

**BUDT 758T Project Phase II Report**

**– Group 10**

**Contents**

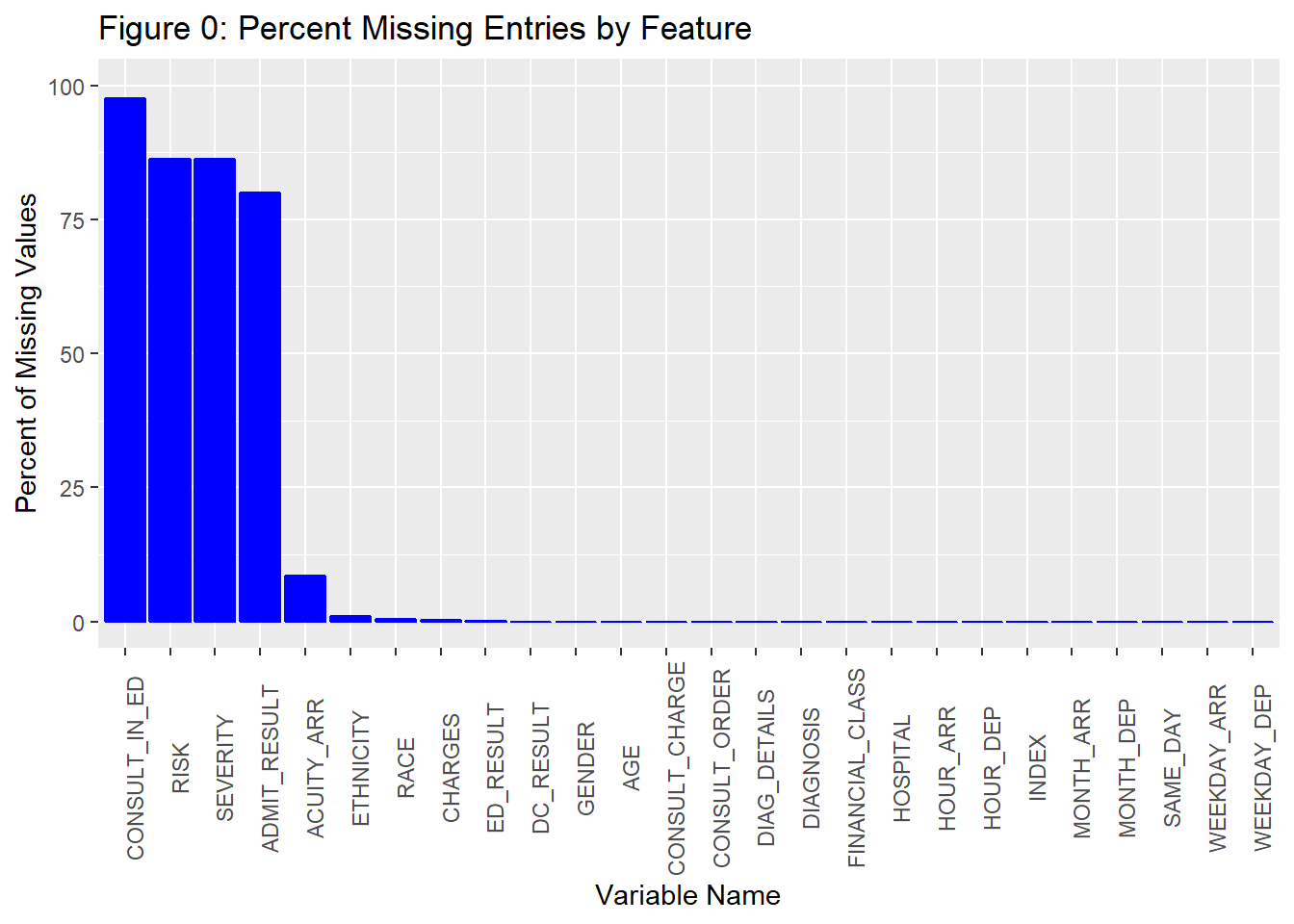
[Section 2: Predictive Modeling Insights 2](#_Toc8542583)

[Appendix 4](#_Toc8542584)

# Section 2: Predictive Modeling Insights

One of the business objectives is met through our ability to predict returning patients. While, true positive rate is expected to be a good candidate metric to evaluate prediction, we have focused on accuracy as TPR doesn’t include misclassification in negative samples and we also don’t want to classify too many non-returning patients as returning patients in pursuit of improving TPR.

