options based on the nature and significance of each variable:

- **Dropping Variables:** Features with excessively high missing rates were dropped, as their scarcity of data rendered them unreliable for predictive modeling.
- **Creating Dummy Variables:** For certain variables, the non-absence of data was what mattered. In such cases, we created dummy variables to indicate whether the original variable was not missing.
- **Interpreting Non-Existence:** We also explored the meaning behind the absence of data in some variables (namely payer_code). This involved analyzing whether the non-existence of data could be indicative of specific conditions or scenarios.

2.3 Handling Low Missing Value Counts

Variables with lower rates of missing data were managed differently to preserve their utility:

- **Imputation using KNN or Random Forest Classifier:** For variables with minor missing data, we employed imputation techniques such as KNN (K-Nearest Neighbors) or a random forest classifier. This approach allowed us to infer missing values based on similar data points or patterns discerned from the data.
- **Imputation using patient id:** Since patients don't tend to change race, or main characteristics of a person, we used some features from the train dataset from patients that were registered multiple times to impute those features where they were missing.
- **Grouping Based on Target:** We grouped variables according to their relationship with the target variable, which helped understanding the impact of different categories on the prediction outcome. What we did was to see if the missing values provided meaningful information and that they couldn't be grouped into other categories, because if so they would rather be imputed.

2.4 Feature Selection

This step was greatly eased with the help of a function devised by us called var_report(), which greatly increased speed in feature selection and provided meaningful insight. It works by generating a visual representation of the data, as well as the deviation from the mean target (for the binary target) from a certain category. These deviations signal variance, which means information. The steps for feature selection were as follows:

- **In-Category Count Analysis:** Initially, we got the count of occurrences within each value of a categorical variable. A low count was considered a preliminary indicator of a less informative feature.
- **Interaction with Target Variable:** We then analyzed how different categories within a categorical variable interacted with the target variable. Categories with similar effects on the target variable and low occurrence

counts were grouped together. This approach helped in reducing the feature space while retaining important information.

- **One-Hot Encoding:** Post grouping, all categorical variables underwent one-hot encoding. This transformation was essential for converting categorical input variables into a form that could be more effectively utilized by the machine learning models.

2.5 Feature Engineering Techniques

Our approach to feature engineering was multi-faceted, involving the creation of new variables based on existing data:

- **Patient History Features:** Using patient IDs, we derived new features such as 'readmitted_before', 'number_of_visits_before', and 'number_of_readmissions_before' from the training dataset. These features aimed to capture each patient's historical interaction with the healthcare system, providing valuable context for the predictive models.
- **Target Mean Encoding:** We utilized a pseudo-target mean encoding to create certain variables like 'diabetes severity'. This method involved replacing a categorical variable with a scale of numbers or a group type (ex.: variable_lowReadmission_group) that is equivalent to the average target mean conditioned on that group, thereby capturing the variable's impact on the likelihood of readmission.
- **Summation of Variables:** In some instances, we combined variables to create new features. For example, summing different types of patient visits (inpatient, outpatient, ...) to form a composite indicator of healthcare utilization.

The combination of these feature selection and engineering techniques allowed us to refine our dataset into a more predictive and efficient form, setting the stage for the subsequent modeling phase.

2.6 Solutions for Data Imbalance

The dataset used in our study presented a significant challenge due to its high degree of imbalance. Data imbalance occurs when the outcome classes in the dataset are not represented equally, leading to models biased towards the majority class, often at the expense of the minority class. This bias can result in poor predictive performance, especially for the underrepresented class. Since the main goal of this project is to predict the minority class, data imbalance is very important to be addressed.

To address the data imablance, we used imblearn library's random over-/undersampling, which have their advantages and disadvantages: the oversampler can promote overfitting to the train data and takes more time to run. We mostly used our own sampler - simple/multiclass sampler - which upsamples the minority class by doing copies of the data in it, so that it has the same number of observations as the majority.

We also used the embedded sampling techniques that come available with ensemble classifiers, such as XGBoost and Random Forests, but we found them to perform a little worse than our own sampler, so we didn't use it for our final submission. (Check Figure 4 for the importance of sampling)

2.7 Data Scaling Techniques

In our analysis, the Min-Max scaler was the optimal choice for data scaling, enhancing the performance of various models. Naturally it wasn't needed for every model, namely in tree-based algorithms (TBA), as they aren't sensible to scale when doing a split. Robust and standard scaler models were also employed, but never in combination, because we mostly found that the TBAs were the best performing, as will be seen latter on.

