PREDICTION OF HOSPITAL READMISSION FOR PATIENTS WITH DIABETES

GROUP 2

Contents

Abstract 2
Introduction 2
Data Preparation 4
Exploratory Data Analysis 7
Model Preparation 13
Training/Validation/Test Samples 13
Supervised Learning Models 13
Performance Metrics 13
Stochastic Gradient Descent 14
Random Forest 15
Gradient Boosting Classifier 16
Comparison of Models 17
Fine-tuning Hyperparameters 19
Use the Best Classifier 21
Unsupervised Learning 22
Conclusion 27
References 28

Abstract

According to the Centers of Disease Control and Prevention, it defines diabetes as “the condition in which the body does not properly process food for use as energy. Most of the food we eat is turned into glucose, or sugar, for our bodies to use for energy. The pancreas, an organ that lies near the stomach, makes a hormone called insulin to help glucose get into the cells of the bodies. When you have diabetes, your body either doesn’t make enough insulin or can’t use its own insulin as well as it should. This causes sugars to build up in your blood.” (Centers For Disease Control and Prevention, 2019).

Since diabetes is not limited to one type of age group, nationality or skin color, considering how hospitals manage diabetes and the ratio of return visit along with the timeline will help hospitals streamline the managements of diabetes. This project will focus on the data for different diabetic patients, hospital visit(s), readmission and in conclusion will look at various predictive models to determine a pattern.

Introduction

The dataset considered for this research is from 130 hospitals in the United States of America comprising of different locations: Midwest, Northeast, South and West. It is from the year 1999 to 2008 (10 years) and considers a total of 78 hospitals in total. The information in the data varies from, gender, age, weight, admission, discharge, time in hospital and other information. However, for this research, the focus will be on readmission after initial visit to the hospital. What does the entire data from the dataset looks like? The information below in note 1.1 will provide a snapshot of the size and information of the data.

From note 1.1, the dataset had a total of 101,766 entries and 49 columns. The Dtype for the columns are mixed between objects (37) and int64 (12). The information in the columns is well detailed with the information required to make an informed decision about the management of diabetes. However, successful management of diabetes also means reducing the numbers of people that will be likely readmitted either due to worsening condition or just from an omission by the medical staffs at the hospital. By focusing on the readmitted patients, it will improve productivity in the hospital and also, the possible of saving lives.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 101766 entries, 2278392 to 443867222
Data columns (total 49 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   patient_nbr               101766 non-null  int64
 1   race                      99493 non-null   object
 2   gender                    101766 non-null  object
 3   age                       101766 non-null  object
 4   weight                    3197 non-null    object
 5   admission_type_id         101766 non-null  int64
 6   discharge_disposition_id  101766 non-null  int64
 7   admission_source_id       101766 non-null  int64
 8   time_in_hospital          101766 non-null  int64
 9   payer_code                61510 non-null   object
 10  medical_specialty         51817 non-null   object
 11  num_lab_procedures        101766 non-null  int64
 12  num_procedures            101766 non-null  int64
 13  num_medications           101766 non-null  int64
 14  number_outpatient         101766 non-null  int64
 15  number_emergency          101766 non-null  int64
 16  number_inpatient          101766 non-null  int64
 17  diag_1                    101745 non-null  object
 18  diag_2                    101408 non-null  object
 19  diag_3                    100343 non-null  object
 20  number_diagnoses          101766 non-null  int64
 21  max_glu_serum             101766 non-null  object
 22  A1Cresult                 101766 non-null  object
 23  metformin                 101766 non-null  object
 24  repaglinide               101766 non-null  object
 25  nateglinide               101766 non-null  object
 26  chlorpropamide            101766 non-null  object
 27  glimepiride               101766 non-null  object
 28  acetohexamide             101766 non-null  object
 29  glipizide                 101766 non-null  object
 30  glyburide                 101766 non-null  object
 31  tolbutamide               101766 non-null  object
 32  pioglitazone              101766 non-null  object
 33  rosiglitazone             101766 non-null  object
 34  acarbose                  101766 non-null  object
 35  miglitol                  101766 non-null  object
 36  troglitazone              101766 non-null  object
 37  tolazamide                101766 non-null  object
 38  examide                   101766 non-null  object
 39  citoglipton               101766 non-null  object
 40  insulin                   101766 non-null  object
 41  glyburide-metformin       101766 non-null  object
 42  glipizide-metformin       101766 non-null  object
 43  glimepiride-pioglitazone  101766 non-null  object
 44  metformin-rosiglitazone   101766 non-null  object
 45  metformin-pioglitazone    101766 non-null  object
 46  change                    101766 non-null  object
 47  diabetesMed               101766 non-null  object
 48  readmitted                101766 non-null  object
dtypes: int64(12), object(37)
memory usage: 38.8+ MB
```

Note 1.1

Data Preparation

It was mentioned in the introduction that emphasis will be placed on the most important variable which is 'readmitted'. The readmitted variable will tell us if a patient was hospitalized within 30 days, more than 30 days or not readmitted after the initial visit to the hospital.

We define the target variable for a binary classification: 0 = "Not readmitted/Readmitted more than 30 days"; 1 = "Readmitted within 30 days". (Please see note 1.2)

