Final Report

BUDT758T - Data Mining and Predictive Analytics

Group 7 Member: Ziyang Dong, Jingtian Li, Mingfeng Shen, Zhenhao Chen

### Executive summary

Hospital readmission shortly after discharge is considered as a significant contributor to health costs. Identifying patients who are likely to be readmitted helps improve treatment quality and save medical resources. This report aims at predicting whether a patient will return to the hospital within 30 days of being discharged, providing health organization with an excellent model to help them predict the future actions of different people, using collected data on returned and non-returned patients.

Before building models, we cleaned the dataset and selected predictive features. For data pre-processing, we re-leveled categorical variables, scaled numeric variables, converted data types to reasonable types, and dropped useless variables.

After data pre-processing, we used clean data to create prediction models that would accurately predict whether a patient would be returned within 30 days. We split the train dataset (with labels) into three parts: training, validation, and testing. We used training data to build the model, used validation data to choose the best parameters, and used accuracy to compare models based on testing data. After that, we identified new patients on the prediction dataset (without labels) that would be readmitted. The baseline accuracy is 74.44% for testing data, 75.45% for prediction data. We built six different models using Logistic Regression, LASSO, Feedforward Neural Network, Random Forest, Boost and XGBoost. Each model has its own features:

* Logistic regression not only helps us predict the binary classification problem but also gives us a general view of the inference at the very beginning.
* LASSO helps us delete or give us a guide to eliminate variables.
* Feedforward neural network can learn its parameters and bias automatically which can have a more accurate model.
* Random Forest method focuses on the whole picture of the variables, we changed our parameters in the function to make a better prediction.
* Boosting method focuses on incorrect data. Each time it re-estimated the model and put extra weight on the points that are misclassified.
* XGBoost is a gradient boosting method that provides parallel computation to predict data in a fast and accurate way.

The best model we got is the model trained by XGBoost with 77.51% testing accuracy and 79.03% prediction accuracy.

Our model could help hospital managers predict whether a patient will come back in the future, therefore, managers can allocate their resources and facilities ready for the patient to avoid time conflicts. Moreover, we thought our model could still improve by adding some new variables. Since the hospital readmission rates for certain conditions are considered an indicator of hospital quality, and also affect the cost of care adversely. We could include such as Sanitation Level, Patients’ Satisfaction in the model.

### Section 1: Data Insights

We imported the train dataset ‘Hospitals\_Train.csv’ into R and found there are 5500 rows that are all entirely missing values, so we first directly deleted those rows from our dataset. Then for each column, we used count() function in plyr package to see if there were abnormal data or similar levels and then used appropriate methods to process them.