2.8 Model testing

In our model testing, we utilized stratified cross-validation to ensure the reliability and generalizability of our results. The problem was that this method didn't allow us natively to scale and sample the data, so we had to create a function. The function start an instanced of stratified KFold cross validation, then loops through the indices assigned to train and validation for each instance and scales and samples the data if requested. Then the model is fitted and the score is calculated based on the scoring method. After that the score is appended to a list and the process continues until it cycles through. The mean score is displayed.

3 Binary Predictions

3.1 Preprocessing for Binary Classification

In our binary classification work, we aimed to figure out which patients were more likely to be readmitted within 30 days. To simplify the variables, we used a good method: grouping variables by how they relate to readmission chances. This helped turn categories into numbers and scales (low, medium and high probability), making them easier for our models to digest. This, of course, isn't possible to do in the multiclass variable.

Examples of this type of processing are the 'n_diabetes' (and 'n_severity), which related to the degree of acuteness of the diabetes diagnosis and 'c_payerGroup' which showed if a certain payer code was associated with higher readmissions. These features were useful for our models, with the n_severity and n_diabetes ranking very high in the decision tree feature importances (2nd and 14th)

3.2 Dedicated RFE

3.2.1 Model used

The Recursive Feature Elimination was simple to do in the binary variable: We used the extreme gradient boosting classifier (XGBC) as it didn't require us to scale (native for tree algorithms) or sample the data (with the scale_pos_weight parameter), as well as because of its ability to work with missing values, which in the beginning was crucial.

3.2.2 Four different datasets

After performing the RFE, we created 4 different dataframes based on the feature ranking dataframe:

1. **Train:** included all the 71 features from the data preprocessing
2. **Train_light:** was composed of 45 features. It didn't contain highly specific variables, like all the payer codes or features that were manually considered less important, such as race or admsission source.
3. **Train_25:** Was a product of the RFE, where the top 25 features were selected
4. **Train_ultra_light:** The same as the train 25, but only 12 features got selected

3.3 Models Used

For this task, we explored a variety of models, each with its own strengths and ideal use cases. The goal was to identify the best performing and time-efficient approach to make the predictions. The models ranged from simpler ones like logistic regression to more complex ensemble and neural network methods. Here are the models we considered using:

- **Logistic Regression:** was used as a baseline for the model exploration, especially to test changes in data and how they affected the score, as it is easy to implement (score doesn't vary widely with hyperparameters), interpretable and not time consuming. Nevertheless, it fails to capture non-linear relationships and thus it didn't achieve the best results.
- **Multilayer Perceptron Classifier (MPLC):** This neural network-based model is excellent for capturing complex patterns in data. It excels in scenarios where relationships between variables are intricate and not easily modeled by traditional algorithms. However, it requires careful tuning of parameters and architecture, and it can be prone to overfitting. Additionally, due to its complexity, MLPC is less interpretable than simpler models like logistic regression and can be more demanding in terms of computational resources.
- **Ensemble Methods (RandomForest, AdaBoost, GradientBoosting, XGBC):** were the most used throughout our work. They are great for robust predictions, as they rely on multiple weak models to make decisions. They are quite robust to overfitting, don't need a scaler and handle complex relationships in data.
- **SVC:** support vector classifiers are particularly effective in high-dimensional spaces. We tried using them, but soon realized they took too long to train and that the results were not good enough.

3.4 Results

Let's now delve into what the data revealed, which models and dataframes stood out in our analysis, and how the models were tuned. We will first present data from vanilla models (default parameters) and then talk about the more tailored models for our data, that were perfected with a random search algorithm.

3.4.1 The dataframes

The best performing dataframe was the train_25, which had lower complexity, was faster to train and achieved the best average scores. Figure 1 shows the dataframes ranked by average time to train and average score on validation data, with the error bars included, which correspond to the standard deviation of the scores obtained during cross validation.

