Case Study Seven

Allen Hoskins, Brittany Lewandowski

December 5, 2022

1 INTRODUCTION

In the field of data science, working with anonymized data is commonplace. Although anonymized data obscure feature names, data science remains a practical application for predicting and classifying business outcomes. Common use cases for anonymized data are in healthcare when patient data is used, and in business, when confidentiality is paramount.

In this case study, we use an anonymized data set, final\_project(5), provided by Dr. Slater to perform binary classification on our response variable, ‘y.’ While our client has indicated that the feature names are not important for us to know, they also shared that there is a cost of $100 for every false positive, and $20 for every false negative predicted. Since this misclassification is costly, our client indicated that our model must minimize misclassification costs.

The rest of this case study focuses on building 3 machine learning models: CatBoost, Random Forest, and Naïve Bayes. The intention is to find which of these 3 models performs the best and reduces costs associated with misclassification for our client. Our hope is that this research can help our client to operate more efficiently from both a business and financial perspective.

2 METHODS

***DATA UNDERSTANDING:***

Data used in this case study was a ‘final\_project(5)’ csv provided by Dr. Slater. This .csv contained 160,000 rows and 51 columns with feature names including: ‘x0’, ‘x5’, and ‘y.’ Other observations we made from looking at our data set included:

1. Given that our response variable, ‘y’ contained binary values, this was a binary classification problem.
2. Although our data set was large, it was not too large for algorithms including Random Forest & Naïve Bayes.

Upon reviewing the contents of our data set and gathering our initial observations, we saved it in a data frame, df, and began our pre-processing.

***DATA PREPROCESSING:***

The first step we took in pre-processing was running the command ‘df.info()’ to view the data types of each of the columns in our data set. Three takeaways we had from running that command were:

1. 47 of our columns were floats
2. 1 of our columns was an integer
3. 3 of our columns were objects

Seeing that 3 of our columns were categorical data types, we recognized that we would need to one hot encode those values for Random Forest & Naïve Bayes to run. For more details on this process, please see our sub-header below titled ‘One Hot Encoding.’

Once we reviewed the data types of our columns, we checked our data for null and missing values. In doing so, we found that many of our columns contained missing values. This indicated that we would need to impute these values using common statistical techniques including mean, and median imputation methods. A full explanation of how we imputed our data can be found in the sub-header ‘Data Imputation’ below.

Before proceeding to data imputation, we took one final look at our data frame. When doing this, we found that several columns contained special characters that our models would not be able to process. Columns that contained these values were ‘x32’ and ‘x37.’ Our ‘x32’ feature contained percentage signs, and ‘x37’ contained dollar signs. Recognizing that we needed to strip our data of those special characters, we wrote a function, parseFloat, which removed the percentage sign from feature ‘x37’ and converted the row values to decimal values. For our ‘x32’ feature, we wrote a lambda function that replaced the dollar signs in all the records in that column.

Other small syntactical changes we made to our data include that we replaced the ‘NaN’ values in our categorical variables with ‘Unknown’ and we renamed the values in ‘x29, to be representative of a month’s full name (i.e., September). With our data formatted correctly, we proceeded to re-code our categorical columns.

***RE-CODING CATEGORICAL COLUMNS:***

After reviewing our data set and re-formatting it, we began re-coding our categorical columns. When viewing the shape of our categorical columns, we recognized that if we one hot encoded these columns as-is, our data set would be wide. To avoid this, we re-coded our categorical columns by specifying a threshold for infrequent classes. Exhibit 1.0 specifies the threshold set for each column and provides explanations as to why those thresholds were chosen. Please note that re-coding of our categorical features was done for our Random Forest & Naïve Bayes models. This re-coded data was not used for our CatBoost model as this model does not require it.

|  |  |  |
| --- | --- | --- |
| Column Name | Threshold Set | Explanation |
| x24 | .027 | ~95% of our data falls into the top 3 classes |
| x29 | .015 | ~95% of our data falls into the top 7 classes |
| x30 | .174 | ~98% of our data falls into the top 7 classes |