* For *HOSPITAL*, according to **graph 1**, we noticed that patients in different hospitals have similar readmission rates. We also did Chi-squared test to find out that there is no significant relationship exists on *HOSPITAL* and *RETURN* (The null hypothesis is that there is no relationship, and p-value is 0.6278). *HOSPITAL* is not informative to predict *RETURN*, so it is excluded.
* For *GENDER* column, there were few abnormal values. We deleted 2 rows of data with gender missing since two rows will not affect the general picture of the dataset.
* For *AGE*, we performed data normalization. Since the range of *AGE* is too large, it would probably increase the weight of the variable in our model. So we used min-max normalization to scale the variable to [0,1] range.
* We re-leveled the variable *FINANCIAL\_CLASS*. We visualized the variable (see **Graph 2**) and then searched the meaning of each healthcare plan and decided to put them into 6 groups because some healthcare plans refer to the same meaning. For instance, there are three private insurances, Blue Shield, Commercial, and Global Contracts which should be in the same group.
  + The first group named Private Insurances including Blue Shield, Commercial, and Global Contracts.
  + The second group contains MA MCO and Military, which are Government Insurance.
  + The third group is called Self-Pay.
  + The fourth group Medicaid referred to those insurance plans that especially help low-income families including Medicaid, Medicaid Pending and Out of State Medicaid.
  + The fifth group named Medicare which means healthcare plans that help elder or disabled people. It includes Medicare and Medicare Replacement Plan
  + The last group is named Others. It includes others and worker’s comp.
* By observation, we found that the data in *WEEKDAY\_ARR* and *WEEKDAY\_DEP*, *HOUR\_ARR* and *HOUR\_DEP* are exactly the same. And for *WEEKDAY\_ARR* and *WEEKDAY\_DEP*, we dropped these two columns since we think which weekdays patients arrived or departed are irrelevant. we decided to drop the column *HOUR\_APR* and *HOUR\_DEP* since by observation we found those two columns are identical. But, after careful consideration, we thought the specific time the patient arrived could affect whether the patient would return in the future. Therefore, we decided to keep and factorize *HOUR\_DEP* in our dataset.
* For *ED\_RESULT*, some levels have similar meanings. We named empty cells as ‘unknown’, named ‘Admit’ and ‘Observation’ as Admit, named ‘Admit to External Psychiatric Facility’ and ‘Admit to UMMS Psychiatry’ as Psychiatry, named ‘Send to L&D after Rooming’, ‘Send to L&D Before Rooming (Mom)’ as L&D.
* For *ACUITY\_ARR*, we dropped one row with abnormal value ‘5 purple’ because it does not appear in ‘Hospitals\_Test\_X’ csv file.
* For *DC\_RESULT*, we took a deep look at this column and found that *DC\_RESULT* might have the same feature as *ED\_RESULT* and which might cause Multicollinearity and redundant. So, we deleted this variable.
* For *CONSULT\_IN\_ED*, the missing value was corrected to 0 because the other value this column has is 1. We thought it is the dummy variable as stated in the dictionary, ‘1’ represents the patient requested consultation in the emergency department, ‘0’ represents the patient did not request.
* For *DIAGNOSIS\_DETAILS*, we thought whether the doctor wrote the words on the diagnosis would affect the patients returned. Therefore instead of considering it as a numeric variable, we decided to categorize *DIAGNOSIS\_DETAILS* into two groups. ‘0’ means the doctor did not write anything on the diagnosis. ‘1’ means the doctor wrote something on diagnosis.
* In the *RISK* and *SEVERITY* column, there were more than 33000 instances that leave the blank. We changed every missing value to ‘None’. By observation, we could see there is a clear relationship between these two columns (see **Graph 3**). We did the Chi-Square test to determine if there is a significant relationship between SEVERITY and RISK variables. The P-value is extremely small (< 2.2e-16), so we concluded that there is an association between these two variables. Considering Multicollinearity, we only kept *RISK.*
* For *CHARGES* column, we changed 100 ‘#Values’ observations to 0. Because patients with ‘#Values’ observations have similar characteristics with patients who were not charged.

|  |  |  |
| --- | --- | --- |
|  | Charges\_Level | Charges\_Return |
| 1 | NULL | 0.4700000 |
| 2 | 0 | 0.5270568 |
| 3 | Other | 0.2270331 |

Besides, we thought range across it was too large. Therefore we first took log of *CHARGES* using the base of 10, and used min-max normalization to scale the range to [0,1].

* Finally, for our prediction variable *RETURN*, we dropped 141 rows with the missing value because random assignment would introduce bias.

After data cleaning, our next step was to change the data type. We converted *HOUR\_ARR, CONSULT\_ORDER*, *CONSULT\_CHARGE* and *CONSULT\_IN\_ED* to factor, and converted *CHARGES* to a numeric variable.

Before building the model, we separated 25% of the instances in Hospitals\_Train as testing data. The rest of the part called Hospitals\_Rest. In Hospitals\_Rest, we split 25% of the instances as the validation data called Hospitals\_Valid, 75% of it as the training data called Hospitals\_Train. The most common class on training data and testing data are both ‘No’. Baseline accuracy on testing data is 0.7444059 (patients did not return to hospitals).