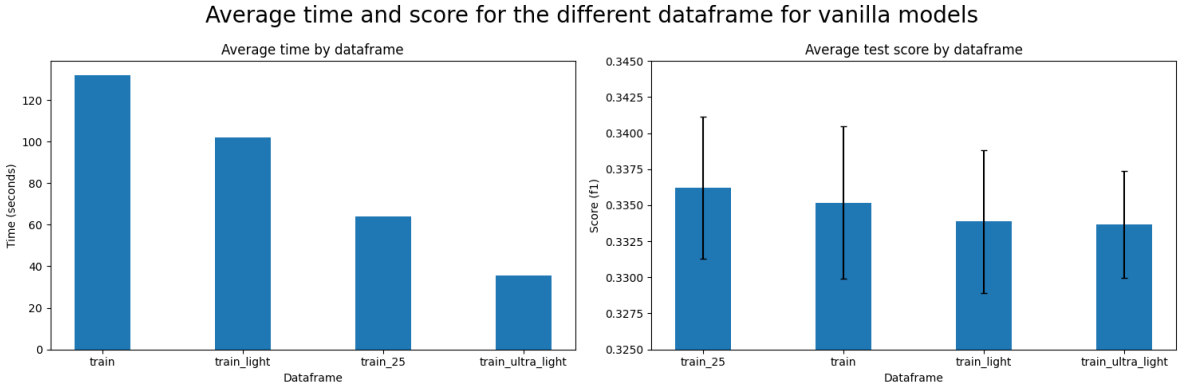

Figure 1: Comparison of dataframes by training time and validation score.

3.4.2 The models

In our model comparison, the Random Forest emerged as the best performer among the vanilla models, closely followed by XGBoost. ADA Boost, however, showed the weakest performance. Logistic Regression displayed the highest standard deviation, indicating variability in its performance across different runs. Time-wise, the MLPClassifier took the longest, while XGBoost was the most efficient. Figure 2 below illustrates these findings, including performance and training time for each model.

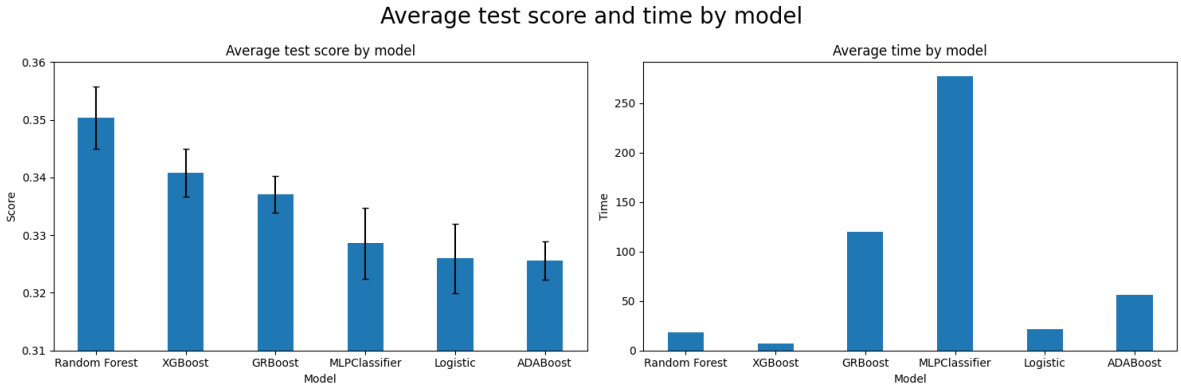

Figure 2: Model performance comparison by f1 score and training time.

3.4.3 Model tuning

The results shown earlier can be made better with the help of hyperparameter tuning. For that we did a random search that cycled through a list of random values for each parameter, followed by manual parameter optimization. That means we took the best randomly selected arguments, and nudged them up and down, one by one, using a greedy approach, which is to say that we only accepted the change if there was an improvement in the score. Figure 3 shows for XGBoost how swapping individual parameters can have a large impact in training time and validation score.