Exhibit 1.0 – Thresholds set for categorical columns to simplify OHE processing

***ONE HOT ENCODING:***

Since Random Forest & Naïve Bayes are largely unable to process categorical features, we needed to one hot encode the categorical columns. To accomplish this, we first separated our categorical columns into a data frame named ‘cat\_df’ and used pandas’ get\_dummies function to one hot encode our categorical variables. Once this finished, we joined our ‘cat\_df’ data frame with our numeric data and saved the results into a data frame named ‘df\_wohe.’ With our categorical features converted, we moved on to imputing our missing values. Please note that one hot encoding of our categorical features was done for our Random Forest & Naïve Bayes models. This one hot encoded data was not used for our CatBoost model.

***DATA IMPUTATION:***

Given that all our columns now contained numeric values, we elected to use the methods of mean and median imputation for our modeling. To impute our missing values, we copied our ‘df\_wohe’ data frame, saving it to two new data frames, named df\_mean2 and df\_median2 respectively. With these new data frames, we created a new column, impute\_col, that contained concatenated values for each of our categorical variables. After creating this column, we grouped our data frame by our ‘impute\_col’ feature and wrote a lambda function to replace missing values in our data with mean and median values. Once our lambda function completed, we dropped our ‘impute\_col’ from our data frame.

The same imputation methods described in this section were also applied to our original data frame, ‘df’, that did not contain one hot encoded data. The decision to impute missing values on non-one hot encoded data was made given that CatBoost was one model that was built in this case study. Since CatBoost’s strengths are that it can handle categorical values, we supplied it with our original imputed data set.

***EXPLORATORY DATA ANALYSIS:***

With our full data set intact, we began our exploratory data analysis. First, we viewed the distribution of our response, ‘y,’ to view if class imbalance was present. Output from the count plot we created, Exhibit 1.1, showed that our response had fewer values of 1 than it did for 0. The imbalance shown here was something we remained cognizant of throughout the rest of our analysis.

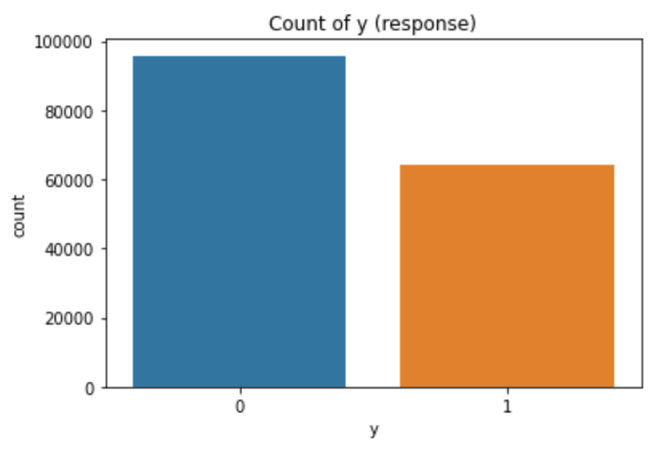


Exhibit 1.1 – Distribution of our response variable, ‘y’

After viewing the distribution of our response variable, we created distribution plots for the rest of the features in our data frame. Results from this showed that many of our variables had normal distributions (Exhibit 1.2 & 1.3). With this discovery, we decided to proceed with our analysis without applying any additional data transformations.

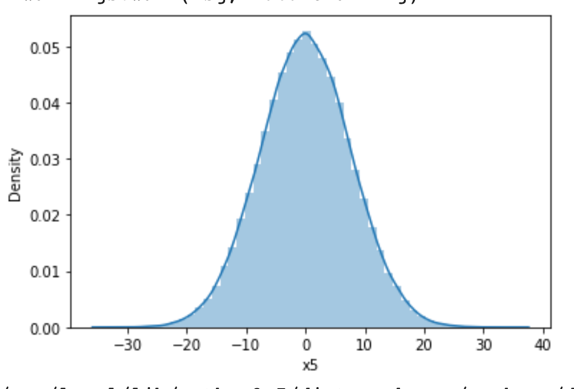
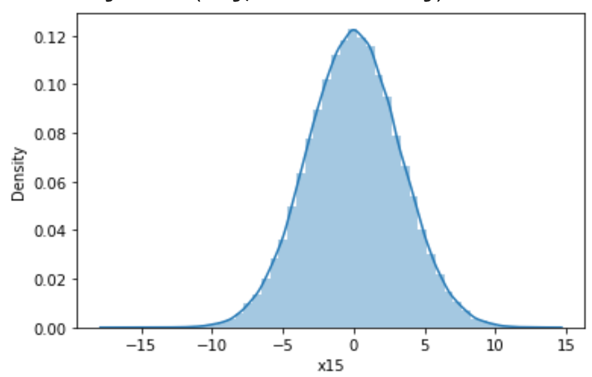
 

