Predicting Impella Use in Cardiovascular Surgery Patients

Macmillan Jacobson

April 2023

1 Abstract

In this paper, we analyze a data set consisting of 42 independent clinical variables in order to see if any of these variables are correlated and predictive of the use of an Impella in cardiovascular surgery patients. Furthermore, we compare and contrast the efficacy of three classification algorithms, Naive Bayes, KNN and random forest. We do this comparison by looking at both the model's accuracy as well as its minimization of type II error. In building our models, KNN and random forest classifiers are tuned using 5-fold random search cross validation. Additionally, we employ SMOTE to compensate for the unbalanced nature of our dependent variables classes. The resulting models suggest that random forest outperforms the Naive Bayes and KNN algirthms in its predictive power, achieving a 98.7% accuracy compared to lower accuracy scores for the other two algorithms while also achieving the minimum type II error. Random forest did much better in its distribution of predictions, not simply leveraging the unbalanced class set and predicting no-Impella to achieve an artificially high accuracy score, which is what KNN did. The final result of the analysis below is that the most predictive variables, in order, are length of stay in the hospital, number of days spent in the ICU and whether or not the surgeon used the radial artery as the access site for the surgery. These importances were found by leveraging the way that the random forest classifier is built.

2 Introduction

In this report we will be examining the effects of several clinical variables on the use of an Impella in cardiovascular surgeries. An Impella is a family of medical devices that are used for temporary ventricular support in patients with depressed heart function which are not always necessary for all surgical patients. It is a well known fact among hospital administrators that Impellas are extremely costly to use in surgeries. As such, administrators would like to know whether or not an Impella is predicted to be used in a surgery before the surgery takes place in order to better allocate resources in the hospital. Specifically, they are interested in finding out when an Impella might be used as opposed to when it probably wont be used. The statistical analysis of this problem boils down to a simple classification problem. To address this classification problem, several learning algorithms will be trained to predict Impella use. The efficacy of each of these classification algorithms will be compared and contrasted. Specifically, the algorithms to be used are random forest, Naive Bayes, and KNN, each of which will be tuned and trained. In the sections to follow, we will define our problem statements, explain where the data is gathered, cover the specific methodology used to train the algorithms, review the results and draw conclusions based on those results.

3 Problem Statement and Data Sources

3.1 Problem Statement

The goal of the analysis covered in this report is to predict Impella use in cardiovascular surgery patients based on several pre-surgical and intra-surgical clinical variables including, but not limited to, length of stay, access site, age, race, gender, medical history and use of medication. We are specifically interested in which independent variables are most important in prediction of Impella use. In doing these classifications, we would like to minimize type II (false-negative) misclassifications. In minimizing false negatives, we would be minimizing the number of times that the hospital that is interested in using this predictor under allocates resources for an Impella. In addition to information regarding our data set specific outcomes, we are also interested in comparing and contrasting the efficacy of the three classification algorithms we will use for this analysis.

3.2 Data Source and Exploration

The data used in this report is gathered from Biome Analytics. The author spent several years working as a data scientist at this company and found symbiosis in the ability to analyze this data set for both Georgia Tech and Biome Analytics. Over the past several decades, hospitals have been gathering and storing millions of data points concerning both cost and outcome information for cardiovascular surgery patients. Most hospitals do not have the resources required to clean and analyze this data in order to draw meaningful conclusions from it. This is where Biome Analytics has made its business, the company works with hospitals to collect their data, clean it, analyze it and present the information gathered from such analysis back to the hospital administrators and surgeons in a meaningful way. The author has worked with Biome to gather a subset of this data that concerns cases where an Impella was used and cases where an Impella might be used.

The data set contains 42 columns including personal characteristics (age, race, gender), clinical variables (tobacco use, stroke history) and hospital stay information (length of stay, number of days in the ICU) among others. Of our 18020 (N) cases, only 542 indicate that an Impella was used. In the preliminary examination of the variables, some multicollinearity was found between the variables. It was noticed that other columns in then data set captured all of the necessary information for the desired analysis, so many columns from the original data set were removed. The resulting correlation matrix can be found in the Appendix in Figure 1. Additionally, readers can find a complete list of our independent variables in the Appendix under Figure 3 including the datatype of each variable.

Of the 42 predicting variables we are using, 38 of them are categorical, and 6 of those are non binary. Specifically, the non-binary variables in our dataset are hospital, procedure type, admission day of the week, zip code, tobacco use, summary payor (insurance) type and access site. Notably, 99.5% of our access sites are femoral or radial. Similarly, 99% of the procedure types were either PCI+Cath or just PCI. The plurality (30%) of the summary payor class variable of the patients in our dataset were

on Medicare. Finally, 90% of our patients either never use tobacco or are former users. The rest of our non-binary categorical variables seem to be generally uniformly distributed over their sample spaces.

