READY TO BE DISCHARGED: EXAMINING HOSPITAL READMISSIONS

Machine-Learning Project

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Diogo Reis, 20230481

João Machado, 20230426

David Gustavo Guarín, 20230602

Diogo Almeida, 20230737

Flávio Ivo, 20230571

2023/2024

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

ABSTRACT

If hospitals and healthcare centres, could predict correctly the readmissions of patients it would change drastically the healthcare system as we know it, as this could give enough time to the hospital to prevent many kinds of emergencies before they happen. In other words, the impact of good predictions could lead the healthcare system to be well-prepared in different domains such as: professional scalings, resources quantity, appointment planning and others. To address this problem, and facing the issue of an imbalanced dataset we performed various data preprocessing and feature selection techniques to have the best dataset possible to develop Machine-Learning models to evaluate two different classification problems: binary and multiclass. In binary, the goal is to predict if a patient will be readmitted within 30 days before being discharged, this is the same in multiclass, though in this case we also check if the individual will be readmitted after 30 days, capturing the timeframe of patient's readmissions. Finally, our results reveal that our best model is Stacking, which uses Naive Bayes and Gradient Boosting with a meta-model of Gradient Boosting, with an F1 score of 0.287 for our binary target, and for our multiclass target the best model is Gradient Boosting with a 0.552 F1 score. In conclusion, to get a better outcome would be necessary a dataset with more consistent information and a more balanced target variable. Using oversampling could improve the score, but it needs a high computational cost.

Contents

1. Introduction

In the healthcare sector, it's crucial that we know of the patient's readmissions done within a certain period of time after their discharge. Machine Learning is a great tool to address this problem because it can help to identify potential problems and holes in the industry and provide the best solutions based on the different metrics and outputs given by the many models applied to the data. This is done mainly by predicting the patient readmissions so that patient care and substantial cost savings can be improved.

To achieve this task, we did multiple steps so that we could get to the best possible conclusion, given the data, of trying to predict the readmissions, both on a binary problem, between the patients being readmitted or not and even a multiclass scenario where we have more data regarding the exact timeframe where that same patient was readmitted. With that in mind, we developed this work from the beginning by checking the data and correcting its inconsistencies, to transforming variables, dealing with outliers, scaling, encoding and applying multiple steps of feature engineering, to finally developing multiple different models, both individual and ensembles, such as Logistic Regression, Naive Bayes, KNN, MLP's, Decision Trees, Random Forests, Bagging Classifiers, Boosting's, Stacking´s and even a Voting Classifier.

This report aims to address these points in detail, backed up by the outputs of the code done, while explaining the full thought process that we had during the development of this work, describing and referencing all this using the main documentations from *sklearn* libraries and some materials from both theoretical and practical sessions, while at the end also concluding a solution for both future studies and possible improvements to the work done here.

2. Data Exploration and Preprocessing

2.1 Data Exploration

First, we began by exploring and understanding the compiled data and its different features. By taking a closer look we found features with missing values or gaps. Typically, these gaps are denoted as "NaN" in numerical datasets or as "null" or "NA" in non-numerical datasets *[1]*.

Afterwards, we assessed the percentage of missing values in each feature to understand their relevance. In the pursuit of data cleaning, we addressed duplicate values. These duplicates, if not properly managed, can lead to overfitting, as the model is learning from the same information twice. Lastly, we verified the presence of readmitted patients which introduced important insights.

Following this, our attention turned to the exploration of descriptive statistics [Table 1], where we found some inconsistencies in some features: the presence of '?' values, variables with high and low dimensionality and confirmed the existence of many "NaN" values.

2.2 Data Preprocessing

2.2.1 Features Drop

Initially, we began by eliminating certain features from the data that we considered not necessary, starting with "country" since it contains one unique value: 'USA'. Then we proceeded to drop 'weight' and 'payer_code' due to having more than half '?' values.

2.2.2 Missing Values

Following this, we continue handling missing values by employing various strategies. We used the mode to replace the 'Unknown/Invalid' entries on the "Gender" column. We realized that replacing "NaN" values in patients that had more than one admission with the last value recorded on any of the admissions done by those same individuals, could be more insightful, so we addressed missing values with that technique in specific columns, such as "age", "race", '"admission_type", "discharge_disposition", "admission_source". Continuing with our data pre-processing, we extended this imputation technique to replace '?' values in columns like "race," "primary_diagnosis," "secondary_diagnosis," and "additional_diagnosis". Lastly, we addressed ´?' values for non-readmitted patients by replacing them with the mode, extending this approach to missing values present on "race" and "age." For columns such as "admission_type," "discharge_disposition," and "admission_source," we opted to replace missing values with the label 'not_saved'.

2.2.3 Categorical Variables Aggregation

We observed that certain categorical variables had multiple distinct values, indicating a diversity of labels. That being said, we opted to group them in a way to improve the overall effectiveness of the process because there was a huge difference in size between variables. In the case of "Discharge Disposition," we condensed the categories into a binary variable, distinguishing between those discharged home and those not, changing the name of the variable to "discharged_home_binary". For "admission_type," we retained specific labels such as 'Emergency', 'Elective' and 'Urgent', while grouping the remaining types as "Other." Similarly, regarding "admission_source," we preserved categories like 'Emergency Room' and 'Physician Referral', grouping all other sources into the collective category of 'Other'. In the context of "medication," we renamed it to "Medicated Status", effectively splitting it into two categories: 'Medicated' and 'Not Medicated'. This simplification organizes the information for more meaningful analysis. In order to

represent "age" more intuitively, we categorized between different ranges such as: [0-20] for ‘child’, [20-60] for ‘adult’, and [60-100] for ‘elderly’. For the "a1c_test_result," we applied the following transformations: ‘>7’ and ‘>8’ were labelled as ‘not normal’ and ‘Norm’ retained its designation as ‘Norm’, and any “NaN” was categorized as ‘NotMeasured’. When addressing the "glucose_test_result," we used a strategy which designates ‘>200’ and ‘>300’ as ‘NotNormal’, because having 200 or more in the glucose result is a tendency for diabetes[9] and ‘Norm’ as ‘Normal’, and any “NaN” are denoted as ‘NotMeasured’.

2.2.4 Visualizations

We performed various visualizations of the data, since those are powerful tools for extracting meaningful insights and guiding the overall data analysis process [2]. To be able to do these visualizations we had to split the features between metric and non-metric. After that, we plotted the Spearman rank-order correlation coefficient between the metric features to be able to quantify and summarize the relationships between multiple variables in a dataset [3].

Another method that we used was plotting histograms for each variable, they provide a graphical representation of the frequency or probability distribution of the data within specific intervals. Finally, we did box-plots that are used for visualizing the distribution, central tendency, and spread of a dataset, and are also used to highlight the presence of outliers. Combining both methods we decided on what values to consider outliers. The results of those three visualizations are on [Table 2], [Table 3], [Table 4], and they will later be treated after normalizing the data.

2.2.5 Feature Engineering

In this section, to extract the highest amount of information as possible from the dataset, we have created two new variables that correlate some features. The first one comes from the idea that we have patients who have been readmitted more than once, so we counted the number of encounters that each patient has done by grouping ‘patient_id’ with ‘encounter_id’. This new feature gives information about how many times a patient has been admitted to the hospital. Then we dropped the features that originate this new one since we no longer needed to know who the patient is nor use the ‘encounter_id’. Then we created a variable that correlates three features that are measured by the duration of each encounter: ‘procedures’. This one was created by summing ‘number_lab_tests’, ‘non_lab_procedures’ and ‘number_of_medications’. Again, we dropped the features that gave origin to the new one.

