Handling missing values

For project 1, we have considered only 20 columns from the dataset to answer our SMART questions. In the second phase of the project which involves model building we have decided to include all of the 186 columns in the dataset. Since, we have not preprocessed the rest of the columns in project 1, we proceeded with the cleaning process first. Calculation of missing values in each column was done first. Missing values were replaced with NA’s for ease of understanding. We have employed multiple methods to handle the missing values. First step involves removing columns with more than twenty three percent of missing values from the dataset. This step was necessary as imputing a quarter of data would generate a reasonable amount of bias.

Next, shape of each column’s distribution was calculated to ensure whether to impute mean, median or mode in that particular column’s missing values. To achieve this, we had to loop through all the columns and generate histograms for each column. After understanding the distribution of the columns, imputation of mean was done to columns with normal (Gaussian) distribution (mean = median), median to columns with skewed distribution (if left skewed, mean < median else mean > median) and mode in categorical columns respectively. We’ve experimented with cluster specific imputation and regression based imputations as well. But they were computationally intensive and were taking a long time to run, which is why we had to drop those methods.

Feature Selection

Feature selection is of one of the most important steps before building the model. Since, we have 186 independent variables, it was of high significance for us to make sure that we have chosen appropriate features for our model. Hence, to accomplish this we have employed four feature selection methods and have chosen features from these. The methods used are, Point Biserial correlation technique, chi-square test of independence, Lasso regularization based feature selection and regression based step-wise selection .

Point biserial correlation method

The point-biserial correlation coefficient is a measure of association that quantifies the strength and direction of the relationship between a continuous numerical variable and a binary (dichotomous) variable. In the context of feature importance methods, the point-biserial correlation is commonly used to assess the relationship between individual continuous features and a binary target variable.

The provided code aims to evaluate the point-biserial correlation coefficients between each feature in the dataset 'g' and the binary target variable 'hospital\_death.' First, the target variable is extracted from the dataset. Then, the script iterates through each feature, calculating the point-biserial correlation coefficient only for numeric features while handling non-numeric features appropriately, marking them as 'NA.' This step ensures that the analysis focuses on the relationship between continuous numerical features and the binary target variable. The correlation coefficients are stored in the 'correlation\_with\_target' vector. Finally, a threshold, in this case, an absolute correlation value greater than 0.1, is applied to select features deemed to have a meaningful association with the target variable. The names of the selected features are stored in the 'selected\_features' vector, which can be printed or used for subsequent analyses.

This code provides a straightforward method for feature selection based on point-biserial correlation, helping to identify features with notable associations with the binary target variable. The correlation of 0.1 which is low is considered because we were aiming to capture features even with a small correlation as our’s is a hospital – patient dataset where every small things matter.

Regression based step-wise selection

Stepwise feature selection is a method used in statistical modeling to iteratively choose a subset of features that contribute most to the predictive performance of a model. It typically involves two main steps: forward selection and backward elimination. In forward selection, the algorithm starts with an empty set of features and adds the most informative variable at each step until a stopping criterion is met. In backward elimination, the algorithm begins with all features and removes the least significant ones based on statistical criteria. The process continues until the model's performance, often measured by criteria like AIC or BIC, is optimized. Stepwise feature selection helps enhance model interpretability, reduce overfitting, and improve computational efficiency by identifying a subset of features that collectively contribute to better predictive accuracy.

The provided R code utilizes the MASS library, alongside dplyr and caret, to perform logistic regression with stepwise feature selection using the Akaike Information Criterion (AIC). The dataset 'g' undergoes preprocessing to exclude columns with less than two unique factor levels and convert binary factors to numeric format. Dummy variable transformation is applied to categorical predictors. Subsequently, logistic regression is conducted with a stepwise forward selection approach to identify relevant features associated with the binary target variable 'hospital\_death.' The final model summary displays the selected features and their corresponding coefficients, providing insights into the significant predictors contributing to the binary classification task.

Chi squared test of independence

The provided R code conducts a series of chi-square tests to assess the association between specific categorical features and the binary target variable 'hospital\_death' in the dataframe 'g'. The features selected for testing include medical conditions and demographic factors.For each feature in the 'features\_to\_test' vector, the code ensures it is treated as a factor variable and then performs a chi-square test using the chisq.test function. The results, including the chi-square statistic and p-value, are accumulated in a data frame named 'chi\_square\_results.'The printed output displays the statistical results for each tested feature, showcasing the chi-square statistic, p-value, and the corresponding feature name. These results offer insights into the potential significance of individual features concerning the occurrence of 'hospital\_death.' Lower p-values indicate stronger evidence against the null hypothesis of independence between the feature and the target variable. Overall, this approach helps identify features with potential predictive value for the binary outcome.