Next step was to focus on imputing missing values. We started by replacing ‘’, ‘#N/A’, ‘#VALUE’ entries to NA for consistency. For the variables (test and train data combined) CONSULT\_IN\_ED (97.6% missing), RISK (86.3% missing), SEVERITY (86.3% missing), and ADMIT\_RESULT (79.9% missing), we have categorized the missing values as a new category as we believe that the missing values are probably not recorded or measured. Next, wherever appropriate, we converted variables to factors. For example, we converted variable SAME\_DAY (0 or 1) to factor variable. Figure below provides details of missing values by variable before imputation.

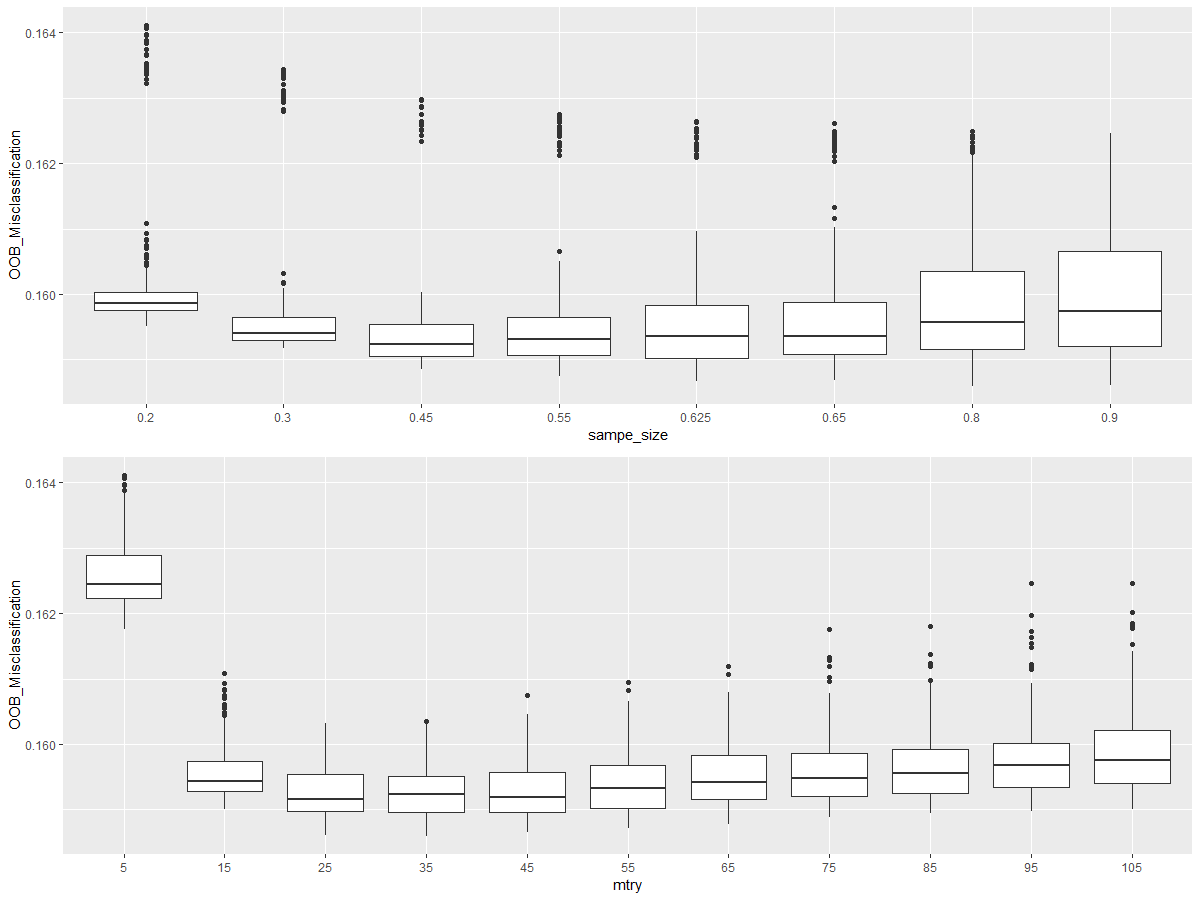


In addition, we did feature engineering on DC\_RESULT, which had many categories with only a limited number of values. We expect that limited observations for a category may impact the working of algorithms especially the QDA, which requires a certain number of observations in each category as it calculates mean and variance by categories in a variable. We binned categories with limited number of observations. Refer to appendix for details of number of observations by category of the DC\_RESULT variable.