```
# Redefine the target variable
df['readmitted'] = df['readmitted'].replace('>30', 0)
df['readmitted'] = df['readmitted'].replace('<30', 1)
df['readmitted'] = df['readmitted'].replace('NO', 0)
```

Note 1.2

The next set in the data preparation was to drop duplicate records and only keep the first visit for each patient. By taking this step, an independent variable can be generated. (note 1.3)

```
# Only keep the first visit per patient
df.drop_duplicates(['patient_nbr'], keep = 'first', inplace = True)
```

Note 1.3

We find that the weight feature has lots of missing values (over 96%) and needs to be removed. We also remove the payer_code feature that is not useful for our analysis. (Note 1.4)

```
race                 2.72
weight              96.01
payer_code          43.41
medical_specialty   48.21
diag_1               0.02
diag_2               0.41
diag_3               1.71
dtype: float64
```

Note 1.4

We fill the missing values of race and three diagnoses with the most frequent values or mode[1]. For medical_specialty, we first sort the dataset to cluster similar patients and then back-fill null values. (note 1.5)

[1] It is imperative to mention that this does not take into consideration correlation and introduce data bias.

```
# Fill the null values with the most frequent values
df['race'] = df['race'].fillna(value=df['race'].describe().top)
```

```
df['diag_1'] = df['diag_1'].fillna(value=df['diag_1'].describe().top)
df['diag_2'] = df['diag_2'].fillna(value=df['diag_2'].describe().top)
df['diag_3'] = df['diag_3'].fillna(value=df['diag_3'].describe().top)
```

```
# Sort the dataset to cluster similar patients and back-fill null values
df = df.sort_values(['diag_1', 'age', 'admission_source_id'])
df['medical_specialty'] = df['medical_specialty'].fillna(method='bfill')
```

Note 1.5

The values of age are listed in the original dataset as; [0–10), [10–20), [20–30) and so on. In order to change it into a numerical feature and to make the processing and prediction analysis to function, we choose the midpoint value of each value of age, as seen in note 1.6 below.

```
# Label-encode age features (i.e. age 0-10 is converted to age 5)
df.age = (labelencoder.fit_transform(df.age)*10) + 5
```

Note 1.6

The result of the change in data value of age will change the information to, (0-10) = 5, (10-20) = 15, (20-30) = 25 and so on. The bins have been converted into definite numbers.

Another important column is discharge_disposition_id. If we look at the IDs_mapping document provided by UCI, we can see that 11,13,14,19,20,21 is related to hospice or death. Thus, we remove these samples since they cannot be readmitted, as demonstrated in note 1.7 below.