Then we considered if it is better to balance the whole dataset since this is an unbalanced classification problem. Our dataset is significantly unbalanced with about 75% of data labeled as non-returned patients. We did not want our models built on this dataset to be more likely to predict ‘No’ because of this imbalance, so we tried SMOTE method to artificially generate new examples of the minority class using the nearest neighbors of these cases. How we balance our data depends on the parameters in SMOTE method. For *perc.over* parameter, we chose 200, which would generate two duplicates for every data in the minority class, and we set *perc.under* to 150 so that it selects almost all the data from the majority class. With these two parameters, we can generate a more balanced dataset. The parameter *k* is a number indicating the number of nearest neighbors that are used to generate the new examples of the minority class. We created nine smoted datasets for Hospitals\_Train with value of k from 2 to 10 and ran a simple random forest model on each of the smoted dataset to choose the best k. But the testing accuracy for the model using smoted dataset is much lower than the accuracy using the original dataset. It probably because upsampling introduces bias. As the data balancing method did not achieve our anticipated goal, we gave up this method.

### Section 2: Modeling Insights

Using the input features, our goal was to create a prediction model that would accurately predict whether a patient would be readmitted within 30 days of being discharged. We compared the models built based on accuracy on test data. So far, we have tried six models to predict the future behavior of a patient.

#### 2.1 Logistic Regression

Logistic model can not only help us predict the binary classification problem, but also give us a general view of the inference at the very beginning. After splitting the data, we used hospitals\_rest to train the logistic model and used hospitals\_test as our testing data. Also, we ran the stepwise selection to filter the variables and calculate the accuracy. We found out the accuracy of stepwise selection logistic model was better than logistic model and the accuracy on our testing data is 76.22%, on prediction is 76.60%.

#### 2.2 LASSO

Lasso can help us delete or give us a guide to eliminate variables. Since Lasso method requires numerical variables, we directly used model.matrix to form a matrix of training data and test data for glmnet function since it was tough for us to turn every categorical variable into dummies. We used cross-validation procedure to train the model to avoid some uneven data distribution. Also, we got the best lamda number which could generate the minimum error. Then we passed the model we trained to testing data to calculate the accuracy on Lasso which is 76.25% and get prediction accuracy on 77.00%.

#### 2.3 Feedforward neural network

Feedforward neural network can learn its parameters and bias automatically which can have a more accurate model. As we did in Lasso, we first changed the processed data into matrix. In the neural network, we designed three hidden layers and one input layer and one output layer. Since the problem we solved is the classification one, we used softmax function in the output layer and we used sigmoid function in among three hidden layers and one input layer. More, we set loss function as categorical cross entropy, optimizer as adam and metrics as accuracy in the keras\_compile function. After figured out settings in the model, we used our training function to fit the model. In keras\_fit function, we set batch size as 32 and 20 epochs. Finally, we used feedforward model to test our testing data and get the accuracy which was 76.55%. And the prediction accuracy was 76.47%. Poor performance is probably due to the instability of our Feedforward neural network.

#### 2.4 Random forest

Compared to trees, random forest will have no bias on the variables selection. It considers all the variables. In random forest method, we changed our parameters in the function to make a better prediction. Since there were only two parameters in the random forest function, we loop the mtry and ntree arguments. We used range between 2 and 8 for mtry and 100-500 for ntree. In the prediction procedure, we found out that when mtry equals 4 and ntree equals 200 would output the highest accuracy. So, we changed them and generate the classification sheet. In our testing data and prediction, the corresponding accuracy is 76.61% and 77.69%.

#### 2.5 Boosting

Boosting method focuses on incorrect data. Each time it re-estimates the model and puts extra weight on the points that are misclassified. Theoretically, it is quite suitable for predicting readmission rate since it focuses on the minority (RETURN is ‘Yes’) in this unbalanced data set. We use 4 parameters to tune our model. Description and tuning range of parameters are as follows:

* n.trees: The total number of trees to fit, ranged from 100 to 500 with interval of 100
* shrinkage: Learning rate, ranged from 0.01 to 0.1 with interval of 0.01
* interaction.depth: Maximum depth of each tree, ranged from 1 to 30 with interval of 1
* n.minobsinnode: Minimum number of observations in the terminal nodes of the trees, ranged from 10 to 100 with interval of 10

The best combination of parameters is {n.trees: 300, shrinkage: 0.04, interaction.depth: 22, n.minobsinnode: 50}. Testing accuracy is 77.05% and prediction accuracy is 78.32%.