After completing the above-mentioned steps, we impute the missing values in the remaining variables using predictive mean matching method using mice package in R. Predictive mean matching uses regression to impute numeric variables and logistic for categorical variables. It builds linear regression using the variables to be imputed as response variable and all other variables as predictor variables. The method uses posterior predictive distribution (distribution of possible unobserved values conditional on the observed values) to identify multiple β’s of the regression/logistic equation and suggests multiple potential candidate values for missing entries.

Once, we had clean and complete data (with no missing values, appropriate variable type, and appropriately binned), we built predictive models such as classification tree, KNN, logistic regression, LDA, QDA. However, accuracy of all these methods on holdout/testing dataset was below 77.5%, slightly above the baseline.

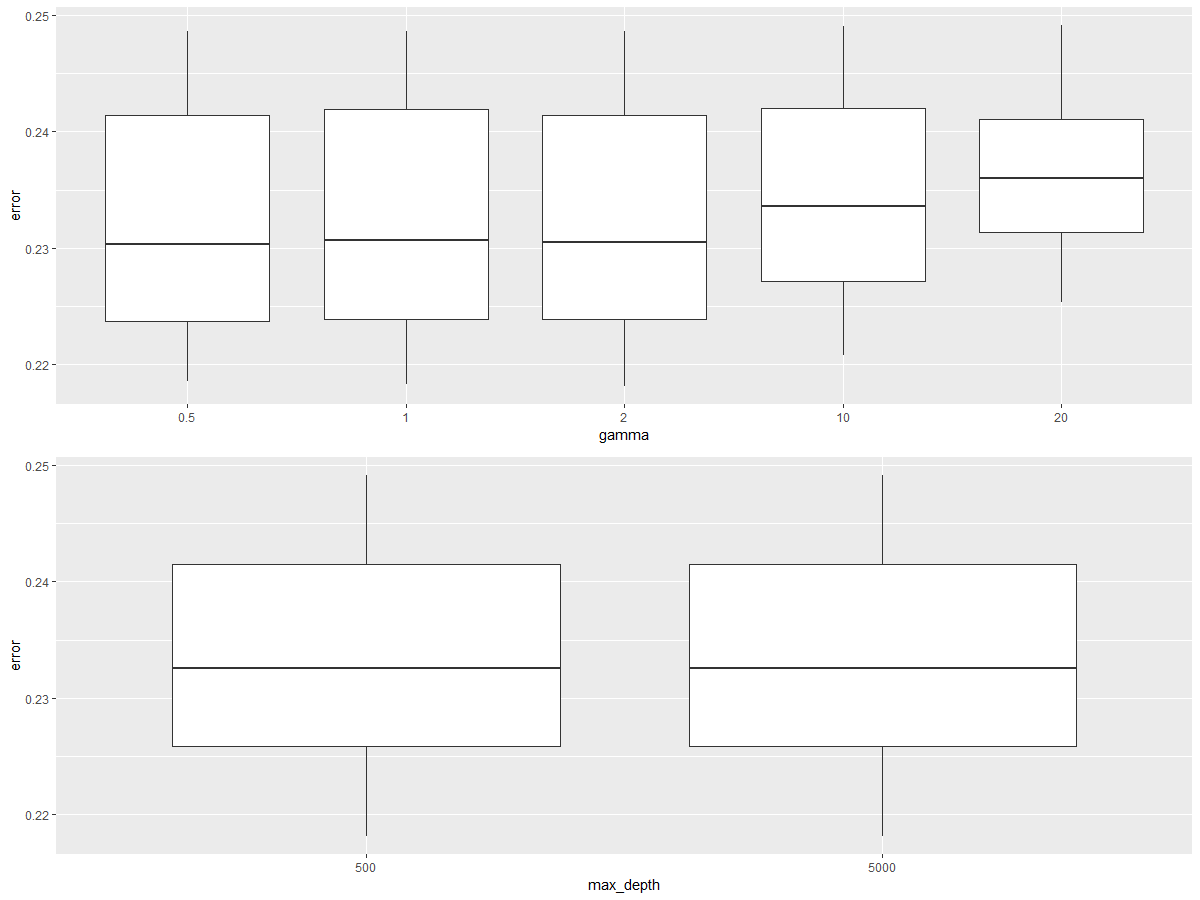
As a result, we decided to try ensemble methods in pursuit of improving prediction accuracy. Before we implement ensemble methods, we used model.matix() function in R to convert categorical variables to dummy variables. This was not necessary for all methods, but we wanted to use same data structures for all models to compare holdout accuracy. We started with random forest and bagging ensembles based on trees, which build multiple trees on bootstrapped sample of the data and final prediction is the sum of all prediction by each individual tree. Random forest decorrelate trees by allowing random selection of features at tree split. We used ranger package in R to implement random forest. Ranger is a fast implementation of random forest. The first step was to optimize the parameters as they can have significant impact on prediction accuracy. We focused on optimizing four parameters - sample.fraction(….), mtry (….), num.trees(…), min\_node\_size (….). We used cartesian search on a grid of 1782 rows containing various combination of parameters. We ran random forest algorithm 1782 times to identify best parameters (based on out of bag (OOB) misclassification rate) and found that the lowest OOB misclassification rate was for num.tress = 5000; mtry= 35; min\_node\_size=35; sample.faction=0.8. The figure below plots OOB misclassification rate vs various combinations of sample.faction and mtry used.