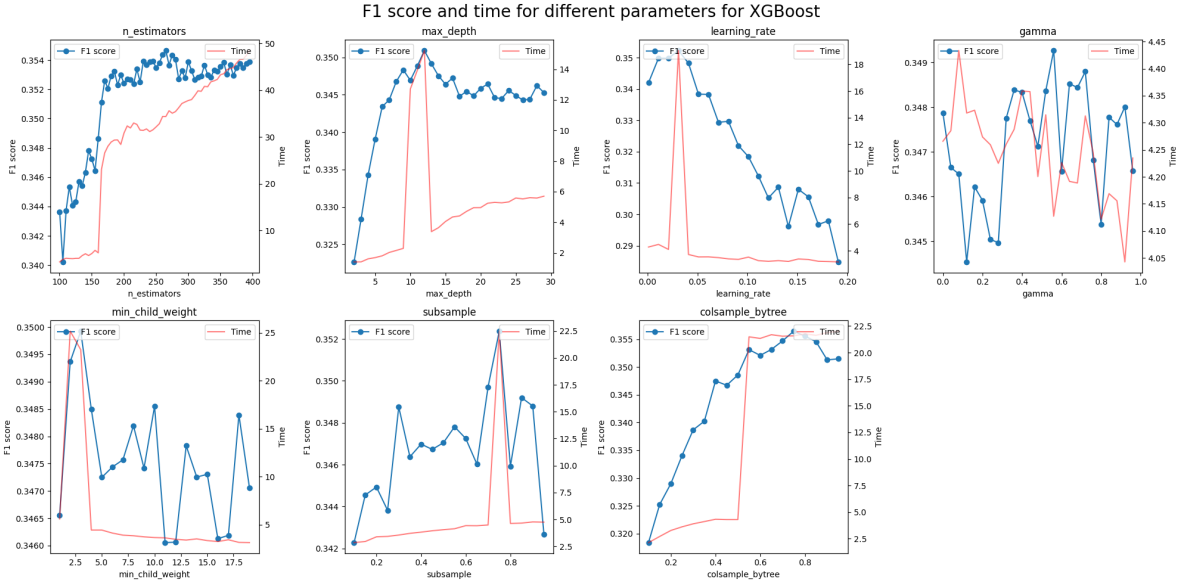

Figure 3: Single parameter changes impact on model performance for XGBoost

3.4.4 Best results

Overall, the best results were obtained with the XGBoost classifier trained on the full train dataset. We obtained an f1 score of 0.364, correctly predicting 29% of the readmissions, while not correctly classifying about a half of the readmitted patients. The random forest classifier achieved an f1 score of 0.359, and the worst performing algorithm was the logistic regression with 0.327. (Check fig. 5 in the appendix for more information on the models and datasets)

For the XGBoost and a validation set of 21370 patients, we correctly classified as not readmitted 15982 (84.2% of not readmitted) and misclassified in total 4189 patients (19.6% of total set); Naturally, the most crucial aspect of our work is predicting readmissions. The model successfully predicted readmissions for 4202 patients, correctly identifying 1199 of these cases (29% of the patients classified as readmitted).

3.4.5 Most important features

It is important to note that feature importances vary widely with model, parameters and random_seed. For the best model, we checked which features were the most important and these included the number of times a patient had

visited the hospital, the severity of their diagnosis, the length of their hospital stay, their age, and the number of medications received. (*vide* figure 6 appendix)

There were also other features that helped determine if a patient wasn't going to be readmitted, like if a patient had died or admitted to a hospice, but these variables don't contain relevant information for our specific objective.

4 Multiclass Predictions

In the multiclass classification part, we expanded our scope beyond the 30-day readmission prediction, aiming to categorize patients into multiple readmission timeframes (>30days,<30days and not readmitted). Unlike the binary classification, where we could simplify variables by grouping them based on their readmission probability, the multiclass approach required a more nuanced handling of variables. This complexity stems from the need to distinguish between multiple categories of readmission times, each with its distinct characteristics, thus rendering the average useless.

This shift to multiclass prediction presented unique challenges but also offered a richer, more detailed understanding of patient readmission dynamics. It allowed us to explore a broader spectrum of patient healthcare trajectories and their implications on readmission risks.

4.1 Preprocessing for Multiclass Classification

The multiclass data is highly skewed towards the No category (53.9%), followed by the >30days category (34.9%), and finally the <30days (11.16%). This means that what we want to retain mostly from the data is features skewed towards the </>30days, or features that are so skewed to the No category that they immediately exclude a readmission.

4.1.1 Function variable report

To figure out which variables were useful in our study, we created a function that could handle multiclass data. Here's how it works: First, the report shows how often each variable appears in our training and testing data. For the training data, it then calculates how much each category of a variable differs from the average for each possible readmission time frame. Then it also displays this data in charts for easier understanding.