Still in this section, due to the high dimensionality of these features, we group diagnosis from ‘primary_diagnosis’, ‘secondary_diagnosis’ and ‘additional_diagnosis’ using the “International Classification of Diseases, Ninth Revision, Clinical Modification – ICD9-CM” [4] which can be checked on [Table 5].

For the next steps, we divide the dataset into numerical and categorical features.

2.2.6 Data Encoding

Since our models only accept numeric values, we proceeded to the categorical features encoding. We use the Target Encoding that encodes each category using an estimator, based on a statistical method: a shrunk estimate of the average target values for observations belonging to the category [5]. We explain this in more detail in the background section.

2.2.7 Data Scaling

Scaling techniques allow numerical features to fit all into a certain range while keeping the differences between values [8, p.87]. Not implementing those techniques could lead to biased results. We perform tests on three different approaches, assessing the accuracy and F1 on an MLP model with all default parameters to check which one gives us the highest performance. We used undersampling in order to deal with the imbalanced data set.

First, we tried Robust Scaler which scales the data using the quantile method, being a good approach if the dataset contains outliers when removing them isn't an option. Then we tried the simplest one which is MinMaxScaler and we ranged between 0 and 1. We also tested StandardScaler which removes the mean and divides each value by the standard deviation. The one that gave the better performance was MinMaxScaler, so we decided to use it to normalize our numerical data.

2.2.8 Outlier Treatment

Performing outlier treatment is essential in preparing the dataset for machine learning models. Treatment methods may involve transforming, imputing, or removing outliers, depending on the dataset and analytical objectives.

In our case, since we couldn't remove dimensions from our dataset, we opted to replace values considered outliers with "NaN." This allowed us to use later the KNN Imputer, a technique that handles missing values by considering the values of their k-nearest-neighbours. The imputation is based on the similarity of the missing data point to its neighbours in the feature space [6]. We used our visualizations insights and checked the new scaled values to make sure that we were replacing the correct values.

3. Binary and Multiclass Classification

3.1 Feature Selection

In this section, we applied filter, wrapper and embedded techniques to find the smallest subset of features that maximizes model performance for predicting the target variable.

On filter methods, which use statistical measures to assess the relationship between each input variable and the target, we started by checking the variance of each numerical variable and concluded that there aren't any variables with 0 variance. Then, we applied Spearman correlation to assess the relationship between numerical variables. After that, we applied ANOVA, also for numerical features. On this statistical test, the variable is more important as the F1 is higher and the p-value is lower (< 0.05).

On the filter methods for categorical features, we used: Mutual Information (explained in further detail in the background section) and Chi-Squared3 which measures the importance of each categorical independent feature in the target. As we had very low values on mutual information we decided to apply a threshold of 0.005 and select the features that had equal or higher values.

We also applied Wrapper methods, where different combinations of features were tested and evaluated. We used the RFE algorithm with a Logistic Regression as a predictor in order to assess which set of variables achieved the highest F1 score possible.

On the Embedded Methods we tried to find the best set of features that contributed the most to the model while the model was being generated. We applied Lasso regression which is a model to solve a linear optimization problem of minimization. Lasso picked all the numerical variables and didn´t discard anyone [7]. We also performed a Decision Tree test to measure feature importance both for categorical and numerical features where we chose the features that had 5% or more of feature importance.

In the end, we compile the results in two tables: one for numerical and the other for categorical variables. Since we performed 5 tests for numerical features we chose to keep only the ones that passed at least 4 tests, which meant they should be kept. Similarly, as we used three tests to assess the feature importance of our numerical features, we chose to utilise 2 tests to decide which categorical features we keep.

The results for binary can be found in [Table 6], [Table 7], [Table 10] and for multiclass in [Table 8], [Table 9], [Table 11].

3. 2 Modelling

Before creating our models, we first had to address an important issue which is our imbalanced dataset. Therefore, for the purpose of balancing our data, we decided to use random undersampling on our training data. This is a technique that randomly removes instances or subset data points from the majority class (denoted "No" in readmitted_binary), generating an equal number of points between readmitted and not readmitted patients [26], we go further into detail in the background section. Although there exist other balancing techniques such as SMOTE and Random Oversampling, we determined this to be the best alternative as the other techniques increase data points from the minority class, we also talk about it more in depth in the background section. Consequently, based on the limited time and the lack of computational power we decided on the ladder. For our Multiclass, we also decided to use undersampling, as there is also an imbalance in our target data. Even though the imbalanced ratio is not as big as with the binary data, there is still a significant difference between classes, leaving us with the conclusion of using undersampling.

Having our dataset ready, we proceeded to create and run different models for both the binary and multiclass, testing and comparing them, to decide which algorithm would give us the best forecast of patient readmissions, a crucial outcome in the healthcare section. The models that we used are: Logistic Regression, Naive Bayes, KNN Classifier, MLP Classifier, and Decision trees. For testing our scores both on binary and multiclass datasets, we decided to use primarily the F1 score. We decided to check on the F1 score as it combines precision and recall, giving information on the class performance rather than the overall like with accuracy. In other words, for this project, we must focus on the false negatives (FN) and false positives (FP). If we get more FN the health department won't be ready to accept a person who needs readmission, or if we have more FP then there will be a lot of time wasted that could be used for other important scenarios. Furthermore, we chose F1 scores as they are better for imbalanced data such as this one, while accuracy is more for balanced data or ideal datasets.[27]

As previously mentioned we decided to use F1 scores for the multiclass target, however, it is important to clarify that this dataset has more than one target value, therefore we cannot use a normal F1 Score technique, instead we used a weighted F1 Score. This means that we will calculate the metrics for each label, and then find their average 'weighted' by the number of true instances, taking into account the imbalanced data. We used this technique instead of 'micro' and 'macro' because, for micro is not the best option for imbalanced datasets as it uses the sum of the overall true positives (TP), FN and FP, giving us more of an overall accuracy. Although macro can be used for an imbalanced dataset, it does not take into account the weights on each label, which is important to our project as we need to understand our FP and FN, for the hospital to make a decision.

Then, we proceeded to test different models. Our first model was a **Logistic Regression** [10]**,** which is a statistical model that is used for classification and predictive analysis. It estimates the probability with a logistic function of an event occurring, in this case, if the person will get readmitted or not. Next, we used **Naïve Bayes** [11]**,** which is a supervised classification algorithm that uses Bayes theorem as a probabilistic classifier, using conditional probabilities to predict an outcome. This algorithm assumes that the feature inputs are independent of other features allowing it to make quick predictions. Our following model is the **KNN Classifier** [12]**,** a supervised learning classifier, which uses distance or proximity to make classifications. In other words, you have a data point whose class is assigned based on the majority of the labels its closest neighbours also are. The algorithm calculates this distance by using the Minkwosky, Euclidean or Manhattan distances. Continuing, we decided to use the **MLP Classifier** [13], not only because it is good for large amounts of input data but also makes quick predictions. MLP also known as multi-layer-perceptron, is an artificial neural network that learns from inputs and as a result, generates outputs depending on the inputs given. Imagine a multilayer of numerous input nodes that are connected to multiple non-linear layers, called hidden layers, then these nodes connect to the output nodes for which this output is calculated based on the weight given to each input. We also used **Decision**

trees [14] which is a supervised algorithm that uses a set of simple decision rules inferred from the data features, to classify the data. Saying that the deeper the tree the more complex the decision rules.