Of our binary variables, we note that most are relatively well balanced, with the minority class accounting for over 10% of data points. Some are quite sparse, accounting for less than 5% of our cases. Notably, there seem to be few cases with stroke history with less than half of a percent of cases indicating a history of strokes. As for our racial data, we can see that over 96% of patients identified as white, 12% identify as hispanic and just over 1% identify as black. Note that some patients identify as several racial categories.

4 of our independent variables are continuous; length of stay, ICU days, distance from hospital and age. For both length of stay and ICU days, the vast majority of patients stayed for under 2 days before their surgeries, however some stayed much longer. That is each of these distributions, in addition to distance from hospital, resemble a lognormal distribution. ICU days, length of stay and distance from hospital have respective means of 3.5 days, 43 days and 56 miles. Though their medians are much lower at 1.75 days, 0 days and 14.9 miles suggesting that there are significant outliers in our dataset. We will not remove these outliers as we want to capture the extreme cases as they may be more likely to result in Impella use. Our age variable, expectantly, is generally normally distributed around a mean of 67 years old with a standard deviation of around 10 years.

4 Proposed Methodology

The solution to our analysis requires use of classification algorithms. Perhaps the most popular classification algorithm in modern days is the random forest. The random forest algorithm is an ensemble learning method that operates by constructing several simple decision trees at training time and aggregating the results of each of these sub trees by a voting system. Because these individual decision trees do not all contain each of the independent variables, we can also learn which of the independent variables are most predictive in our classification. In addition to the random forest, we will also employ the use of two less popular algorithms and compare their efficacy. First, we will be using KNN which finds the K most similar, "nearest-neighbor," cases to a given data point and in essence takes a plurality vote from these K most similar cases to make a classification. Finally, we will use Naive Bayes (NB). The NB classifier is based on Bayes' Theorem which takes into account prior knowledge and assumptions on the distribution of our predicting variables. The algorithm then uses these distribution assumptions to aggregate and predict the class based on a probability sample.

The choice of these three classifiers is by design. Part of the goal of this report is to compare the efficacy of each of these methods in a real world sample. The random forest classifier is extremely popular in modern machine learning. The reason for this is two fold, not only does random forest perform extremely well in classification models, only rivaled by neural networks in its accuracy, but it also provides a logical and easily understood way to determine the importance of each variable in its prediction. The second two choices are much less popular. The KNN algorithm is relatively un-

sophisticated in its design, but does perform quite well under the right circumstances. Similarly, NB is also antiquated and often considered less sophisticated than modern machine learning techniques. Thus, is made sense to pick these two less sophisticated methods to compare to our more modern random forest in order to confirm or deny the efficacy of the more popular algorithm.

The hyper parameters for both the random forest classifier and the KNN classifier will be tuned using a 5-fold cross validation scheme. Each of these folds will be generated by randomly sampling from our data set. Furthermore, we will use a 20-80 test-train to validate the efficacy of our models.

Unfortunately, our data set is quite lopsided in its classes as described above. As such, we will need to employ some methods to help alleviate this problem. The method to be used is synthetic minority oversampling technique (SMOTE). SMOTE works by generating new, synthetic, data points that are similar to those found in the data set. We then stratify our test train split using both the original data and the new, synthetic, data on our classes. This ensures that there is a more balanced representation of our two classes as well as a more well distributed representation of our classes in both our training and testing test. This allows our classifiers to more accurately predict Impella use. If this method was not used, each classifier could simply assign no Impella use to each class and achieve an extremely high prediction accuracy. After training the classifiers on this over-sampled data set, we then test our efficacy on a normal, not over-sampled, test train split. We do this to test our models on more realistic distributions of classes and thus gain a better picture of each model's accuracy.

In comparing our classifiers, we will focus specifically on accuracy, and moreover minimizing type II error. It is by these two metrics that we will decide which predictor achieves the best results.

5 Analysis and Results

To achieve all of the above proposed methodology, we used Python's SKLearn package. SKLearn contains everything a user would need to train all three of the models above including the tuning of their hyperparameters and implementation of SMOTE. First, we trained our KNN model. In our hyperparameter tuning, we found that K=26 resulted in the lowest MSE during cross validation. Notably, when looking at the accuracy of our model on our testing set, the KNN classifier achieved an accuracy score of 97% which was quite surprising. However, when displaying the confusion matrix, the reason the KNN model performed so well became clear. Since we had such unbalanced classes, the KNN model was simply assigning no-Impella to each tuple. The same happened when using SMOTE to over-sample and retrain. Obviously, the accuracy score, though extremely high, is also misleading and the model did not do very well. Specifically, the KNN reported only type II misclassifications. Below is the confusion matrix for our 26-NN classifier.