This step helps us see if a particular feature, like a patient's race, is important or not. For instance, let's say the 'Caucasian' category in the 'race' variable shows a significant difference (or skew) from the average for patients readmitted in less than 30 days, and this category also appears frequently. In that case, the 'Caucasian' category is considered to have a lot of useful information, making it worth including in our analysis.

4.1.2 Dealing with missing values

More missing values were imputed here, as opposed to the works with the binary features, since it was harder to group together features, for instance the variables source and disposition. Most variables were simply one-hot encoded and

thus not grouped at all, which led to a huge increase in data dimensional (145 vs 71 native features, for multiclass and binary respectively).

Grouping for the multiclass categorical variables had to be done manually and in a time-consuming manner, which meant that we had to look at the anomaly (distance from target mean) profile for each of the values the category could take and find similarities.

4.1.3 Feature engineering

For this problem, we created much less features than in the binary one, as target mean encoding was out of reach. Variables like n_diabetes, n_severity were not possible, so instead we just created a categorical feature for the diabetes ICD9 code (starting on 250.**), and tried to group together some of them, before one hot encoding.

Contrary to what the reader may believe, feature engineering actually reduces the number of features by combining them and grouping them into more informative ones. As such, the almost total absence thereof for the multiclass problem was the main culprit of the high feature count.

4.1.4 Perliminary Selection

The medications feature that came native to the dataframe wasn't usable as it was, because it was a in list form, therefore it had to be broken down into several binary variables, most of which had extremely low minority counts. For that reason, we performed a chi-square independence test together with an examination of the minority counts. If they were independent, we'd drop them, and if the count was too low, the same. This led us to drop 9 of the 17 different medications.

4.2 RFE and regularization

Feature elimination in the multiclass problem was even more important than in the binary one, as there were more than double the features (for the same information). Since we were going to remove many more features, we wanted to use a robust feature selection method, and for that we used regularization (lasso) together with RFE.

4.2.1 RFE

We used RFE with cross validation to get more solid results, and for the models we chose the logistic regression (multinomial) and an XGBoost classifier. The XGBoost algorithm chose 110 features as important, while the linear regression chose only 8. This was important to obtain some kind of ranking.

4.2.2 Regularization

For the regularization, we couldn't use the lasso model directly, as it wouldn't work natively for classification problems. As such, we used the logistic regression with a l1 penalty. Then we iteratively chose different regularization terms - C (higher means less regularization) - and finally stored the f1 scores. We noticed that after C=0.0025, the score

remained stable, while the number of features gradually increased. So, we chose a value that was close to that (0.005), to select 26 features. (Check figure 7 to see the coefficients of each feature that was selected)

4.2.3 Ensemble

To perform the final selection, we scaled each of the ranking to be between 0 and 100, summed them, and then re-ranked each variable, to obtain the final ranking. We thought 25 to be a good enough number of features, so we just dropped the features that had a ranking above 25.

4.2.4 Results

We were left with a dataframe containing approximately the same information as the original one, but much faster to train and less prone to overfitting. The most important variables were the number of times a patient had been admitted, if a patient had died and if its weight had been measured.

4.3 Models Used and Results

We chose 4 models to work with our data: Logistic Regression, XGBoost, Random Forest and MPLClassifier. Advantages and disadvantages were already stated so lets proceed to the results.

The best performing model was the Random Forest Classifier, with an f1 (weighted) score of 0.635, followed by XGBoost, Logistic Regression and finally MPL Classifier. The classification report de RFClassifier revealed a striking variance across different readmission categories:

- For patients not readmitted, the model showed a decent performance with a precision of 0.75 and a recall of 0.83, leading to an F1 score of 0.79.
- For patients readmitted after more than 30 days after their visit, the precision dropped to 0.58 and recall to 0.48, resulting in a lower F1 score of 0.53.
- The most challenging predictions were for patients readmitted within 30 days, where both precision and recall were only about 0.27, leading to an F1 score of just 0.27.

These numbers illustrate the varying degrees of difficulty the models faced in predicting different readmission scenarios. While the models were relatively good at identifying patients not likely to be readmitted, their performance dropped significantly when it came to predicting readmissions, especially within the critical 30-day window.