#### 2.6 XGBoost

XGBoost is a gradient boosting method that provides parallel computation to predict data in a fast and accurate way. XGBoost provides plenty of parameters. We evaluate the performance of the models given sets of parameter. Description and tuning range of parameters are as follows:

* booster: Booster type to use. We tried both gbtree and gblinear
* max\_depth: Maximum depth of a tree, ranged from 3 to 15 with interval of 1
* nrounds: Maximum number of iterations (maximum number of trees for gbtree), ranged from 20 to 100 with interval of 1
* eta: Learning rate, ranged from 0.1 to 0.5
* min\_child\_weight: Minimum number of instances in a child node, ranged from 1 to 10 with interval of 1
* subsample: Subsample ratio of training instance, ranged from 0.5 to 1
* colsample\_bytree: Subsample ratio of columns when constructing each tree, ranged from 0.5 to 1

The best set of parameters is {booster: gbtree, max\_depth: 10, nrounds=43, eta=0.169, min\_child\_weight=2.13, subsample=0.789, colsample\_bytree=0.542}. Testing accuracy is 77.51% and prediction accuracy is 79.03%.

### Section 3: Conclusions and Recommendations

In general, within this case, we processed data first since there were many levels in several columns and a lot of empty or blank cells in the datasets. For categorical variables with similar levels, we re-leveled the column values to decrease the number of levels. For empty or blank cells, we basically classified those values as ‘Unknown’ for future analysis. For numeric variables, we performed normalization to avoid different weights of variables. We also dropped some useless variables for prediction. After that, we divided data into three parts: training, validation, and testing. We used training data to build the model, used validation data to choose the best parameters, and used accuracy to compare models based on testing data.

Then we trained different models to see which one performs the best in predicting whether a patient will return to the hospital within 30 days of being discharged. Among six models we ran, XGBoost method is proved to has the best accuracy. The model built using this method has test accuracy of 77.51% and prediction accuracy of 79.03%. Though data cleaning and feature selection are crucial, parameter tuning is the most important factor of improving prediction accuracy.

For hospitals, managers can make full use of the results and models we made in this project. As far as we think, the hospital managers can collect the patients’ features according to the variables and put them into the models we designed to get a convincible result. If the hospital can predict whether a patient will come back in the future, managers can manage their resources and facilities ready for the patient to avoid time conflicts. Moreover, managers can cut the expenses on doctors’ salaries and depreciation on facilities if there is only a little chance that the patient will come back and pursue future treatment. However, the models and results can only give hospitals a general view of the probability of whether a patient will come back. Hospitals should always be ready for patients and make them feel they are considered.

Moreover, We thought it still had some improvement spaces we could work on. For instance, We all know that the hospital readmission is when a patient who is discharged from the hospital, gets re-admitted again within a certain period of time. Hospital readmission rates for certain conditions are now considered an indicator of hospital quality, and also affect the cost of care adversely. In the variable analysis, may we could add the Sanitation Level, Patients’ Satisfaction, the gap between estimated charges and actual charges and etc.. These variables may provide us more accurate results since these are more related to the definition of hospital readmission.

### 

### Roles and Responsibilities

Zhenhao Chen: Boosting, XGBoost

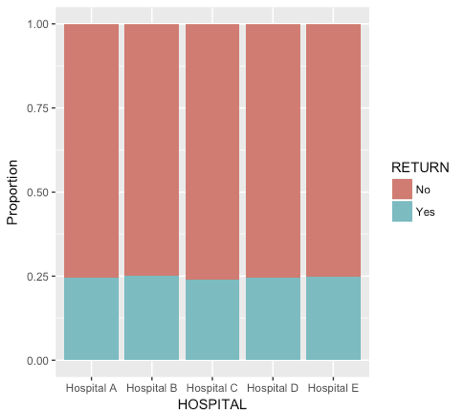
Mingfeng Shen: Logistic Regression, LASSO, Ridge, Random Forest, Feedforward Neural Network.