		Predicted	
		0	**1**
True	**0**	3496	0
	1	108	0

The second classifier we train is our NB model. Again, the accuracy score from the resultant model is quite high at 93%, though not as high as KNN. When we take a look at the confusion matrix we note that it does not have the same issue that was apparent in the KNN model. This is encouraging. Even though the accuracy score is worse for this model, the distribution of predictions is much better, though there is still quite a bit of type II error. Below is our confusion matrix for the NB classifier.

		Predicted	
		0	**1**
True	**0**	3312	184
	1	65	43

Finally, we train our random forest classifier. We expect this one to do much better, and it does. In our hyperparameter tuning under SMOTE, we find that the optimal max depth is 16 for each tree while the optimal number of estimators is 350. These hyperparameters resulted in the lowest MSE during our 5-fold cross validation. Our SMOTE random forest achieved an accuracy score of 98.7% which is extremely high. Additionally, we can notice that the SMOTE assisted random forest minimised type II error which is one of the two metrics we will measure the classifiers by. The confusion matrix is displayed below.

		Predicted	
		0	**1**
True	**0**	3467	29
	1	18	90

In training our random forest, we also gained a very easy way of determining which of our independent variables were the most predictive in the use of an Impella. The three most significant variables, in order, are length of stay, ICU days and use of a radial access site for the surgery. It seems that the longer the patient stays in the hospital (inside or outside the ICU), the greater the chance that an Impella will be used. Though this may be uninformative, since sicker patients need to stay in the hospital longer and are more likely to need heart support during a surgery. What is more interesting is that the use of a radial access site seems to be correlated with the use of an Impella given that the radial access site is generally considered to be a safer alternative to other access sites. The complete results of the feature importances can be found in the Appendix under Figure 2.

6 Conclusion

In the analysis above, we have shown how a random forest classifier can be constructed to predict Impella use in cardiovascular surgery patients with an accuracy of almost 99% while minimizing type II error. In the construction and training of this model, we have found that the three distinct variables that most well predicted the use of an Impella to be the length of hospital stay, time spent in the ICU and whether or not the access site for the surgery was the radial artery. Furthermore, when comparing the

random forest to two simpler classification methods, NB and KNN, random forest was found to be superior in both accuracy and distribution of predictions. Several future studies could be conducted based on these results. It would be interesting to adjust for risk factors. For instance, it is possible that Impella use is simply used for sicker patients and sicker patients tend to stay in the hospital for longer. Similarly, doctors may prefer a radial access site for sicker patients. These correlations may be just that, correlation, rather than causation. Further analysis must be conducted to see if this is the case. It would also be interesting to compare how random forest does with a more modern deep learning solution, but again, further analysis is required.

6.1 Lessons Learned

I learned quite a bit during this exploration. First and foremost, I learned how to apply SMOTE to an unbalanced data set. Unbalanced data sets have always plagued my work in both my professional and academic lives. It was quite interesting to find an elegant and easily implemented solution to this problem. In my research, I also came across different implementations of the NB algorithm. In the analysis described above, I simply used the Gaussian NB algorithm. In digging a little deeper, I found that the Gaussian NB algorithm works well for continuous independent variables, but our data set consists of mostly boolean variables. As such, in future analysis and review, it would make more sense to use a multinomial or binomial NB algorithm for our data set.

I also dug a little deeper into the way that random forests work in determining the feature importances. It became clear that random forests are biased toward weighting continuous variables higher in the feature importance charts. This means that length of stay and ICU days, described above, being continuous variables may be artificially shown to be higher than their true importance. Further analysis would be required to see if this it the case.

Finally, I learned the importance of keeping the end user of our analysis in mind while building the models. In focusing on the minimization of type II error, I was able to more accurately distinguish between strong and weak classifiers and report the correct results.