Wanting to improve even further our results, we decided to implement different ensemble models which are: Bagging, Boosting and Stacking. These models combine numerous algorithms into one with the purpose of getting better outputs.

Starting with **Bagging**, this is an ensemble process that uses multiple weak learners to generate predictions from input data and then it averages or obtains the max values from the overall prediction of each model [28]. For bagging we used two classifiers, **RandomForest** [15] and **BaggingClassifier** [16].

Contrary to bagging, **Boosting** uses sequential training, meaning that instead of averaging the results of multiple models, it uses only one model at the time to generate a prediction. It then uses the data that was predicted wrong as an input for a new algorithm. This goes on multiple times until it finally converges into the best result [28]. For Bagging we used GradientBoosting and AdaBoosting, Where **Adaboosting** [17] uses weak learners and it assigns more weight to the wrong predicted instance from these models and decreases the weight to the right predictions, with these weights it trains a new stronger model [29]. Meanwhile, for **GradientBoosting** [18] the weak learners work on the same sample distribution, and instead of assigning weight, it uses the gradient descent optimization process in the training model to minimise the error, which is calculated by the loss function [29].

Now for our next ensemble technique: **Stacking** [19] combines different heterogeneous models into one, meaning it generates predictions from different algorithms, averaging the result and using these outputs to create a new training set that is used for a meta-model that is in charge of creating the last predictions. For this, we try three different combinations, Logistic Regression and Gradient Boosting with Logistic Regression being the meta-model, Gradient Boosting and Naive Bayes with meta-model Gradient boosting, and lastly Logistic Regression with Random Forest with Gradient Boosting with a meta-model of Random Forest. We decided to use these models because they generated the best F1 scores individually. For Multiclass, we used the same combination of Logistic Regression with Gradient Boosting with the default meta-mode Logistic Regression, but then we implemented MLP and Gradient Boosting having the Gradient Boosting as the meta-model.

Our last ensemble, which is similar to Bagging is **Voting** [20], which we go into further detail in the background section.

3.3 Assessment

As previously mentioned we used feature selection across all models, to increase the model's focus and effectiveness. Following *sklearn's* guidelines, we employed grid-search carefully to optimise the parameters of most models, while keeping in mind computational cost. We tried to contrast the performance on both the training and validation sets and used accuracy and F1 scores as the main indicators for assessment. This methodology not only assessed the efficiency but also thoroughly checked for overfitting, guaranteeing that our chosen models were robust and optimal for practical implementation.

Firstly, we used the **Logistic Regression** model, where we obtained different results on the many metrics we chose to display [Table 20] [Table 21], both on the binary and multiclass models. Since this was our first attempt, we applied a simple model using the undersampled data, but we didn't achieve impressive results. On binary, we saw high accuracy but difficulty in predicting the minority class of the readmissions ("yes"); on multiclass, we saw a very significant decrease in accuracy but a more balanced distribution of F1 scores, which implies a more capable predicting capacity.

Secondly, we evaluated the **Gaussian NB** [Table 20] [Table 21] using only the var smoothing parameter of 0.0001 that we chose empirically, since we tested the different possibilities shown on the *sklearn* documentation, and the results didn't change significantly. We once again achieved similar results to the previous Logistic Regression, both in binary and multiclass classification, but this time slightly worse while at the same time decreasing the overfitting.

While using the **KNN Classifier** we noticed distinct differences between both problems (binary and multiclass). On the binary classification task, our model, optimised with parameters such as 4 neighbours, the 'auto' algorithm, 'distance' weight type, and the 'manhattan' metric, achieved an accuracy of 0.639 and an F1 score of 0.208, both on the validation set. This model exhibited a training set accuracy of nearly 0.999, suggesting a perfect or near-perfect fit on the training data, indicating overfitting to the training data. In the multiclass, we employed the same model but with 2 neighbours, the 'ball-tree' algorithm, 'uniform' weight type, and the 'euclidean' metric. This configuration gave us a weighted F1 score of approximately 0.316 on the validation set, reflecting a moderate performance. The algorithm displayed varied performance in classification tasks, adapting well to training data but struggling to generalize effectively across classes. [Table 20] [Table 21]

While using a 5-fold Stratified-Kfold, we tested and evaluated different parameters on multiple different hidden layer sizes that we confirmed in the *sklearn* documentation. From there, we used a grid search of both the optimal hidden layers we found previously and the remaining parameters we wanted to test to find the best combination that we finally applied to the **Neural Networks (MLP)**, giving us the best results yet [Table 20] [Table 21] on both binary and classification problems, while showing slight overfitting on the binary.

For the **Decision Tree** model, where we found the optimal combination for criterion, max-depth, max-features, max-leaf-nodes, min-samples-leaf, min-samples-split and splitter, to be gini, 1, None, None, 1, 2 and best, respectively and having in account that we had fixed previously the criterion and the splitter. While, for multiclass we got gini, 6, None, None, 2,2 and best as the best parameters. We applied these parameters for the binary and multiclass models and got results very close to the MLPs [Table 20] [Table 21], only that the accuracy increased and the precision, recall and F1-scores all decreased on binary. On multiclass the overall metrics increased compared to the previous models.

On bagging we first chose to use a **Random Forest**, a very robust model that normally deals better with unbalanced problems than a normal decision tree. Here we chose a variety of combinations of parameters where we would have a trade-off between performance and computational cost, based on the *sklearn* documentation (max depth=8, n estimators=11, ccp-alpha=0.0001 on binary). The results [Table 20] [Table 21] ended up being solid in the sense that on the binary problem we achieved an overall F1 score of 0.275 on the validation set and on the multiclass an F1 score of 0.541, while at the same time showing no signs of overfitting on both problems in part due to having a low standard deviation.

We also used a **Bagging Classifier** with a base estimator of a Decision Tree for both binary and multiclass problems. Bootstrap Aggregating is a powerful ensemble technique that combines the predictions from multiple machine learning algorithms to make more accurate predictions than any individual model. The main difference between the application of this ensemble on both problems was the parameters used, on binary the based estimator was a Decision Tree, with 100% of the samples, and 20% of the features without replacement. It achieved a training set F1 score of 0.746 and a validation set F1 score of 0.249. While the multiclass with the same base estimator but different parameter tuning (max samples=40%, max features=20%, with replacement) proved to be equally effective [Table 20] [Table 21].

Continuing, our next ensemble model is Boosting. For this model, we opted to use AdaBoosting and GradientBoosting. Starting with **Ada Boosting,** we used the default base estimator, the decision tree classifier. Then we ran a grid search with the purpose of finding the best values for the parameters: learning rate, number of estimators, and max depth of the base estimator. Having reached our best score of 0.279 we see that our learning rate, the weight applied to each classifier at each boosting iteration, shows that each classifier had a low contribution on the input, explaining why we had a high number of estimator been 100, lastly we had a max depth of 3 on our decision tree. For multiclass we got the same learning rate, a number of estimators 50 and max depth 1, but with a score of 0.422 [Table 20] [Table 21]. We also searched to find the best parameters in **Gradient boosting.** The parameters are learning rate, n estimators and subsample. Similarly, with Ada Boosting, the learning rate is 0.01 but this time it gave us 500 numbers of estimators,

which is a good result because a large number of estimators usually means better performance and lower over-fitting. For subsample we got a middle ground result of 0.6, knowing that there is a tradeoff of having a lower sub-sample means lower variance and greater bias. Finally, we observe that this model got the best F1 score yet of 0.286. For the multiclass it got the opposite result having a lower number of estimators being 2 and a higher learning rate of 1 with a subsample of 0.8, but an F1 score of 0.552 [Table 20] [Table 21].