Exhibit 1.3 – Distribution of feature ‘x15’

Exhibit 1.2 – Distribution of feature ‘x5’

The last step we performed in our exploratory data analysis was evaluating if multicollinearity existed in our data set. To do this we created a correlation plot and found that feature ‘x41’ had a correlation coefficient of 1. Knowing that highly correlated features can skew modeling results, we removed the feature from our data frame and proceeded to model building.

4 MODEL BUILDING

Since our client asked us to develop the best binary classification model that minimizes misclassification, we decided to use three different algorithms: CatBoost, Random Forest, and Naïve Bayes. Please note that since CatBoost can handle categorical features and missing values, we used our original data set without one hot encoded variables for model building.

Our decision to use these models was based on the following reasons:

1. All algorithms are fit for binary classification tasks.
2. CatBoost can handle categorical features and missing values, both of which were present in our data set.
3. Random Forest is a standard classification algorithm that can be used to determine a base level of accuracy you should strive for when fitting models to your data.
4. Naïve Bayes is recognized as a good classification algorithm that helps prevent overfitting.
5. By using both standard and boosting classification algorithms, we will be able to find the best model for our data.

Once we decided what models we were building, we separated our feature columns from our response, split our data into an 80/20 train/test split, and fit our models to our mean and median imputed data sets. Since each of our models evaluate different sets of hyperparameters, exhibits 1.4 & 1.5 below display the parameters that we passed to each one. Please note that our CatBoost model took a list of our categorical features, ‘cat\_feat\_mean’ for processing.

Catbooost Hyperparameters:

|  |  |
| --- | --- |
| Parameter | Value Passed to Parameter |
| loss\_function | logloss |
| custom\_metric | AUC, Accuracy, F1 |
| verbose | 200 |
| random\_seed | 0 |
| learning\_rate | .25 |
| thread\_count | -1 |
| early\_stopping\_rounds | 200 |

Exhibit 1.4– Parameters Passed to CatBoost Models

Random Forest Hyperparameters:

Exhibit 1.5 – Parameters Passed to Random Forest Models

|  |  |
| --- | --- |
| Parameter | Value Passed to Parameter |
| random\_state | 0 |
| n\_jobs | -1 |
| criterion | Gini |
| n\_estimators | 100 |
| max\_depth | 10 |
| max\_features | Sqrt |
| oob\_score | True |
| bootstrap | True |

Although our Naïve Bayes model contained no hyperparameters, our data was not on the same scale. Consequently, we applied a Standard Scaler to our data prior to modeling. This was done to ensure that the results from this model were representative of our data.

5 RESULTS

Exhibits 1.6.1-2.4.8 below display our model results as well as misclassification costs associated with each model. Our best-performing model was the CatBoost model with mean imputed data. Not only did this model have the lowest total misclassification cost of $111,460 with an average cost per prediction of $3.48. The CatBoost model also had the best overall accuracy, 94%, and F1 score, 95%.