```
# Remove patients with discharge disposition codes related to hospice or death
df = df[~df.discharge_disposition_id.isin([11,13,14,19,20,21])]
```

Note 1.7

We also group patients based on similar categories of discharge codes, such as if they are discharged to home or to other hospitals. Similarly, we group patients based on categories of admission types and admission sources according to the IDs_mapping document. The code below was used to perform this task. (note 1.8)

```
# Group patients based on similar categories of discharge codes
df['discharge_disposition_id'] = df['discharge_disposition_id'].apply(lambda x : 1 if int(x) in [6, 8]
                                                                      else ( 2 if int(x) in [3, 4, 5, 16, 22, 23, 24]
                                                                      else ( 9 if int(x) in [12, 15, 17]
                                                                      else ( 27 if int(x) in [28, 29, 30]
                                                                      else ( 18 if int(x) in [25, 26]
                                                                      else int(x) )))))
```

```
# Group patients based on similar categories of admission types
df['admission_type_id'] = df['admission_type_id'].apply(lambda x : 1 if int(x) in [2, 7]
                                                        else ( 5 if int(x) in [6, 8]
                                                        else int(x) ))
```

```
# Group patients based on similar categories of admission sources
df['admission_source_id'] = df['admission_source_id'].apply(lambda x : 1 if int(x) in [2, 3]
                                                            else ( 4 if int(x) in [5, 6, 10, 18, 22, 25, 26]
                                                            else ( 9 if int(x) in [15, 17, 20, 21]
                                                            else ( 11 if int(x) in [12, 13, 14, 23, 24]
                                                            else int(x) ))))
```

Note 1.8

We drop two medications which are not prescribed at all. They are constant variables and will not be useful for our predictive model. The two medication dropped are ‘examide’ and citoglipton. The code in note 1.9 demonstrates how this function was performed.

```
# Drop two medications which were not prescribed at all (pertaining 'No' for all instances)
df.drop(['examide', 'citoglipton'], axis=1, inplace=True)
```

Note 1.9

We convert the following nominal features into numeric features: change, diabeteMed, max_glu_serum and A1Cresult. We assign numbers to indicate the change of medication and the different test results. For instance, we use the number ‘7’, if the A1Cresult is greater than 7%. We use the number ‘8’, if the A1Cresult is greater than 8%. If the test result is normal, we use the number ‘5’. And if there is no test done (‘none’), we use the number ‘0’.

```
# Convert the change feature
df['change'] = df['change'].apply(lambda x : 1 if x == 'Ch' else -1)
```

```
# Convert the diabeteMed feature
df['diabetesMed'] = df['diabetesMed'].apply(lambda x : -1 if x == 'No' else 1)
```

```
# Convert the max_glu_serum feature
df['max_glu_serum'] = df['max_glu_serum'].apply(lambda x : 200 if x == '>200'
                                                else ( 300 if x == '>300'
                                                else ( 100 if x == 'Norm'
                                                else  0)))
```

```
# Convert the A1Cresult feature
df['A1Cresult'] = df['A1Cresult'].apply(lambda x : 7 if x == '>7'
                                        else (8 if  x == '>8'
                                        else ( 5 if x == 'Norm'
                                        else  0)))
```

Note 2.1

As for diagnoses, we want to focus on the primary diagnosis (diag_1). We use the ICD9 diagnosis codes to group the values of the primary diagnosis. We also group 70 types of medical specialties into fewer categories based on domain knowledge.

```
# Use the ICD9 diagnosis codes to group primary diagnosis
df['diag_1'] = df['diag_1'].apply(lambda x : 'other' if (str(x).find('V') != -1 or str(x).find('E') != -1)
                                  else ('circulatory' if int(float(x)) in range(390, 460) or int(float(x)) == 785
                                  else ('respiratory' if int(float(x)) in range(460, 520) or int(float(x)) == 786
                                  else ('digestive'   if int(float(x)) in range(520, 580) or int(float(x)) == 787
                                  else ('diabetes'    if int(float(x)) == 250
                                  else ('injury'      if int(float(x)) in range(800, 1000)
                                  else ('musculoskeletal' if int(float(x)) in range(710, 740)
                                  else ('genitourinary'   if int(float(x)) in range(580, 630) or int(float(x)) ==
                                  else ('neoplasms'       if int(float(x)) in range(140, 240)
                                  else ('pregnecy'        if int(float(x)) in range(630, 680)
                                  else 'other')))))))))))
```