Lasso regularization based feature selection

The provided R code employs the glmnet package to perform Lasso (L1 regularization) logistic regression for feature selection in the context of predicting 'hospital\_death.' A specific set of features, encompassing various medical and demographic variables, is pre-defined for inclusion in the model. The code ensures the presence of these features in the dataframe 'g' and proceeds to create a predictor matrix (X) and a response variable (y). The Lasso model is then fitted using cross-validation to determine the optimal regularization parameter (lambda) that minimizes cross-validated error. Subsequently, the model is refit with the chosen lambda, and the coefficients are extracted. The non-zero coefficients, indicative of the selected features, are identified and printed as the final result. This approach helps identify the most influential features for predicting hospital mortality while mitigating the impact of irrelevant or redundant variables through L1 regularization.

Common feature extraction advantages

The chosen feature selection methods offer distinct advantages in enhancing model interpretability, reducing dimensionality, and improving predictive accuracy. The Point Biserial correlation technique is effective in identifying linear relationships between continuous features and a binary target variable. The chi-square test excels at detecting associations between categorical features and the target. Lasso regularization not only selects relevant features but also imposes sparsity, mitigating multicollinearity and improving model generalization. Regression-based step-wise selection systematically refines the feature set by adding or removing variables based on statistical criteria, optimizing model performance. By extracting common features identified by these diverse techniques, we ensure a comprehensive and robust feature set that captures both linear and non-linear relationships, categorical associations, and penalizes irrelevant variables. This integrated approach contributes to a more informed and accurate model for predicting hospital mortality, leveraging the strengths of each method for a well-rounded feature selection strategy.

Why didn’t we remove columns based on multicollinearity

In the context of healthcare and patient data analysis, the decision to not prioritize the identification and removal of multicollinear variables is well-founded. Health data is characteristically interrelated, encapsulating complex biological, behavioral, and environmental factors that collectively influence health outcomes. The intrinsic correlations among these variables often hold significant clinical relevance, providing insights into the multifaceted nature of health and disease. Omitting variables based on multicollinearity concerns may inadvertently strip the dataset of critical information pertinent to understanding patient health. Furthermore, modern machine learning algorithms, especially tree-based models, are adept at handling correlated predictors. They can discern and leverage these relationships effectively without substantial detriment to the model’s performance. Hence, in a healthcare data analysis scenario, maintaining such interconnected variables is not only methodologically sound but also crucial for a holistic understanding of the health dynamics at play.

Encoding categorical columns

The code snippet provided performs one-hot encoding on the specified categorical columns in the "new\_dataframe2" dataset. Categorical columns such as "ethnicity," "gender," and others are selected for encoding. The model.matrix function is used to convert these categorical columns into a binary matrix, where each unique category becomes a binary column. The "- 1" in the formula ensures that no intercept term is included. The result is stored in the "encoded\_data" data frame. Finally, the non-categorical columns from "new\_dataframe2" are appended to the one-hot encoded data, resulting in an updated dataset, "new\_dataframe2," with the categorical variables transformed into a format suitable for machine learning model training. This preprocessing step is essential to ensure that the model can effectively interpret and utilize categorical information during training.

Logistic regression

The provided code segment demonstrates the training and evaluation of a logistic regression model using the caret and glmnet libraries. The dataset, "new\_dataframe2," is split into training and testing sets (80/20 split), and a logistic regression model is trained using the glmnet function. Predictions are then made on the testing set, and the predicted probabilities for the positive class are extracted. A threshold of 0.5 is applied to convert these probabilities into binary predictions. The code evaluates the model's performance by generating a confusion matrix and calculating accuracy. If the lengths of the vectors match, the confusion matrix and accuracy are printed. This process ensures an assessment of the logistic regression model's predictive ability on the given dataset.

Decision tree before tuning

The decision tree model, illustrated in the provided code, offers distinct advantages over logistic regression, particularly in scenarios where the relationships between predictors and the response are non-linear and complex. Decision trees excel at capturing intricate patterns and interactions within the data, providing a more flexible and interpretable framework. Unlike logistic regression, decision trees inherently perform feature selection during the model-building process, promoting simplicity and potentially preventing overfitting by prioritizing the most informative variables. Additionally, decision trees are adept at handling various data types and are robust to outliers.