In the quest for a better F1 score, we decided to implement **Stacking**, combining our best models to increase even further their F1 scores. For this, we tried many combinations as previously mentioned. As no surprise, the stackings with Gradient Boosting got the best results [Table 20] [Table 21], one even surpassing our best F1 score with a result of 28.7%. Even though the difference is not notorious, this increase is significant because of our dataset, as each model didn't have much of a gap between each other. Our multiclass didn't really improve from Gradient Boosting so we decided to use Gradient Boosting as our principal model on multiclass.

Finally, our last model was the **Voting Classifier**. For binary classification, we combined a Logistic Regression model with adjusted class weights, a Random Forest Classifier that was tuned using GridSearchCV, and a Gradient Boosting Classifier. We wanted to use the advantages of each individual model by employing a soft voting technique. The binary classification results showed difficulties in consistently identifying the minority class, while in the multiclass scenario, we modified our technique to account for the complexity of numerous readmission classes. We modified the class weights in Logistic Regression to account for the imbalanced distribution of these classes and used a weighted F1 scoring measure during the hyperparameter tuning of Random Forest. While the binary model showed a drop in overall accuracy, the distribution of F1 scores across different classes indicated an enhancement in differentiating between the three readmission groups. This improvement was particularly good in the increased representation and prediction of the less frequent '<30 days' category. [Table 20] [Table 21]

All the results were then compared side-by-side to get a better idea of different performances and to get an overall understanding of the behaviour of each model on both training and validation predictions. For binary: [Table 12] [Table 13] [Table 14] [Table 15] [Table 20]. For multiclass: [Table 16] [Table 17] [Table 18] [Table 19] [Table 21].

4. Conclusion

In conclusion, after a thorough procedure of data exploration to data preparation equal in both binary and multiclass targets we discovered that our best features were two different sets. Then by running and testing a variety of models with different parameters through grid-search, we concluded that regarding the binary target, our best model was stacking with Naive Bayes and Gradient Boosting (also the meta-model) and for multiclass Gradient Boosting. This is understandable as our best single model was gradient boosting which tends to eliminate overfitting with the many iterations, and this model was presenting overfitting. However, our result is not good enough as we got a 0.287 in our F1 score and 0.286 on Kaggle score, meaning this model is still not fit to be used in the healthcare industry as it will give us a high wrong prediction causing disorder in the hospital. Yet, this result can be because of our limited data, which was not only highly imbalanced but had many "Nan" and '?' values, this can be seen with our multi-class prediction as it has a less imbalanced target than the binary but we got a better score of 0.552. Scores can improve if we use oversampling for generating and creating data, and during the project we tried to apply it, getting a Kaggle score of 0.311 which is an increased value compared to our final result. Unfortunately, we have to quit that approach due to limitations on computational power, so for future work with better technological devices, it would be possible to start using this and try many other options of models and preprocessing techniques. Furthermore, in order to have the best predictions possible, it would be a good choice for the hospitals to recollect more data focusing on patients that get readmitted before 30 days, thus balancing the target values, and therefore improving the models and subsequently its F1 scores.

5. References

[1]. (n.d.). S6.4. Imputation of missing values — scikit-learn 1.3.2 documentation. Retrieved December 15, 2023, from https://scikit-learn.org/stable/modules/impute.html

[2]. (n.d.). 5. Visualizations — scikit-learn 1.3.2 documentation. Retrieved December 15, 2023, from https://scikit-learn.org/stable/visualizations.html

[3]. (n.d.). scipy.stats.spearmanr — SciPy v1.11.4 Manual. Retrieved December 15, 2023, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html

[4]. (n.d.). Online ICD9/ICD9CM codes. Retrieved December 15, 2023, from http://icd9.chrisendres.com/index.php?action=contents

[5]. (n.d.). sklearn.preprocessing.TargetEncoder — scikit-learn 1.3.2 documentation. Retrieved December 15, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.TargetEncoder.html#rf862141e5a0c-mic

[6]. (n.d.). sklearn.impute.KNNImputer — scikit-learn 1.3.2 documentation. Retrieved December 15, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html

[7] Neal, R. M. (n.d.). 1.1. Linear Models — scikit-learn 1.3.2 documentation. Retrieved December 15, 2023, from https://scikit-learn.org/stable/modules/linear_model.html#lasso

[8] Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2020). *[8] Fundamentals of Machine Learning for Predictive Data Analytics, Second Edition: Algorithms, Worked Examples, and Case Studies*. MIT Press.

[9] Megías, C., & Albarrán, G. (2023, April 23). *Glucose Tolerance Test - StatPearls*. NCBI. Retrieved December 17, 2023, from https://www.ncbi.nlm.nih.gov/books/NBK532915/

[10] *sklearn.linear_model.LogisticRegression — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[11] *sklearn.naive_bayes.GaussianNB — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

[12] *sklearn.neighbors.KNeighborsClassifier — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[13] *sklearn.neural_network.MLPClassifier — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[14] *sklearn.tree.DecisionTreeClassifier — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[15] *sklearn.ensemble.RandomForestClassifier — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[16] *sklearn.ensemble.BaggingClassifier — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

[17] *sklearn.ensemble.AdaBoostClassifier — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[18] *sklearn.ensemble.GradientBoostingClassifier — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

[19] *sklearn.ensemble.StackingClassifier — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html

[20] *sklearn.ensemble.VotingClassifier — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html

[21] *RandomUnderSampler — Version 0.11.0*. (n.d.). Imbalanced-learn. Retrieved December 19, 2023, from https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html

[22] *2. Over-sampling — Version 0.11.0*. (n.d.). Imbalanced-Learn. Retrieved December 19, 2023, from https://imbalanced-learn.org/stable/over_sampling.html

[23] *sklearn.feature_selection.mutual_info_classif — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html

[24] *6.3. Preprocessing data — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/preprocessing.html#target-encoder

[25] *sklearn.preprocessing.TargetEncoder — scikit-learn 1.3.2 documentation*. (n.d.). Scikit-learn. Retrieved December 19, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.TargetEncoder.html

[26] *The Role of Undersampling in Tackling Imbalanced Datasets in Machine Learning*. (2023, March 22). Train in Data Blog. Retrieved December 19, 2023, from https://www.blog.trainindata.com/undersampling-techniques-for-imbalanced-data/

[27] Chng, J. (2021, August 31). *The F1 score*. Towards Data Science. Retrieved December 19, 2023, from https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

[28] Chaudhary, K. (2020, August 24). *Bagging, Boosting, and Stacking in Machine Learning*. Drops of AI. Retrieved December 19, 2023, from https://dropsofai.com/bagging-boosting-and-stacking-in-machine-learning/

[29] *Gradient boosting vs AdaBoost | Learn the Differences and Comparisons*. (2023, March 8). EDUCBA. Retrieved December 19, 2023, from https://www.educba.com/gradient-boosting-vs-adaboost/

6. Creativity and other self-studies

Balancing Techniques

In both our feature selection and modelling, a key aspect was employing undersampling to address imbalanced data, where one class has significantly fewer instances than the other. We chose this approach because using other balancing techniques such as SMOTE or oversampling would increase the data points on our minority variable (denoted "Yes" in readmitted_binary), causing a larger dataset and a greater need in computational power when executing the models.

Explaining it in more detail, undersampling aims to balance the class distribution in the dataset. When using RandomUnderSampler we under-sample the majority class by randomly picking samples with or without replacement [21]. In our case, we chose to set the sampling strategy to the default parameter 'auto', which meant to not resampling the minority class, and with a random_state of 42, with "default replacement" (without replacement). We did all this by once again following the documentation of imbalanced data.