```
df['medical_specialty'] = df['medical_specialty'].apply(lambda x : 'surgery' if str(x) in surgery
                                                         else ( 'pediatrics' if str(x) in pediatrics
                                                         else ( 'psychic' if str(x) in psychic
                                                         else ( 'neurology' if str(x) in neurology
                                                         else ( 'obstetrics' if str(x) in obstetrics
                                                         else ( 'endocrinology' if str(x) in endocrinology
                                                         else ( 'other' if str(x) in other
                                                         else str(x) )))))))
```

Note 2.2

Most of the algorithms (or ML libraries) in Python produce better results with numerical variables. We group all numerical features, including converted nominal features, into a list. Then we find significant numerical features using Spearman Correlation Coefficient[2]. Basically, we use Spearman's correlation coefficient to check whether numeric features and the target variable are dependent or independent. If they are independent (p-value >0.4), these features will be considered as insignificant and will get removed.

To convert categorical features to numbers, we will use a technique called one-hot encoding[3]. If we create a column for each unique value, we will get correlated columns. Therefore, we use the drop_first option to drop the first categorical value for each column. As for ID types of categorical features, we need to convert them into strings so that the get_dummies function can work properly.

Exploratory Data Analysis

We want to explore the correlations of our feature variables with readmission status. We use the absolute value of the correlation coefficients, which give us the relationship strength. Basically, the larger the number, the stronger the relationship.

[2] Spearman's correlation coefficient, (ρ, also signified by r_s) measures the strength and direction of association between two ranked variables. (Laerd Statistics, 2018)
[3] A one hot encoding is a representation of categorical variables as binary vectors. (Brownlee, 2017)