5 Discussion

5.1 Analysis of Model Performance

In our analysis, the XGBoost and Random Forest models, both tree-based algorithms, outperformed the others. They were more effective in handling the complexities of our dataset, offering better predictive accuracy. However,

it's important to note that despite their relative success, the recall and consequently the F1 score for these models was quite low. This is a critical observation, especially since our primary objective was to predict readmissions accurately. The weighted F1 score in the multiclass setting doesn't capture the full picture, as it tends to be inflated by the model's performance on the majority class.

Another intriguing finding was the wide variation in feature importances across different models and model parameters (including the random seed). This inconsistency suggests that many features might not be as informative as initially thought, limiting our ability to draw concrete conclusions about their impact on readmission predictions.

As for the MLP Classifier, its performance was notably poor and inefficient. Despite its theoretical capability to capture complex relationships within the data, in practice, it was slow and showed limited effectiveness in our specific context. This result was somewhat surprising and disappointing, considering the potential of neural network models in complex classification tasks.

5.2 Challenges and Limitations

Our study faced several challenges and limitations. The huge data imbalance definitely hindered us from making real-world usable predictions. Additionally, the significant discrepancies in feature importances across models indicate a potential limitation in the predictive power of our features.

6 Conclusion

In conclusion, our study made significant strides in predicting hospital readmissions using various modeling techniques. While the Random Forest and XGBoost models showed the most promise, the overall challenge of accurately predicting readmissions within a 30-day window remains. The variability in feature importance across models and the limitations observed in MLP Classifier's performance highlight the complexities involved in healthcare data analysis. These insights pave the way for future research to refine predictive models further, ultimately trying to enhance patient care and healthcare resource management.

6.1 Future Research Directions

We think that in the future, in order to achieve great results, we need more data regarding the patient's history, for instance incorporating unstructured data like clinical notes (through natural language processing techniques).

Another possible direction for future research is the exploration of personalized medicine approaches. Developing models that account for individual patient characteristics, lifestyle factors, and genetic information could lead to more tailored and effective prediction of readmission risks. This personalization would not only improve model accuracy but also enhance patient care by supporting more targeted interventions and preventive measures.

7 References

8 Charts

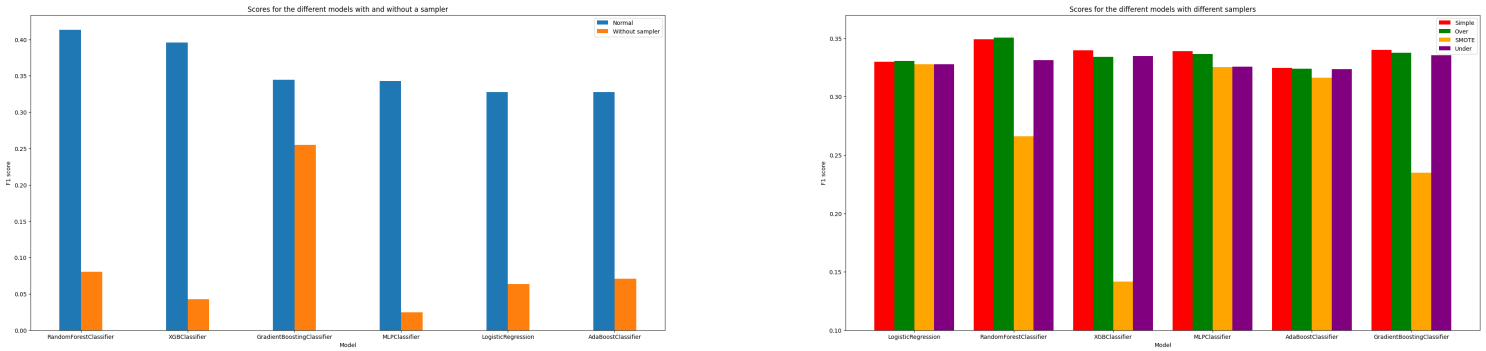

Figure 4: Left: F1 score with and without a sampler. Right: F1 scores for different samplers.