Oversampling techniques are slightly different, in the case of RandomOverSampling new samples are generated by random sampling with the replacement of the minority class, this is done by duplicating some of the original samples of the minority class so that the majority class does not take over the remaining classes during the training process [22]. There are multiple ways to minimize certain problems and situations, like using the parameter shrinkage if the goal is not to repeat samples (though the original data needs to be numerical). On other oversampling techniques, mainly SMOTE, the new samples generated are done so by interpolation, where the insertion of the data in the minority class is not made following any distinction between easy and hard samples, as mentioned in the documentation of imbalance techniques.

Mutual Information (MIC)

While doing the Feature Selection phase, we incorporated the MIC as an additional analysis tool, complementing our previous methods of feature selection. This approach assisted us in determining which variables were most suitable for inclusion in our modelling. The MIC allows us to explore the relationship between each independent variable and our dependent variable.

Their values range between 0 and 1, where higher values indicate a stronger relationship between the feature and the target. Meaning, that the closer the feature is to 0 the more independent it is to the target compared to the other variables. Therefore in our project, we established a threshold of 0.005, in order to not discard all variables but to decide the variables with the most dependency with our target variable. This means that the features we are looking for are the ones with higher values, as this gives more substantial information about the dependent variable. Mutual Information, in essence, provides a broader perspective, illustrating the reduction in uncertainty regarding the dependent variable once we have information about the independent variable.[23].

Target Encoding

Target encoding is a useful technique in machine learning for converting categorical features to numerical. It works by first selecting the categorical features that we have, then it compares how many target variables it has assigned to that categorical value, and subsequently calculates a mean average of the target variable with the categorical value. For example, we have a column which has 3 possible options: Normal, not normal or not measured. The algorithm goes through each option and calculates how many there are on the dataset, meaning it will count how

many people got a normal result. Then it measures how many people got readmitted from the individuals that got a normal result. To finally, calculate the division between the number of readmitted people and the number of people who have normal on the column. This process is done with the values “not normal”, and “not measured”. Finally, this value is replaced for each categorical value. This process is done for each categorical feature in our dataset. Converting every value to numerical. [24], [25]

We decided to use Target encoding, as it gives us a percentage for each categorical value concerning the target value, Creating a bigger correlation between both. Additionally, we tried one-hot-encoding but the results were not good enough.

The Voting Classifier

This is an ensemble technique in machine learning that leverages the predictions of numerous models to improve overall predictive performance, which normally exceeds the capability of individual models. It functions by combining predictions from multiple underlying models.

Each base model within the ensemble generates a distinct prediction, and after that the ensemble merges these predictions to yield the final result. The combination can be achieved through either 'hard voting' or 'soft voting'. In hard voting, the best possible prediction is determined by selecting the class that wins the majority rule voting from the base models [20]. Soft voting predicts the class label based on the argmax of the sums of predicted probabilities, so it calculates the average probability estimates from each individual model, and the final prediction is determined by selecting the class with the highest average probability, as mentioned in the documentation. This is normally more suited for ensembles of well-calibrated classifiers.

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
encounter_id	56988.0	NaN	NaN	NaN	548710.125219	259893.814793	100000.0	322854.25	547261.5	774720.5	999980.0
country	56988	1	USA	56988	NaN	NaN	NaN	NaN	NaN	NaN	NaN
patient_id	56988.0	NaN	NaN	NaN	54394273.120604	38817247.097834	135.0	23398488.0	45441346.5	87726521.25	189502619.0
race	54183	6	Caucasian	40524	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gender	56988	3	Female	30601	NaN	NaN	NaN	NaN	NaN	NaN	NaN
age	54161	10	[70-80)	13906	NaN	NaN	NaN	NaN	NaN	NaN	NaN
weight	56988	10	?	55165	NaN	NaN	NaN	NaN	NaN	NaN	NaN
payer_code	56988	18	?	22529	NaN	NaN	NaN	NaN	NaN	NaN	NaN
outpatient_visits_in_previous_year	56988.0	NaN	NaN	NaN	0.368464	1.28641	0.0	0.0	0.0	0.0	42.0
emergency_visits_in_previous_year	56988.0	NaN	NaN	NaN	0.19483	0.888386	0.0	0.0	0.0	0.0	76.0
inpatient_visits_in_previous_year	56988.0	NaN	NaN	NaN	0.64131	1.27662	0.0	0.0	0.0	1.0	21.0
admission_type	54022	7	Emergency	30157	NaN	NaN	NaN	NaN	NaN	NaN	NaN
medical_specialty	56988	68	?	27985	NaN	NaN	NaN	NaN	NaN	NaN	NaN
average_pulse_bpm	56988.0	NaN	NaN	NaN	99.556363	23.031045	60.0	80.0	100.0	119.0	139.0
discharge_disposition	54910	25	Discharged to home	33801	NaN	NaN	NaN	NaN	NaN	NaN	NaN
admission_source	53219	16	Emergency Room	32188	NaN	NaN	NaN	NaN	NaN	NaN	NaN
length_of_stay_in_hospital	56988.0	NaN	NaN	NaN	4.395732	2.993015	1.0	2.0	4.0	6.0	14.0
number_lab_tests	56988.0	NaN	NaN	NaN	43.091212	19.672638	1.0	31.0	44.0	57.0	121.0
non_lab_procedures	56988.0	NaN	NaN	NaN	1.342634	1.705808	0.0	0.0	1.0	2.0	6.0
number_of_medications	56988.0	NaN	NaN	NaN	16.015758	8.161822	1.0	10.0	15.0	20.0	75.0
primary_diagnosis	56988	664	428	3814	NaN	NaN	NaN	NaN	NaN	NaN	NaN
secondary_diagnosis	56988	670	428	3756	NaN	NaN	NaN	NaN	NaN	NaN	NaN
additional_diagnosis	56988	718	250	6416	NaN	NaN	NaN	NaN	NaN	NaN	NaN
number_diagnoses	56988.0	NaN	NaN	NaN	7.422598	1.939481	1.0	6.0	8.0	9.0	16.0
glucose_test_result	2922	3	Norm	1415	NaN	NaN	NaN	NaN	NaN	NaN	NaN
a1c_test_result	9496	3	>8	4552	NaN	NaN	NaN	NaN	NaN	NaN	NaN
change_in_meds_during_hospitalization	56988	2	No	30636	NaN	NaN	NaN	NaN	NaN	NaN	NaN
prescribed_diabetes_meds	56988	2	Yes	43920	NaN	NaN	NaN	NaN	NaN	NaN	NaN
medication	56988	283	['insulin']	17425	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 1