Random Forest Mean Imputed Data Results:

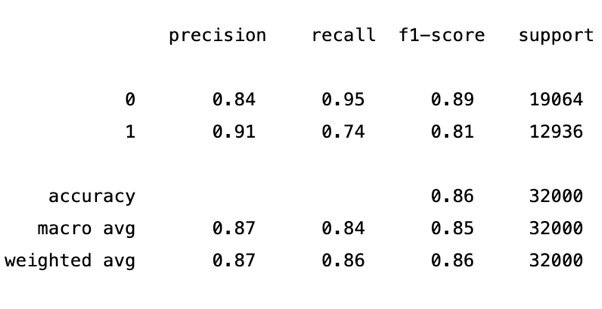


Exhibit 1.6.1 – Random Forest Mean Imputed Data Model Classification Report

Chart, treemap chart

Description automatically generated

Exhibit 1.6.2 – Random Forest Mean Imputed Data Confusion Matrix

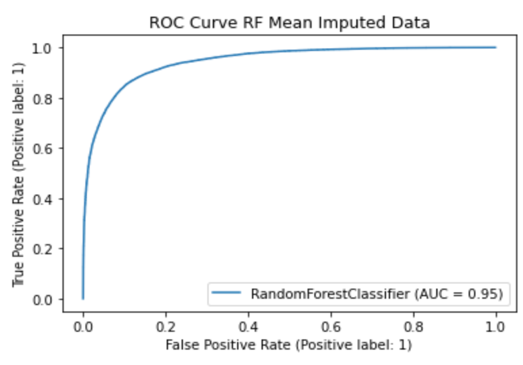


Exhibit 1.6.3 – Random Forest Mean Imputed Data ROC Curve

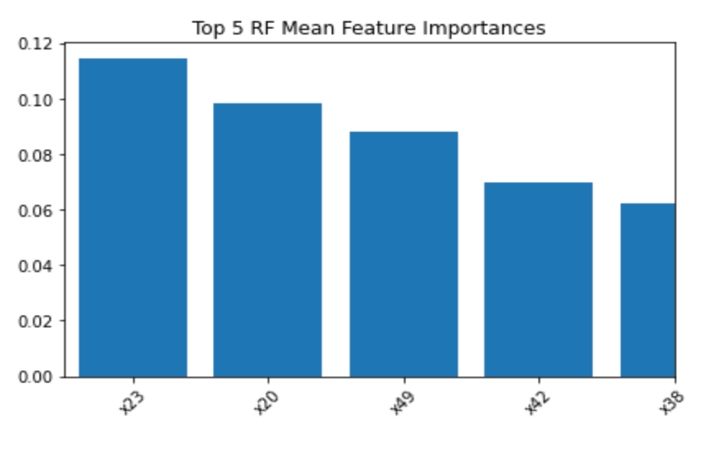


Exhibit 1.6.4– Random Forest Mean Imputed Data Feature Importance

Random Forest Median Imputed Data Results:

Table

Description automatically generated

Exhibit 1.7.1 – Random Forest Median Imputed Data Classification Report

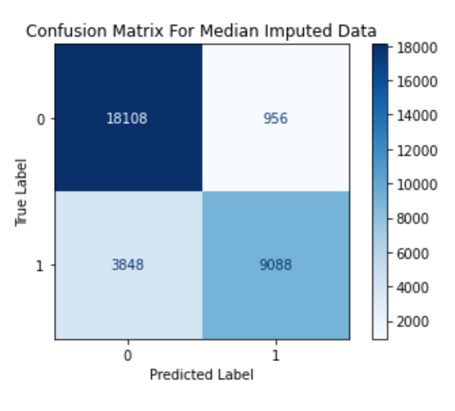


Exhibit 1.7.2 – Random Forest Median Imputed Data Confusion Matrix

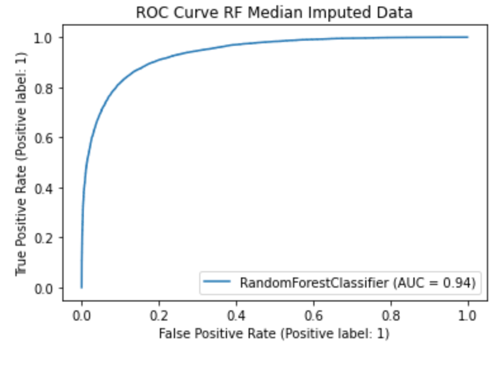


Exhibit 1.7.3 – Random Forest Median Imputed Data ROC Curve

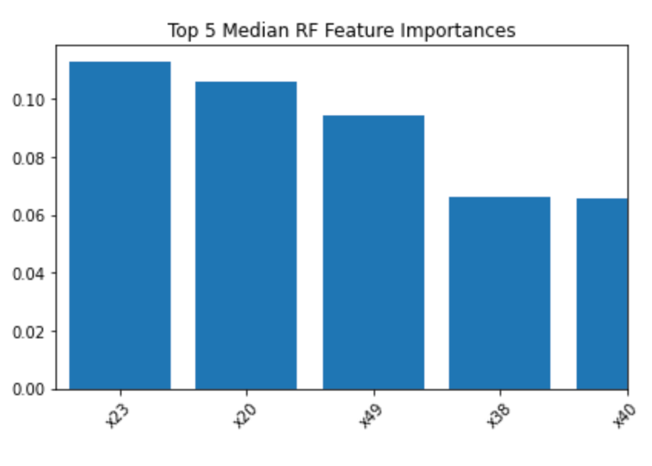


Exhibit 1.7.4– Random Forest Median Imputed Data Feature Importance

Naïve Bayes Mean Imputed Data Results:

Table

Description automatically generated

Exhibit 1.8.1 – Naïve Bayes Mean Imputed Data Classification Report

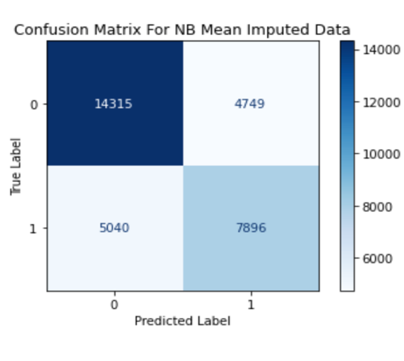


Exhibit 1.8.2 – Naïve Bayes Mean Imputed Data Confusion Matrix

Exhibit 2.0 – Naïve Bayes Mean Imputed Data Model Results

Naïve Bayes Median Imputed Data Results:

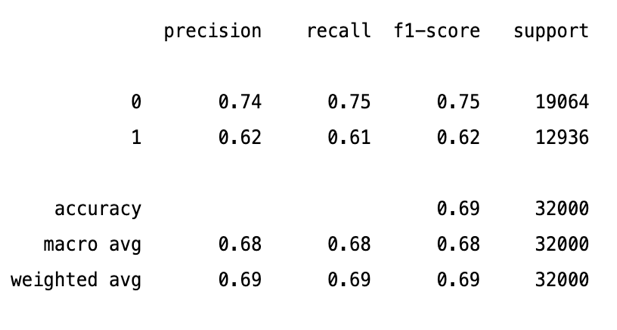


Exhibit 1.9.1 – Naïve Bayes Median Imputed Data Classification Report

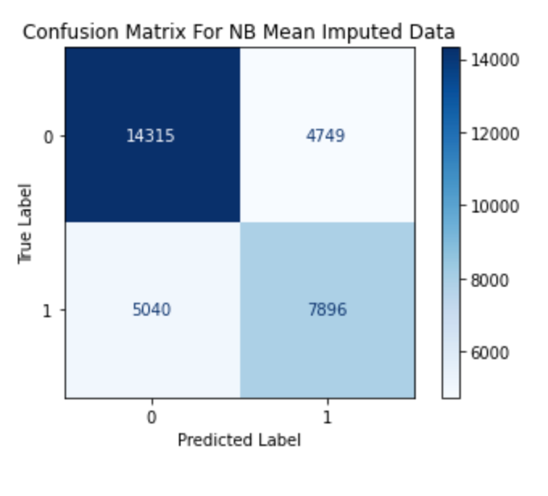


Exhibit 1.9.1 – Naïve Bayes Median Imputed Data Confusion Matrix

CatBoost Mean Imputed Data Results:

Table

Description automatically generated

Exhibit 2.1.1 – CatBoost Mean Imputed Data Classification Report

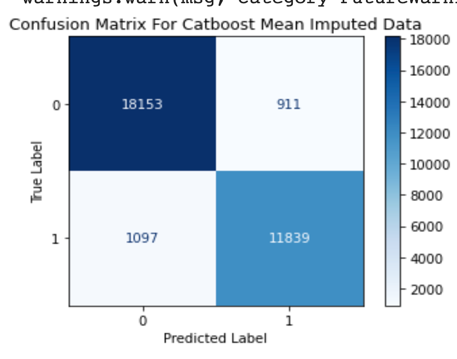


Exhibit 2.1.2 – CatBoost Mean Imputed Data Confusion Matrix

CatBoost Median Imputed Data Results:

Table

Description automatically generated

Exhibit 2.2.1 – CatBoost Median Imputed Data Classification Report

Chart, treemap chart

Description automatically generated

Exhibit 2.2.2 – CatBoost Median Imputed Data Confusion Matrix

CatBoost Drop Data Results:

Table

Description automatically generated

Exhibit 2.3.1 – CatBoost Drop Data Classification Report

Chart, treemap chart

Description automatically generated

Exhibit 2.3.2 – CatBoost Drop Data Confusion Matrix

Misclassification Costs For CatBoost Models:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False Negative Cost | False Positive Cost | Total Misclassified | Total Cost | Avg. Cost Per Prediction |
| $21,940 | $91,100 | 2,008 | $113,040 | $3.53 |

Exhibit 2.4.1 – CatBoost Mean Imputed Data Misclassification Costs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False Negative Cost | False Positive Cost | Total Misclassified | Total Cost | Avg. Cost Per Prediction |
| $22,460 | $89,000 | 2,013 | $111,460 | $3.48 |

Exhibit 2.4.2 – CatBoost Median Imputed Data Misclassification Costs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False Negative Cost | False Positive Cost | Total Misclassified | Total Cost | Avg. Cost Per  Prediction |
| $21,960 | $97,100 | 2,069 | $119,060 | $3.76 |

Exhibit 2.4.3 – CatBoost Drop Data Misclassification Costs

Misclassification Costs For Random Forest Models:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False Negative Cost | False Positive Cost | Total Misclassified | Total Cost | Avg. Cost Per  Prediction |
| $67,380 | $95,600 | 4,353 | $162,980 | $5.09 |

Exhibit 2.4.5 – Random Forest Mean Imputed Data Misclassification Costs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False Negative Cost | False Positive Cost | Total Misclassified | Total Cost | Avg. Cost Per  Prediction |
| $76,960 | $96,600 | 4,804 | $173,560 | $5.42 |

Exhibit 2.4.6 – Random Forest Median Imputed Data Misclassification Costs

Misclassification Costs For Naïve Bayes Models:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False Negative Cost | False Positive Cost | Total Misclassified | Total Cost | Avg. Cost Per  Prediction |
| $100,800 | $474,900 | 9,789 | $575,700 | $17.99 |

Exhibit 2.4.7 – Naïve Bayes Mean Imputed Data Misclassification Costs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False Negative Cost | False Positive Cost | Total Misclassified | Total Cost | Avg. Cost Per  Prediction |
| $100,800 | $474,900 | 9,789 | $575,700 | $17.99 |

Exhibit 2.4.8 – Naïve Bayes Median Imputed Data Misclassification Costs

CONCLUSION

Altogether this case study helped us gain experience working with anonymized data and gave us the opportunity to apply all the knowledge gained in this course. Additionally, this case study challenged us by giving us full autonomy to determine which models were most appropriate for our client’s needs.

Seeing that our CatBoost model performed the best and had the lowest total misclassification cost, we would suggest that our client utilizes the CatBoost model with our tuned parameters. The results of the model showed how boosting helps to reduce bias and errors in our models.

Two changes we would incorporate in this case study if we were given additional time include, upsampling class 1 of our response, and fitting additional models such as SVM or Neural Networks. Since our response variable showed evidence of being unbalanced, we would be interested to see how balancing the feature would impact model results. Similarly, since SVMs are known to be powerful classification models, we would be interested to see how it performs on our data set. The additional Neural Network model would allow us to potentially utilize pre-trained models and the increased power of the network.