In this R code snippet, a decision tree model is constructed using the 'rpart' package for binary classification. The dataset is preprocessed by converting the target variable, 'hospital\_death,' into a factor in both the training and testing sets. The decision tree model is trained on the training data using the 'rpart' function, and predictions are then generated on the testing dataset. Subsequently, a confusion matrix is computed and displayed, offering insights into the model's performance in terms of true positives, true negatives, false positives, and false negatives. The accuracy of the model is calculated from the confusion matrix, providing a measure of its overall predictive capability. Additionally, the 'rpart.plot' package is employed to visualize the decision tree, enhancing interpretability. This comprehensive workflow allows for a thorough evaluation of the decision tree model's effectiveness in classifying instances of hospital mortality.

Addressing class imbalance in target variable

Addressing class imbalance is crucial in binary classification tasks to ensure that machine learning models do not favor the majority class and exhibit biased predictions. In the provided analysis, the recognition of imbalanced classes in the 'hospital\_death' variable prompted the application of two class imbalance techniques: SMOTE (Synthetic Minority Over-sampling Technique) and ROSE (Random Over-Sampling Examples). These methods generate synthetic instances of the minority class, thus augmenting the training data and mitigating the impact of class imbalance on model performance. By employing SMOTE and ROSE, the models become more adept at learning patterns within the minority class, enhancing their ability to make accurate predictions for both classes. This proactive approach to handling class imbalance contributes to more reliable and fair model outcomes, ensuring that the model does not disproportionately favor the majority class and providing more robust predictions across both classes.

SMOTE

The Synthetic Minority Over-sampling Technique (SMOTE) is a resampling method designed to address class imbalance by generating synthetic instances of the minority class. In the provided code, SMOTE is applied to the 'hospital\_death' variable, which is the target binary outcome. The initial step involves converting integer and character columns to numeric format, excluding the target variable, to ensure compatibility with the SMOTE algorithm. SMOTE operates by identifying instances of the minority class (in this case, cases where 'hospital\_death' equals 1) and creating synthetic samples by interpolating between neighboring minority instances. The 'perc.over' parameter controls the extent of oversampling, and the 'k' parameter determines the number of nearest neighbors to consider during the interpolation process. The result is a rebalanced dataset that includes synthetic instances of the minority class, ensuring a more equitable representation for improved model training. This strategic augmentation aids in mitigating the impact of class imbalance, enhancing the model's ability to make accurate predictions for both classes.

ROSE

Embracing the ROSE (Random Over-Sampling Examples) technique involves leveraging the caret package to partition the dataset into training and testing sets. The essence of ROSE lies in its capacity to strategically oversample the minority class, 'hospital\_death' in this context, by introducing synthetic instances with controlled randomness. The ROSE algorithm employs a proximity-based approach to generate synthetic examples, ensuring a nuanced and realistic representation of the minority class distribution. Unlike traditional oversampling methods, ROSE's controlled randomness captures the intricacies of the minority class, enhancing the quality of the synthetic instances. The decision to prefer ROSE over alternatives like SMOTE is rooted in its ability to address class imbalance nuances effectively. This tailored strategy aligns with the unique characteristics of the dataset, promoting a more accurate and robust model.

Hyper parameter tuning

Initially, the data undergoes preprocessing steps, ensuring valid and consistent column names through the make.names function and addressing any potential missing values. Subsequently, the classification model is trained using the caret package's train function, incorporating 10-fold cross-validation to optimize the model's complexity parameters. The provided R code includes hyperparameter tuning for a decision tree model. The hyperparameters of a decision tree model, such as the complexity parameter (cp), control the model's behavior and performance. In the given code, the tuneGrid argument is utilized within the train function to explore different values of the complexity parameter (cp) in a specified range (seq(0.01, 0.05, by = 0.02)).By employing 10-fold cross-validation (trControl = trainControl(method = "cv", number = 10)), the code systematically evaluates the decision tree model's performance for different values of the complexity parameter. This process helps identify the optimal value of cp that maximizes the model's accuracy or another chosen performance metric.Therefore, the code performs hyperparameter tuning by searching for the most suitable complexity parameter for the decision tree model, enhancing its potential for generalization and predictive accuracy on unseen data.

AUC & ROC

The provided R code utilizes the pROC library to calculate the Receiver Operating Characteristic (ROC) curve and the Area Under the Curve (AUC) for a classification model. Assuming 'tuned\_model' is a trained model and 'testing\_data3' is the corresponding test dataset, the code generates probability predictions for the positive class. Subsequently, it extracts the predicted probabilities and calculates the ROC curve and AUC using the pROC package.The ROC curve visually represents the trade-off between true positive rate and false positive rate at various probability thresholds. The AUC provides a scalar value summarizing the overall performance of the model—higher AUC values generally indicate better discrimination between classes. Therefore, the code evaluates and prints the AUC value as a measure of the model's ability to distinguish between positive and negative instances, offering insights into its predictive accuracy.

Model coparisons and assessing which model is best and why

SMART QUESTIONS