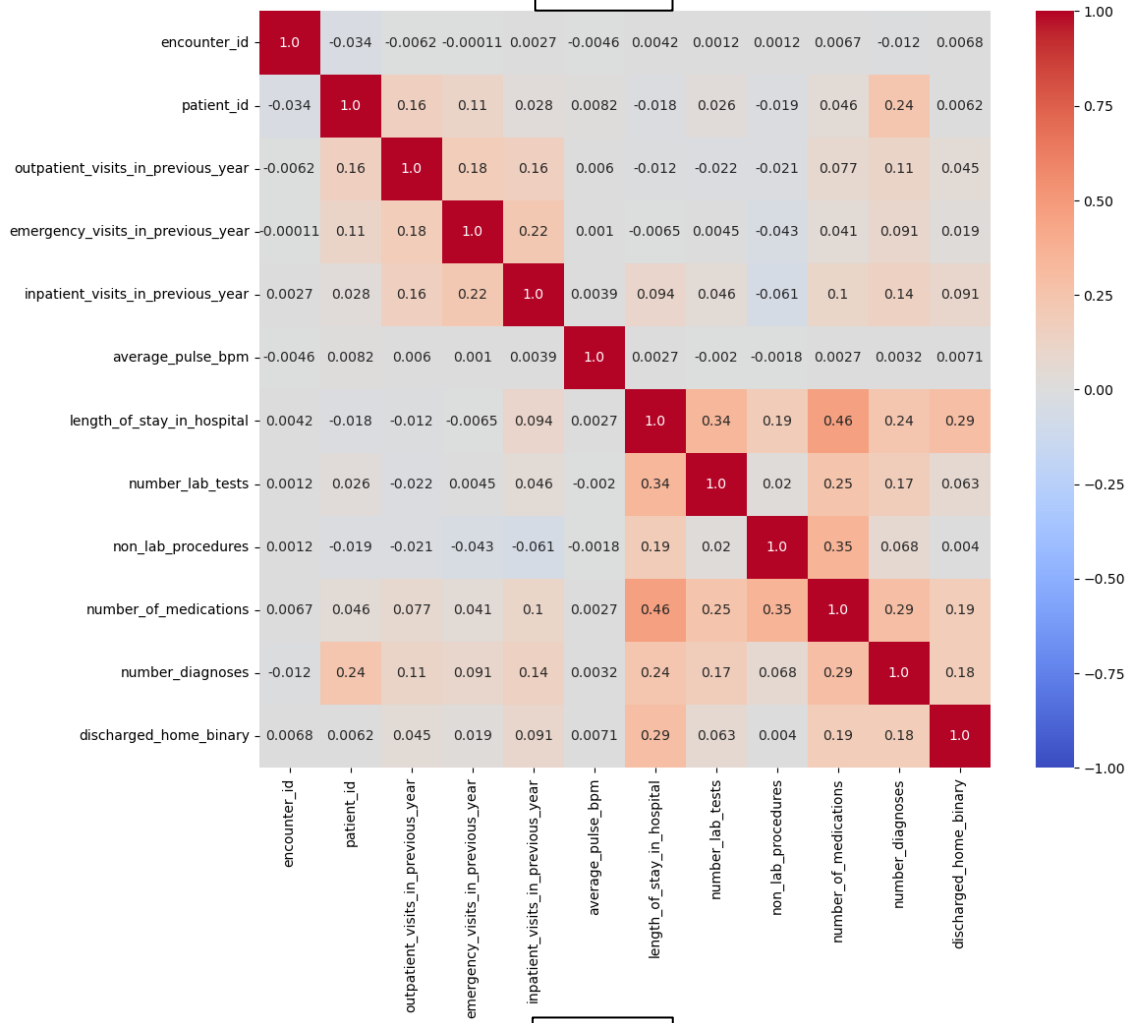

Table 2

Numeric Variables' Histograms

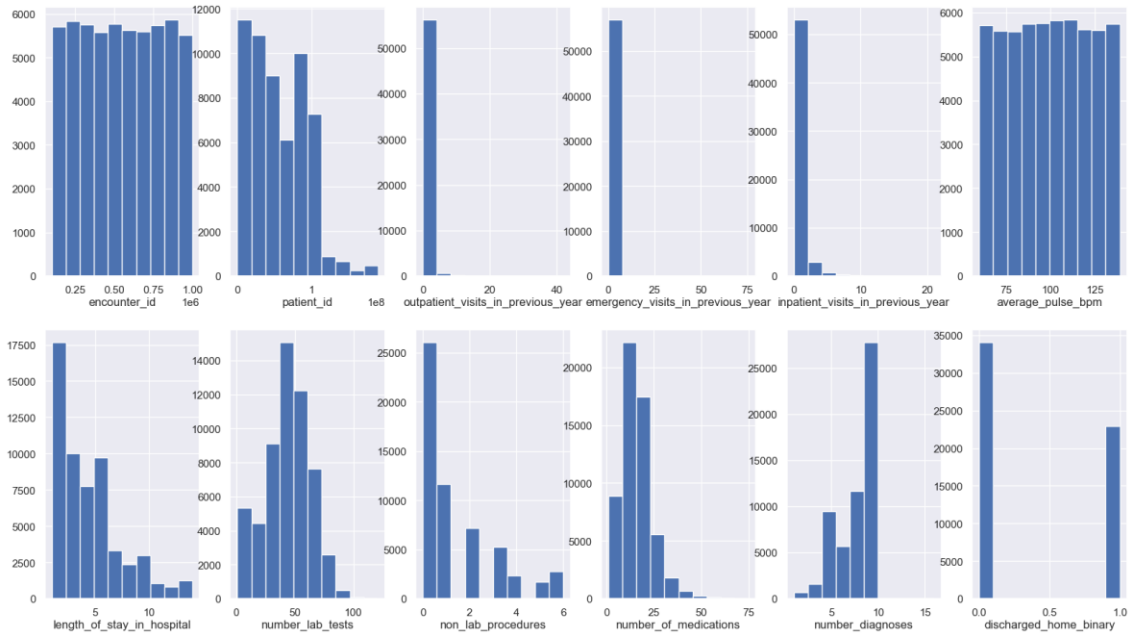

Table 3

Numeric Variables' Box Plots

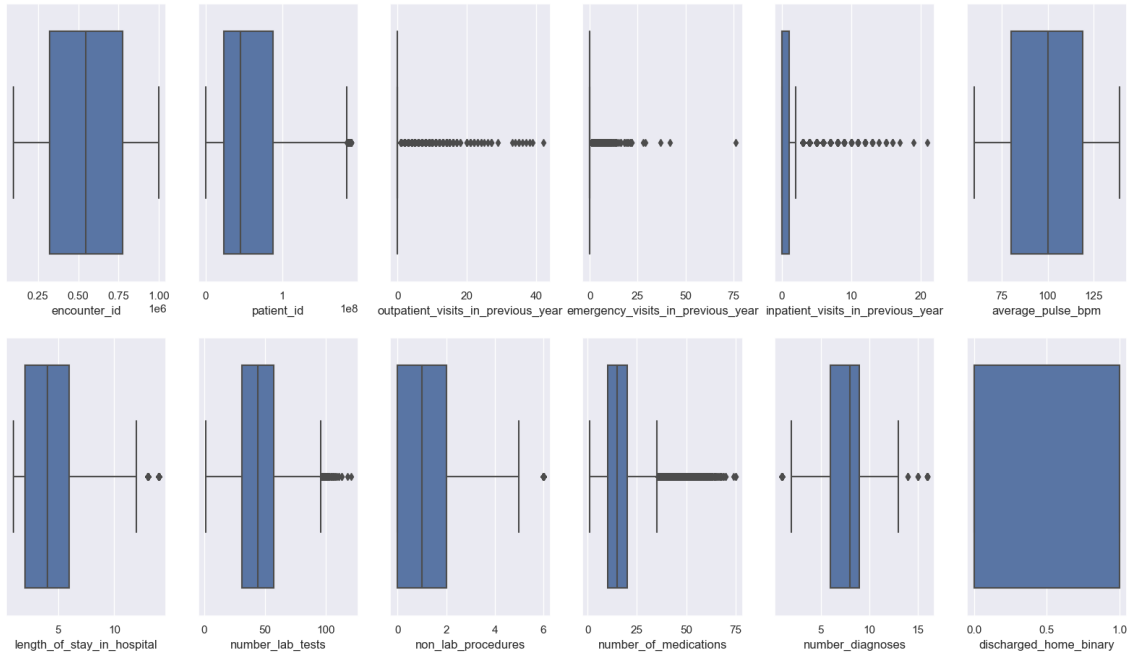

Table 4

Category ICD-9-CM	Codes
INFECTIOUS AND PARASITIC DISEASES	001-139
NEOPLASMS	140-239
ENDOCRINE, NUTRITIONAL AND METABOLIC DISEASES, AND IMMUNITY DISORDERS	240-279
DISEASES OF THE BLOOD AND BLOOD-FORMING ORGANS	280-289
MENTAL DISORDERS	290-319
DISEASES OF THE NERVOUS SYSTEM AND SENSE ORGANS	320-389
DISEASES OF THE CIRCULATORY SYSTEM	390-459
DISEASES OF THE RESPIRATORY SYSTEM	460-519
DISEASES OF THE DIGESTIVE SYSTEM	520-579
DISEASES OF THE GENITOURINARY SYSTEM	580-629
COMPLICATIONS OF PREGNANCY, CHILDBIRTH, AND THE PUERPERIUM	630-679
DISEASES OF THE SKIN AND SUBCUTANEOUS TISSUE	680-709
DISEASES OF THE MUSCULOSKELETAL SYSTEM AND CONNECTIVE TISSUE	710-739
CONGENITAL ANOMALIES	740-759
CERTAIN CONDITIONS ORIGINATING IN THE PERINATAL PERIOD	760-779
SYMPTOMS, SIGNS, AND ILL-DEFINED CONDITIONS	780-799
INJURY AND POISONING	800-999
SUPPLEMENTARY CLASSIFICATION OF FACTORS INFLUENCING HEALTH STATUS AND CONTACT WITH HEALTH SERVICES	V01-V89
SUPPLEMENTARY CLASSIFICATION OF EXTERNAL CAUSES OF INJURY AND POISONING	E800-E999

Table 5

Feature	LASSO	RFE	ANOVA	DT	Spearman	Approach
outpatient_visits_in_previous_year	X	X	X		X	Keep
emergency_visits_in_previous_year	X		X		X	Discard
inpatient_visits_in_previous_year	X	X	X		X	Keep
average_pulse_bpm	X			X	X	Discard
length_of_stay_in_hospital	X		X	X	X	Keep
number_diagnoses	X	X	X	X	X	Keep
discharged_home_binary	X	X	X		X	Keep
number_of_visits	X	X	X	X	X	Keep
primary_diagnosis_icd9	X		X	X	X	Keep
secondary_diagnosis_icd9	X			X	X	Discard
additional_diagnosis_icd9	X		X	X	X	Keep
procedures	X		X	X	X	Keep

Table 6

Feature	Mutual Information	Chi-squared	Decision Tree	Approach