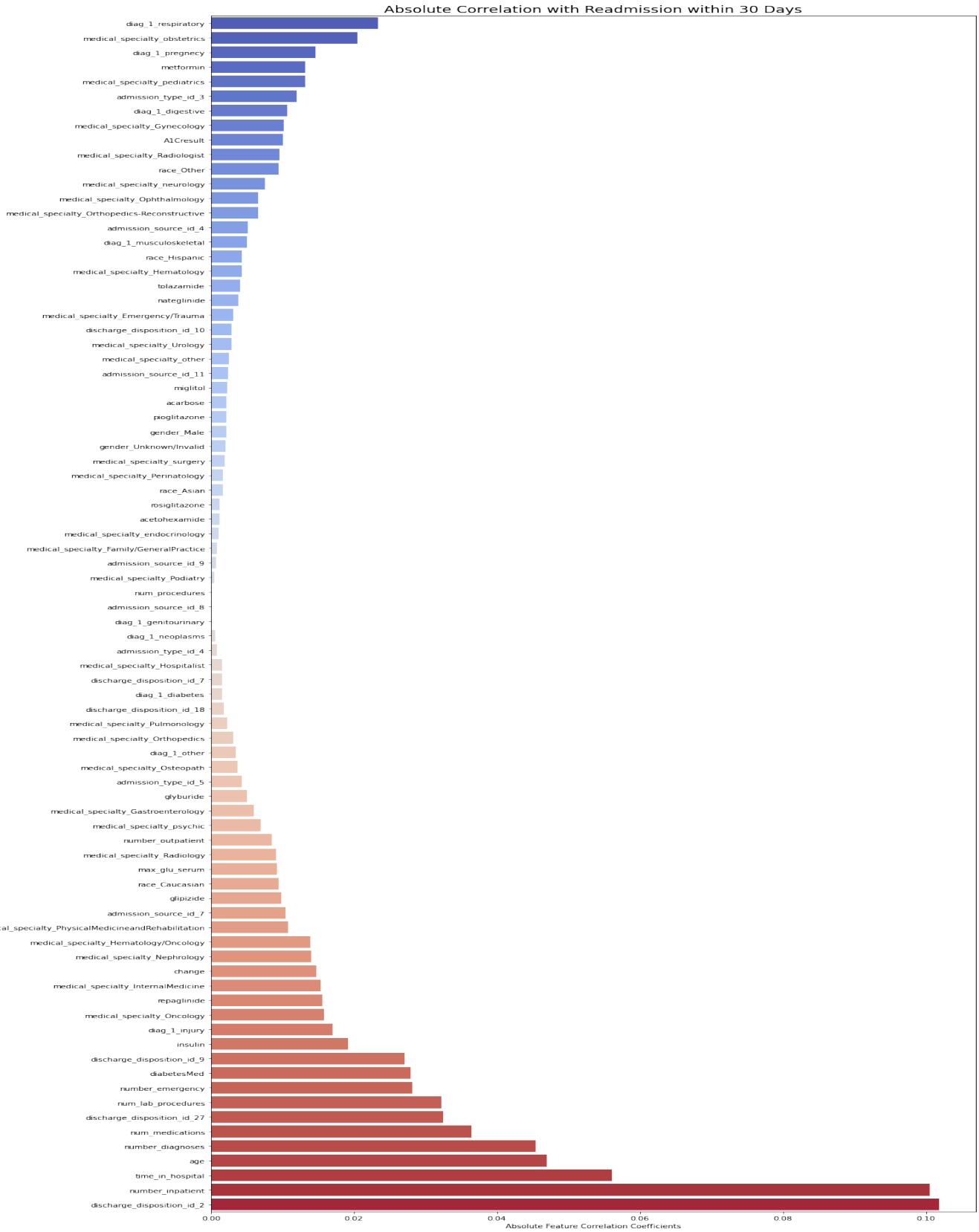

Figure 1-1 Absolute Correlation with Readmission within 30 days

According to the Figure 1-1, we can find that the following features are strongly correlated with readmission status: discharge_disposition_id_2, number of inpatient visits, time in hospital, age, number of diagnoses, number of medications, discharge_disposition_id_27, number of lab procedures, number of emergency visits, diabetes medications, and discharge_disposition_id_9. According to the IDs_mapping document, discharge_dispoistion_id_2 means that patients were discharged/transferred to another short-term hospital. In addition, we group 2, 3, 4, 5, 10, 16, 22, 23, 24 together, since they are all related to being discharged to other hospitals or healthcare facilities. According to the IDs_mapping document, discharge_disposition_id_27 means that patients were discharged/transferred to a federal health care facility. In addition, we group 27, 28, 29, 30 together, since they are all related to being discharged to other types of healthcare institutions not defined