Jingtian Li: Data Cleaning and Variables Analysis

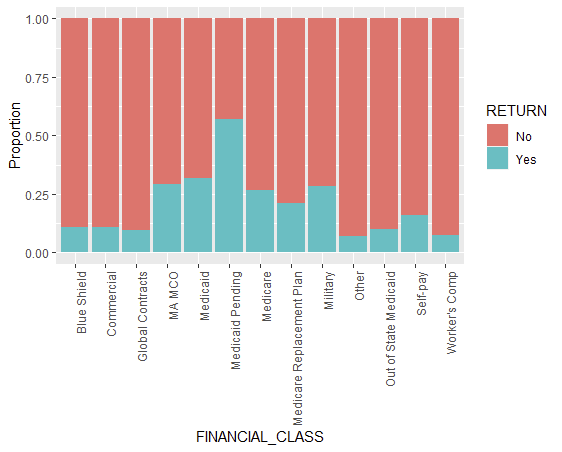
Ziyang Dong: Data visualization, Data Balancing(SMOTE), Tree, Bagging

### Appendix

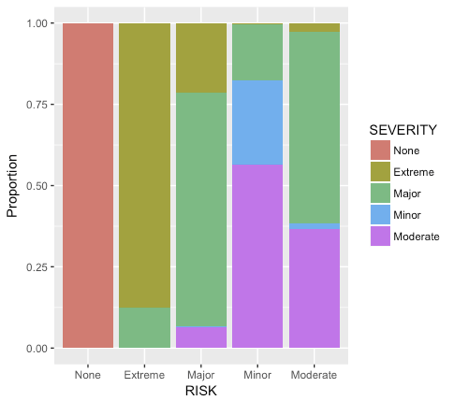
#### Graph



Graph 1



Graph 2



Graph 3

#### Submission Description

Submission Summary

The baseline for our test data is 74.4406%.

The baseline for prediction data is 75.4488%.

The method, test accuracy and prediction accuracy for each submission are shown as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Submission | Method | Test Acc | Prediction Acc |
| 1 | Logistic | 76.22% | 76.60% |
| 2 | Boosting | 76.39% | 77.29% |
| 3 | Random Forest (n=500) | 76.61% | 77.69% |
| 4 | Random Forest (n=5000) | ≈76.61% | 77.72% |
| 5 | LASSO | 76.25% | 77.00% |
| 6 | Blend (cutoff=0.2) |  | 77.58% |
| 7 | Feedforward | 76.55% | 76.47% |
| 8 | Random Forest (50 models) | 76.61% | 77.83% |
| 9 | Blend (cutoff=0.5) |  | 77.60% |
| 10 | Random Forest (100 models) | 76.95% | 77.98% |
| 11 | Tuned Boosting (100 models) | 77.05% | 78.32% |
| 12 | Tuned XGBoost (100 models) | 77.51% | 78.94% |
| 13 | Tuned XGBoost | Bad | 78.46% |
| 14 | Tuned XGBoost (940 models) |  |  |
| 15 | Tuned XGBoost (1000 models) |  |  |
| 16 | Blend (cutoff=0.5) |  |  |
| 17 | Tuned XGBoost |  |  |

\*The sets of parameters for Tuned XGBoost are different.

Best Prediction Accuracy

XGBoost, a blend of 1000 models

Tuning Result:

[Tune] Started tuning learner classif.xgboost for parameter set:

Type len Def Constr Req Tunable Trafo

booster discrete - - gbtree - TRUE -

max\_depth integer - - 3 to 15 - TRUE -

nrounds integer - - 20 to 100 - TRUE -

eta numeric - - 0.01 to 0.5 - TRUE -

min\_child\_weight numeric - - 1 to 5 - TRUE -

subsample numeric - - 0.5 to 1 - TRUE -

colsample\_bytree numeric - - 0.5 to 1 - TRUE -

With control class: TuneControlRandom

Imputation value: -0

Exporting objects to slaves for mode socket: .mlr.slave.options

Mapping in parallel: mode = socket; cpus = 4; elements = 1000.

[Tune] Result: booster=gbtree; max\_depth=10; nrounds=43; eta=0.169; min\_child\_weight=2.13; subsample=0.789; colsample\_bytree=0.542 : acc.test.mean=0.7856112

#### Code