7 Appendix

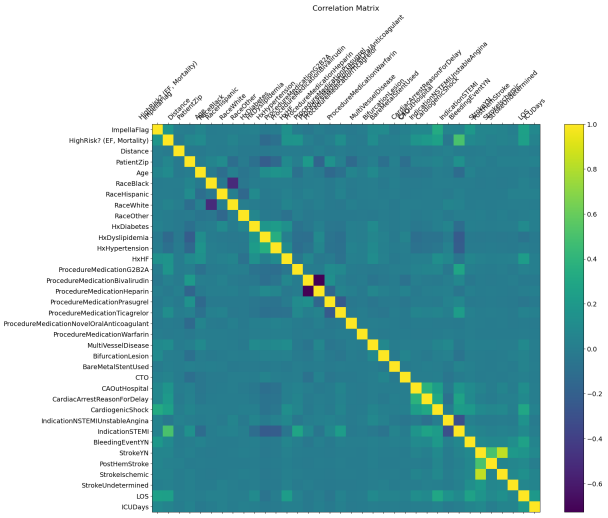

Figure 1: Correlation Matrix for Numerical Independent Variables

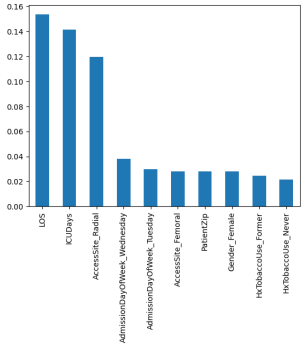

Figure 2: Feature Importances from the Random Forest

7.1 Figure 3: Variable Definitions

Note that if the definition of the variable is apparent, the description is left blank.
(C) - Categorical, (CB) - Categorical and binary, (Cont) - Continuous

Procedure (C) Procedure that patient underwent

HighRisk? (CB) A flag indicating whterh or not eh patient is deemed high risk

Distance (Cont) Distance from patient's home to the hospital

PatientZip (C) Zip code of patient's home

Hospital (C)

AdmissionDayOfWeek (C)

Age (Cont)

Gender (CB)

RaceBlack (CB)

RaceHispanic (CB)

RaceWhite (CB)

RaceOther (CB)

SummaryPayorClass (C) Description of the type of health insurance subscribed to by the patient

HxDiabetes (CB) Boolean variable indicating whether or not the patinet has diabetes

HxDyslipidemia (CB)

HxHypertension (CB)

HxTobaccoUse (C) Categorical variable describting frequency of tobacco use

HxHF (CB)

ProcedureMedicationG2B2A (CB) Boolean variable indicating use of medication during procedure

ProcedureMedicationBivalirudin (CB)

ProcedureMedicationHeparin (CB)

ProcedureMedicationPrasugrel (CB)

ProcedureMedicationTicagrelor (CB)

ProcedureMedicationNovelOralAnticoagulant (CB)

ProcedureMedicationWarfarin (CB)

MultiVesselDisease (CB) Boolean variable indicating whether or not the patient has multiple diseased organs

AccessSite (C) Access site for surgery

BifurcationLesion (CB) Boolean variable for a lesion

BareMetalStentUsed (CB)

CTO (CB) Chronic total occlusions

CAOutHospital (CB)

CardiacArrestReasonForDelay (CB)

CardiogenicShock (CB)

IndicationNSTEMIUnstableAngina (CB)

IndicationSTEMI (CB)

BleedingEventYN (CB)

StrokeYN (CB)

PostHemStroke (CB)

StrokeIschemic (CB)

StrokeUndetermined (CB)