We used the best parameters (obtained above) to build tress on complete training dataset except on the holding data, which we kept separate to compare accuracy of different ensemble models we are going to build. Nonetheless, we found that the holdout accuracy of the prediction increased drastically from <77.5% to 78.5%.

In pursuit of improving accuracy further, we focused on implementing gradient boosting method (GBM) based on trees, which is useful to correctly classify difficult to classify observation by increasing their weights. Boosting is also known as a method to convert weak learners into strong learners. It begins by training a classification tree algorithm on original dataset and gives greater weightage to misclassified datapoints in the subsequent steps to build trees. Final prediction is sum of predictions by all trees. However, this additional step adds more parameters to the model, which also require tuning/optimization. The new parameters are mainly related to the learning rate. We used GBM implementation in h2o package as it allows parallel computation and hence computation is faster. The parameter trained include col\_sample\_rate (..); learn\_rate (….); learn\_rate\_annealing (…); max\_depth (….); min\_rows (….); ntrees (….); sample\_rate (….). We used cartesian search on a grid of 70 rows containing various combination of parameters. We ran GBM algorithm 70 times to identify best parameters (based on lowest misclassification rate) and found that the lowest misclassification rate was for col\_sample\_rate=1; learn\_rate=0.1; learn\_rate\_annealing=20; max\_depth=; min\_rows (….); ntrees (….); sample\_rate (….). We used the best parameters (obtained above) to build tress on complete training dataset except on the holding data. We found that the holdout accuracy of the prediction increased drastically from 78.5% to 78.7%.

Thereafter, we implemented extreme gradient boosting algorithm (Xgboost) based on trees, considered one of the best algorithms for prediction accuracy and used by many to win data challenge competition at Kaggle. Xgboost is regularized GBM. As a result, regularization parameters are added to the list of parameters for tuning. We used xgb.cv function in the xgboost package in R to optimize parameters based on cross validation. The parameter trained include col\_sample\_rate (..); learn\_rate (….); learn\_rate\_annealing (…); max\_depth (….); min\_rows (….); ntrees (….); sample\_rate (….). Cartesian search on a grid of 70 rows containing various combination of parameters. We found that the lowest misclassification rate was for max\_depth=500; min\_child\_weight=15; subsample=1 0.2; eta=0.7; colsample\_bytree=0.05; lambda=0.9; alpha=2; gamma=10; nrounds=25. We used the best parameters (obtained above) to build tress on complete training dataset except on the holding data. We found that the holdout accuracy of the prediction increased further from 78.7 to 79.1%.



Finally, we implemented a model stacking method that used predictions from base learners (xgboost, random forest, GBM, logistic regression) to be used as variable of a new model (xgboost), which is used for final prediction. We split dataset in three parts and used the first part to train base learners and used the model to make prediction for second and third part of the dataset. Then, we trained new model on the second part and used it to make prediction on the third part, which is the holdout set. We found that the holdout accuracy of the prediction increased further from 79.1 to 79.7%.

The best model, which is stacked model was trained on complete training dataset and available for prediction on an unseen dataset. We suggest that the hospital should regularly update the model as new data is gathered.

# Appendix