race	X		X	Keep
gender	X		X	Keep
age	X		X	Keep
admission_type			X	Discard
medical_specialty			X	Discard
admission_source	X		X	Keep
glucose_test_result	X	X	X	Keep
a1c_test_result		X	X	Keep
change_in_meds_during_hospitalization				Discard
prescribed_diabetes_meds				Discard
Medicated_Status	X			Discard

Table 7

Feature	LASSO	RFE	ANOVA	DT	Spearman	Approach
outpatient_visits_in_previous_year	X	X	X		X	Keep
emergency_visits_in_previous_year	X	X	X		X	Discard
inpatient_visits_in_previous_year	X	X	X		X	Keep
average_pulse_bpm	X			X	X	Discard
length_of_stay_in_hospital	X	X	X	X	X	Keep
number_diagnoses	X	X	X		X	Keep
discharged_home_binary	X	X	X		X	Keep
number_of_visits	X	X	X	X	X	Keep
primary_diagnosis_icd9	X		X	X	X	Keep
secondary_diagnosis_icd9	X			X	X	Discard
additional_diagnosis_icd9	X		X	X	X	Keep
procedures	X	X	X	X	X	Keep

Table 8

Feature	Chi-squared	Decision Tree	Mutual Information	Approach
race		X	X	Keep
gender		X		Discard
age		X		Discard
admission_type		X	X	Keep
medical_specialty		X		Discard
admission_source		X	X	Keep
glucose_test_result			X	Discard
a1c_test_result		X	X	Keep
change_in_meds_during_hospitalization		X		Discard
prescribed_diabetes_meds				Discard
medicated status				Discard

Table 9

Feature
outpatient_visits_in_previous_year
inpatient_visits_in_previous_year
length_of_stay_in_hospital
number_diagnoses
discharged_home_binary
number_of_visits
primary_diagnosis_icd9
additional_diagnosis_icd9
procedures
a1c_test_result
gender
race
age
admission_source
glucose_test_result

Table 10

Feature
outpatient_visits_in_previous_year
'inpatient_visits_in_previous_year
'length_of_stay_in_hospital
'number_diagnoses
'discharged_home_binary
'number_of_visits
'primary_diagnosis_icd9
'additional_diagnosis_icd9
'admission_source
'a1c_test_result
'procedures
'race
'admission_type