LOS (Cont) Length of stay in the hospital

ICUDays (Cont) Number of days spent in the intensive care unit of the hospital

Below is the code used in the analysis described above.

```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# Modelling
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,
    precision_score, recall_score, ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV,
    train_test_split, cross_validate, RepeatedStratifiedKFold,
    cross_val_score
from scipy.stats import randint
from statistics import mean
from imblearn.over_sampling import BorderlineSMOTE, SMOTE

# Tree Visualisation
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz


# Read in data
df = pd.read_excel('Data.xlsx', [1]).get(1)

# Drop redundant columns and those with high NaN counts
df2 = df.drop(columns=[
   'BiomeEncounterID', 'VisitId', 'TotalCost', 'DirectCost',
   'ProcedureRelatedDirectCost', 'SyntaxScore', 'Mortality',
   'InsurancePayerManagedCare',
   'InsurancePayerMedicareTraditional',
       'InsurancePayerMedicareRisk',
   'InsurancePayerSelfPayUnmapped',
       'InsurancePayerMedicaidTraditional',
   'InsurancePayerOther', 'InsurancePayerMedicaidRisk',
   'InsurancePayerManagedMedicare', 'InsurancePayerMedicare',
   'InsurancePayerCommercial', 'InsurancePayerMedicaid',
   'InsurancePayerManagedMedicaid', 'InsurancePayerSelfPay',
   'InsurancePayerWorkersCompensation',
       'InsurancePayerPayorNotProvided',
   'FullMortalityRisk', 'HospitalSystem', 'PrePCILVEF',
       'MVSupport', 'Quarter',
   'ProceduralistNPI', 'RadialAccess', 'RaceUnknown',
       'RaceUnknown', 'IndicationOther'
])
```

```
df2 = df2.fillna({ # Fill rest of NaNs with either specific
    values or mode of column
   'StrokeUndetermined': 'Yes',
   'Distance': df2['Distance'].mean()
}).fillna(df2.mode().to_dict('records')[0]).replace(['Yes',
    'No'], [1,0]) # Replace Yes/No with binary value

# Replace PatientZip with mode of zip grouped by Hospital, fix
    leading 0 issue with integer cast
dfz = df2[df2['PatientZip'] !=
    0].groupby('Hospital')['PatientZip'].agg(pd.Series.mode)
dfz['UCSF'] = dfz['UCSF'][0]
dfz = dfz.reset_index().rename(columns={"PatientZip":
    "zipMode"})
df2 = df2.merge(dfz, on='Hospital')
df2['PatientZip'] = np.where(df2['PatientZip'] == 0,
    df2['zipMode'], df2['PatientZip']).astype(int)
df2 = df2.drop(columns='zipMode')

# Plot correlation matrix for numerical columns
dfnum = df2.select_dtypes(include=np.number)
f = plt.figure(figsize=(19, 15))
plt.matshow(dfnum.corr(), fignum=f.number)
plt.xticks(range(dfnum.shape[1]), dfnum.columns, fontsize=14,
    rotation=45)
plt.yticks(range(dfnum.shape[1]), dfnum.columns, fontsize=14)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title('Correlation Matrix', fontsize=16);

# One-hot keying categorical variables
dfstr = df2.select_dtypes(exclude=['int64', 'float64'])
df2 = pd.concat([df2, pd.get_dummies(dfstr).astype(int)],
    axis=1)
df2 = df2.drop(columns = dfstr.columns)

# Split the data into features (X) and target (y)
X = df2.drop('ImpellaFlag', axis=1)
y = df2['ImpellaFlag']

# Random Forest
rf = RandomForestClassifier()
rand_search = RandomizedSearchCV(
   rf,
   param_distributions = {'n_estimators': randint(50,500),
       'max_depth': randint(1,20)},
   n_iter=5,
   cv=5
)
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, stratify=y)

rand_search.fit(X_train, y_train)

# Print the best hyperparameters
print('Best hyperparameters:', rand_search.best_params_)

# Generate predictions with the best model
y_pred = rand_search.best_estimator_.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm).plot();

#Use SMOTE to oversample the minority class
oversample = SMOTE()
over_X, over_y = oversample.fit_resample(X, y)
over_X_train, over_X_test, over_y_train, over_y_test =
    train_test_split(over_X, over_y, test_size=0.2,
    stratify=over_y)

# Retrain using SMOTE
rf = RandomForestClassifier()
rand_search = RandomizedSearchCV(
   rf,
   param_distributions = {'n_estimators': randint(50,500),
       'max_depth': randint(1,20)},
   n_iter=5,
   cv=5
)

rand_search.fit(over_X_train, over_y_train)

# Print the best hyperparameters
print('Best hyperparameters:', rand_search.best_params_)

SMOTE_SRF = RandomForestClassifier(
   n_estimators=rand_search.best_params_['n_estimators'],
   max_depth = rand_search.best_params_['max_depth'],
   random_state=0
)
#Randomly spilt dataset to test and train set
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
    test_size=0.2, stratify=y)
#Train SMOTE SRF
SMOTE_SRF.fit(over_X_train, over_y_train)
#SMOTE SRF prediction result
y_pred = SMOTE_SRF.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm).plot();

# Create a series containing feature importances from the
    model and feature names from the training data
feature_importances =
    pd.Series(SMOTE_SRF.feature_importances_,
    index=X_train.columns).sort_values(ascending=False).head(10)

# Plot a simple bar chart
feature_importances.plot.bar();

# Train KNN Model
k_values = [i for i in range (1,31)]
scores = []

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

for k in k_values:
   knn = KNeighborsClassifier(n_neighbors=k)
   score = cross_val_score(knn, X_scaled, y, cv=5)
   scores.append(np.mean(score))

best_index = np.argmax(scores)
best_k = k_values[best_index]

X_train, X_test, y_train, y_test = train_test_split(X_scaled,
    y, test_size=0.2, stratify=y)

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(over_X_train, over_y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

```
# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm).plot();

# Train NB model
nb = GaussianNB()
nb.fit(over_X_train, over_y_train)

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, stratify=y)
y_pred = nb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

ConfusionMatrixDisplay(confusion_matrix=cm).plot();
```