elsewhere. Discharge_disposition_id_9 means that a patient was admitted as an inpatient to this hospital. Similarly, we group 9, 12, 15, 17 together, since they are all related to being readmitted to the same hospital.

We also explore the correlation between numerical features and from that exploration, we discovered that time_in_hospital is highly related to num_lab_procedures, num_procedures and num_medications. This conclusion is logical since those patients who stay longer in hospital are more likely to undertake higher numbers of procedures such as, lab procedures, additional treatments and get more medications.

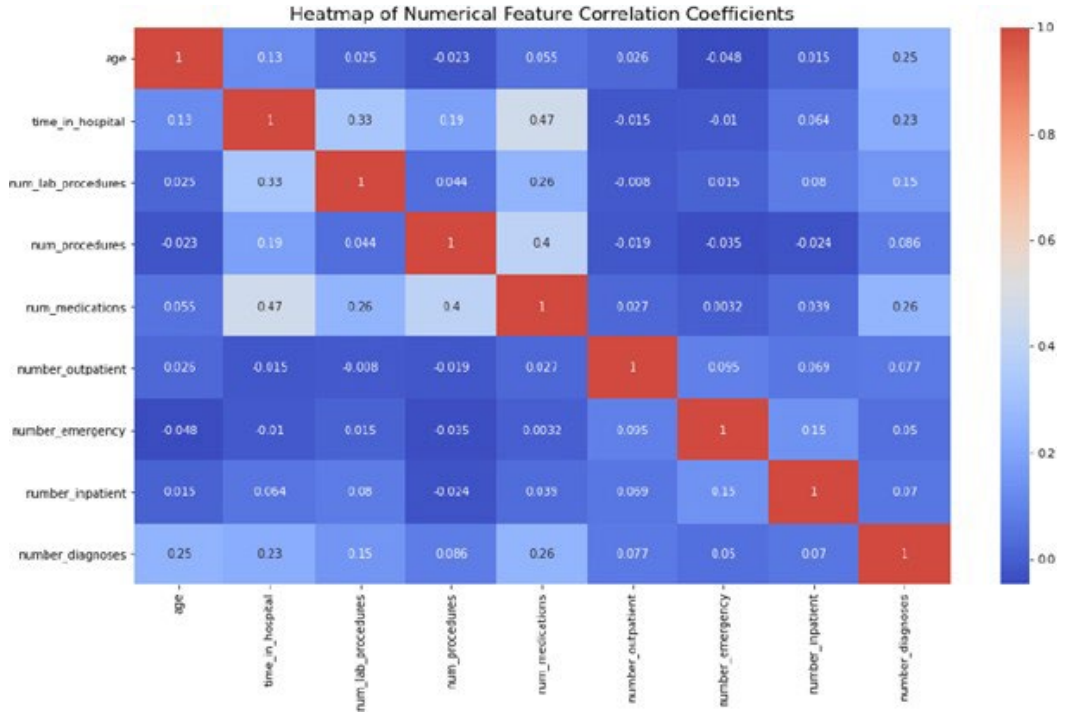

Figure 1-2 Heatmap of Numerical Feature Correlation Coefficients

Then we focus on the exploration of highly correlated features with readmission status. The first one is the frequency distribution of previous inpatient visits. According to the following chart, we

can tell that for patients who were readmitted within 30 days, a greater proportion of them had some history of inpatient visits. This conclusion is expected due to the fact that those patients have a pattern of returning to the hospital due to the seriousness of their condition.