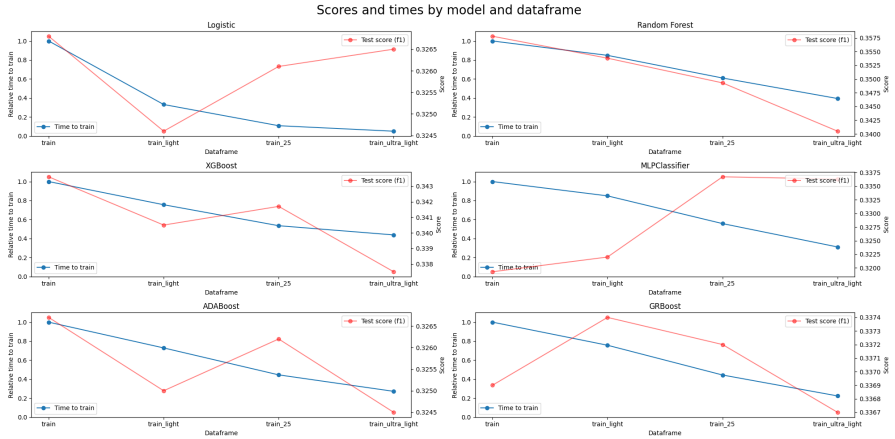

Figure 5: Scores and times of fit for every model and all the datasets (binary)

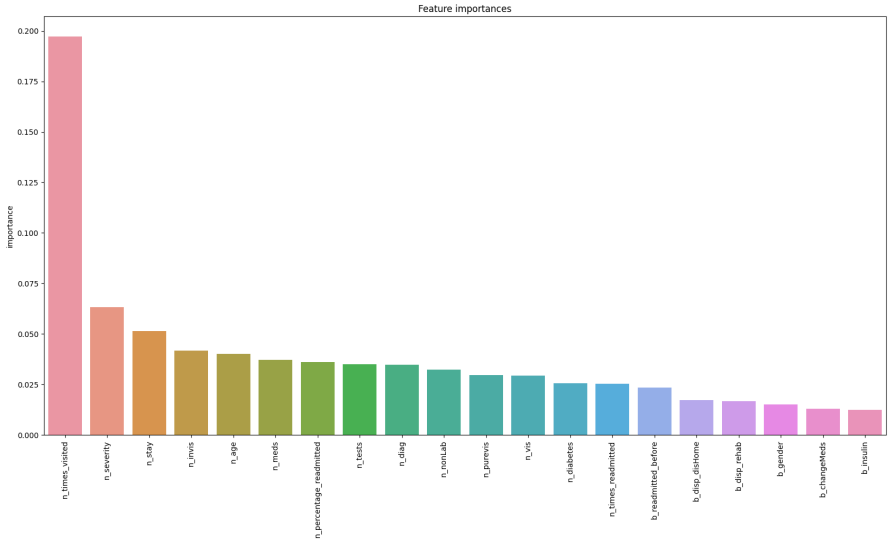

Figure 6: Feature importances for the random forest classifier (binary)

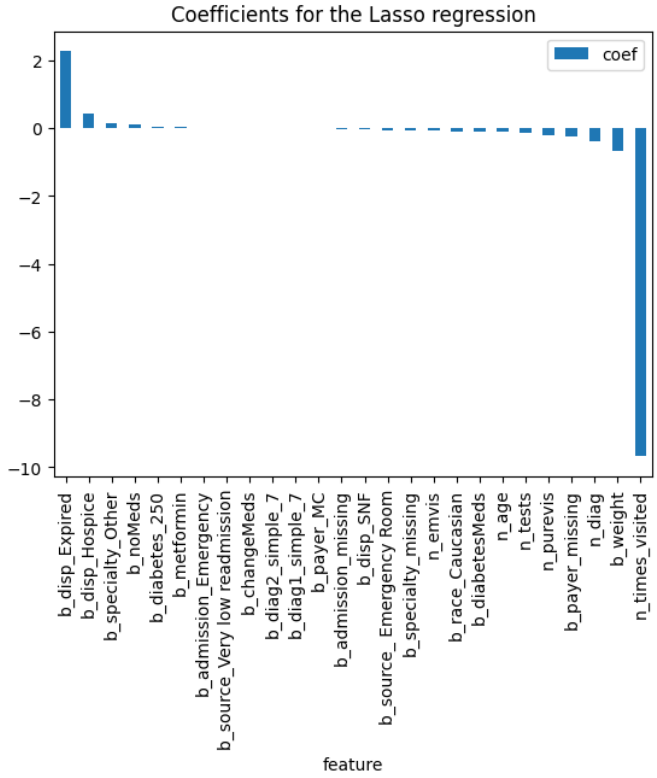

Figure 7: Lasso regularization - feature importances (multiclass)