Table 11

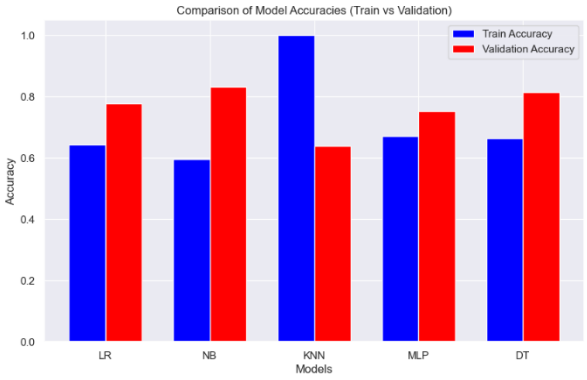

Table 12

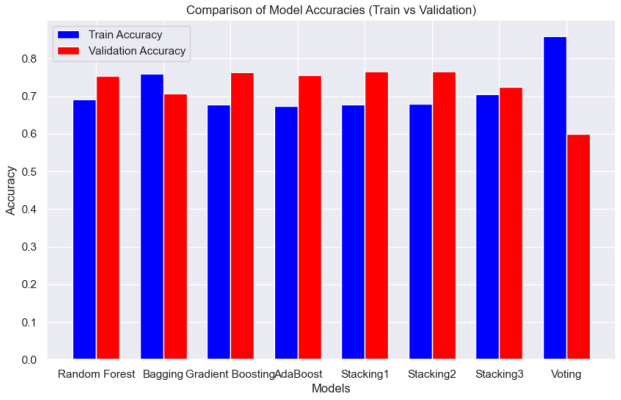

Table 14

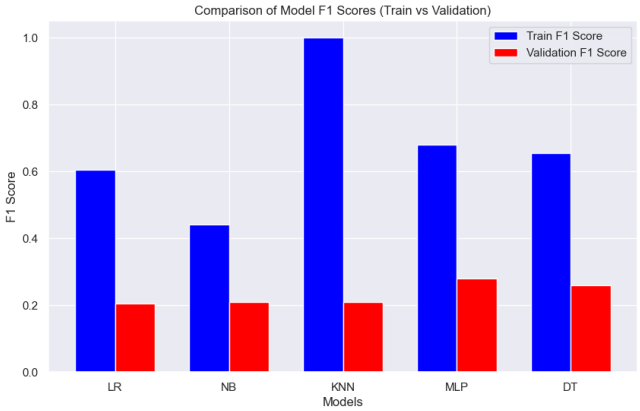

Table 13

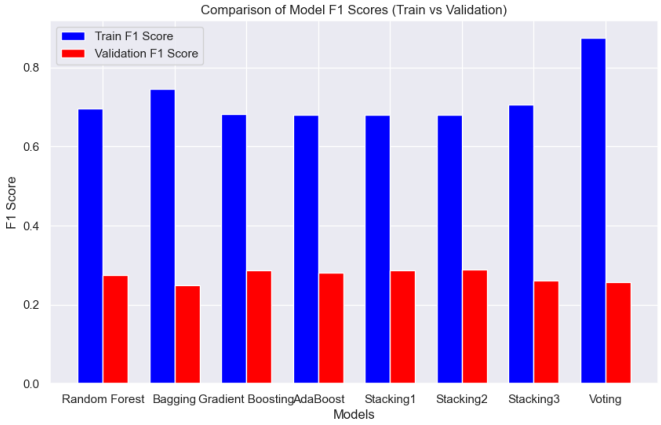

Table 15

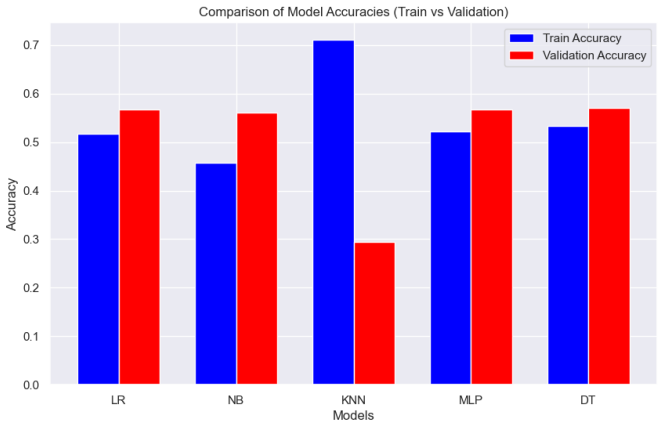

Table 16

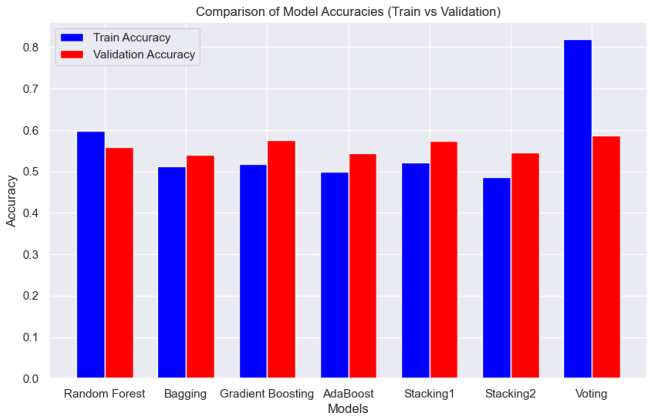

Table 18

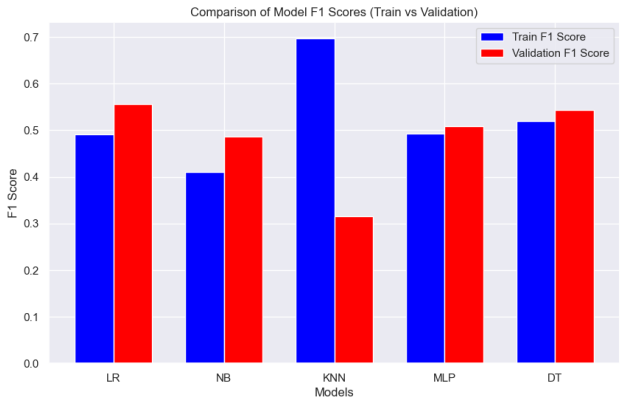

Table 17

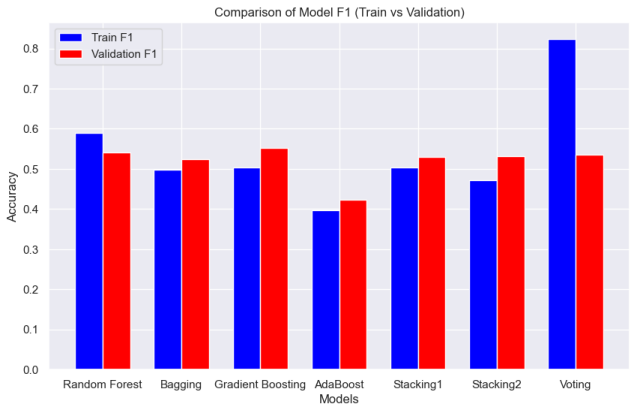

Table 19

Model	Accuracy Train	Accuracy Validation	F1 Train	F1 Val
Logistic Regression	0.6436610911364702	0.7784250421111735	0.6044251626898048	0.25100830367734284
Naive Bayes	0.5949663904955448	0.8333099382369455	0.4406303972366149	0.2101762554040572
KNN Classifier	0.9997655150851962	0.6397389107243122	0.9997654600891251	0.20848111025443333
MLP Classifier	0.6719556041894638	0.7526670409882089	0.6793490717396286	0.2796402289452167
Decision Tree	0.6632015007034547	0.8150617630544638	0.6549763792137079	0.25962348974431015
Random Forest	0.6922776301391277	0.75414093206064	0.6967105769971498	0.27519139250982827
Bagging Classifier	0.7596529623260904	0.7077484559236384	0.7463917525773196	0.24972972972972973
Gradient Boosting	0.6778958886978271	0.764528354856822	0.6815547484738428	0.2866255581543696
AdaBoosting	0.6735969985930905	0.7554042672655812	0.6794104099493321	0.2798098780739822
Stacking1	0.6775050805064874	0.76614261650758	0.68015503875969	0.28650963597430407
Stacking2	0.6800062529310614	0.7659320606400898	0.6807050382155669	0.2878496690155883
Stacking 3	0.7056432702829452	0.7253649635036497	0.7049514258853025	0.26127996979422313
Voting Classifier	0.8587619196498358	0.6002947782144863	0.8741380511248867	0.2575935340894277

Table 20

Model	Accuracy Train	Accuracy Validation	F1 Train	F1 Val
Logistic Regression	0.5172737220572143	0.5676586187535093	0.49193506201720955	0.5011054134190449
Naive Bayes	0.4577145536970455	0.5607804604154969	0.4099608629312454	0.4868203106836663
KNN Classifier	0.7105935073732479	0.2941465468837732	0.6966396795858695	0.3159872770251708
MLP Classifier	0.5212860194883018	0.5673778775968557	0.4926336074805868	0.5089810967532663
Decision Tree	0.5341045281642437	0.5706765861875351	0.5201332063105848	0.5431592146306953
Random Forest	0.5975717784378094	0.5596574957888827	0.5887830578057717	0.5409333090652663
Bagging Classifier	0.513105101349591	0.54035654126895	0.49779775968160433	0.5235655840156741
Gradient Boosting	0.5184200927518108	0.5753790005614823	0.5030478363570519	0.5523076121101913
AdaBoosting	0.4993486530144339	0.5435850645704661	0.39737638462024927	0.4224058625088279
Stacking1	0.5223281746652076	0.5729225154407637	0.503149746460197	0.5306884212480572
Stacking2	0.48679068313271845	0.5466732172936553	0.4724274452376502	0.5319585808039137
Voting Classifier	0.8188734302537648	0.5868192026951151	0.8229449361102387	0.5349787462112952

Table 21