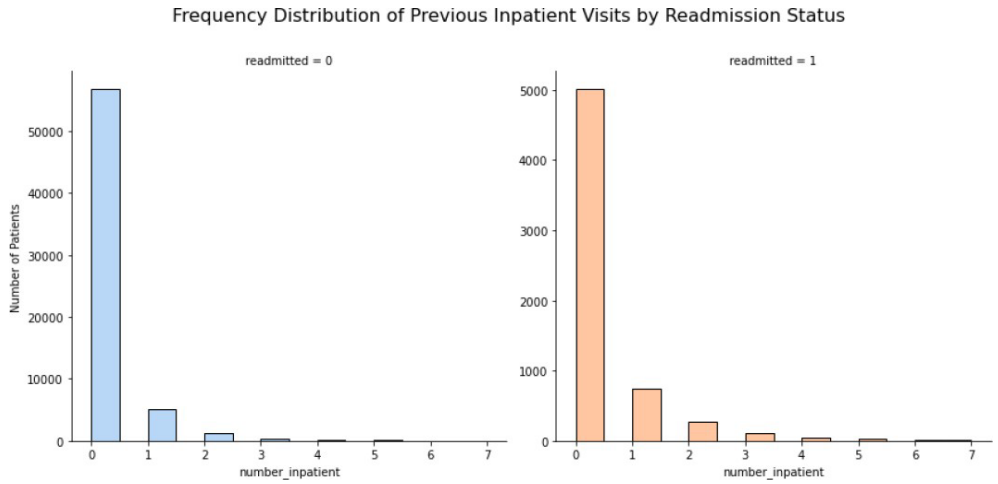

Figure 1-3 Frequency Distribution of Previous Inpatient Visits by Readmission Status

We also want to see the influence of time spent in hospital on readmission status. Based on the following charts, we can see that readmitted patients spent longer time in hospital.

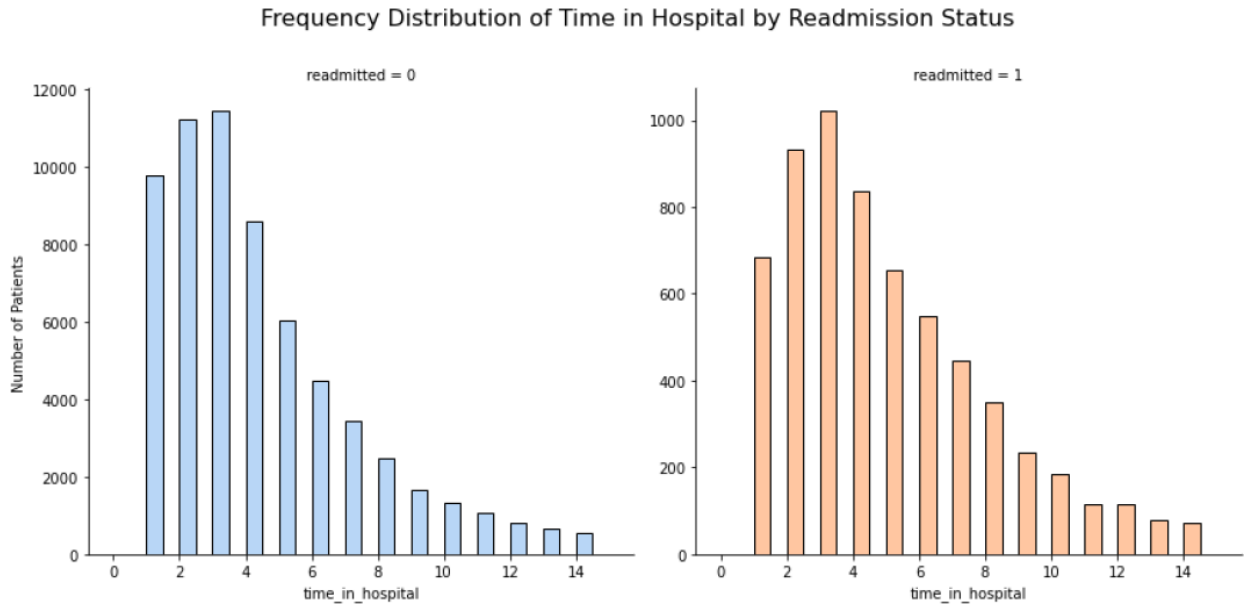

Figure 1-4 Frequency Distribution of Time in Hospital by Readmission Status

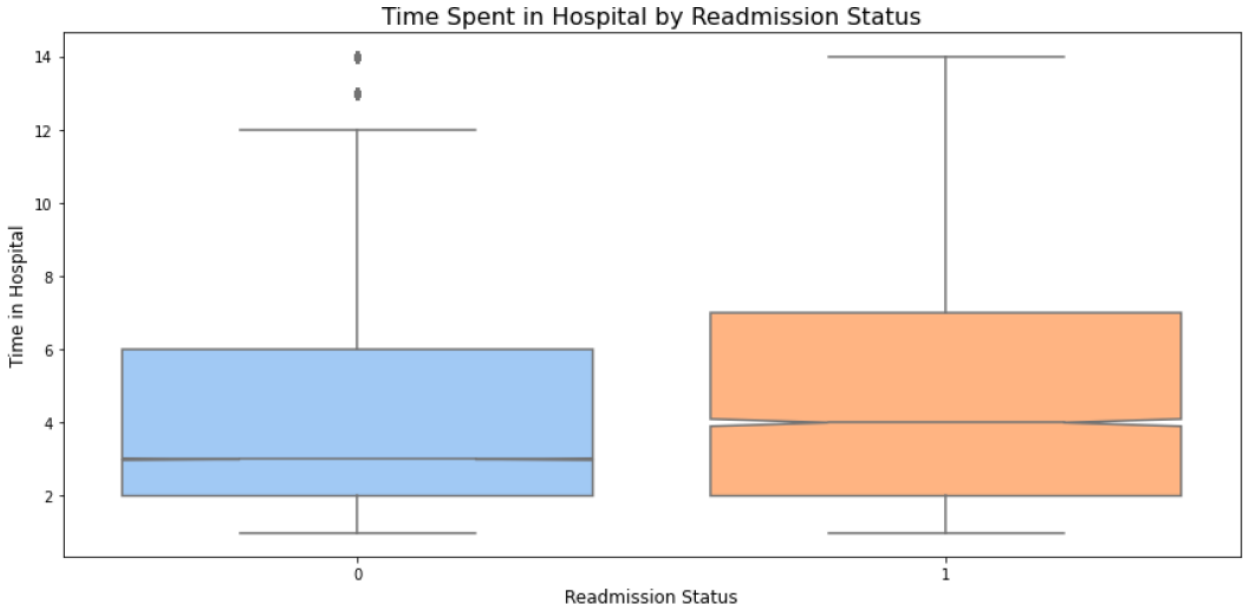

Figure 1-5 Time Spent in Hospital by Readmission Status

According to our correlation chart, age is another important feature. We want to visually explore the relationship between age and readmission status. And we find that readmitted patients are older than their counterparts.

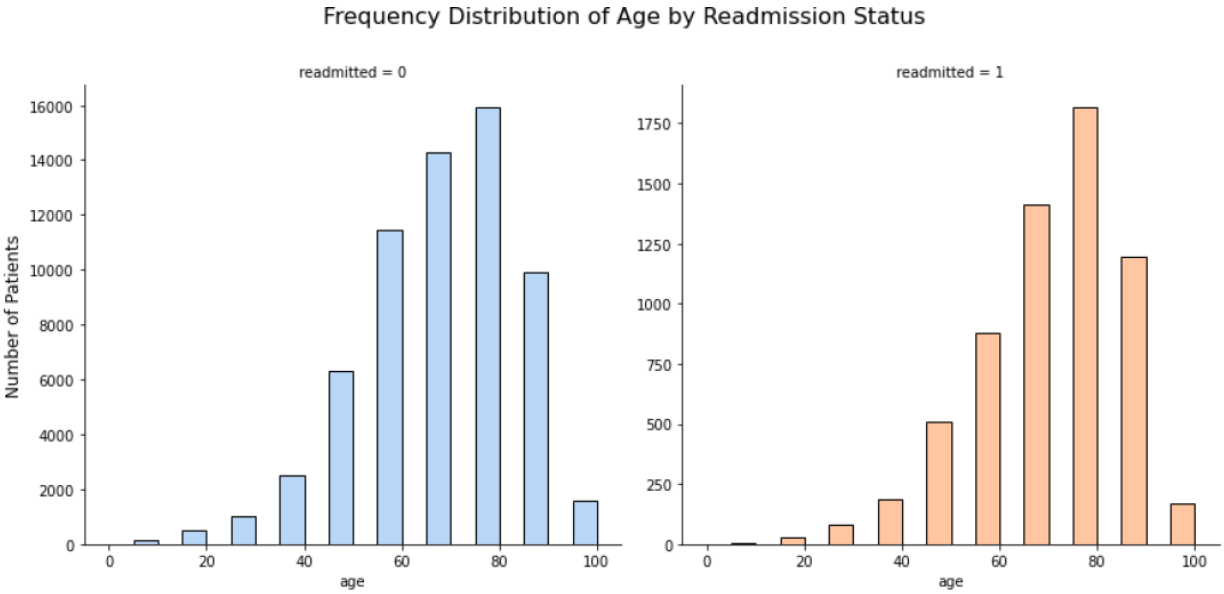

Figure 1-6 Frequency Distribution of Age by Readmission Status

From the above chart, the older the patient, the more frequently they are readmitted to the hospital compared to patients from the ages of 0 to 40 years old.

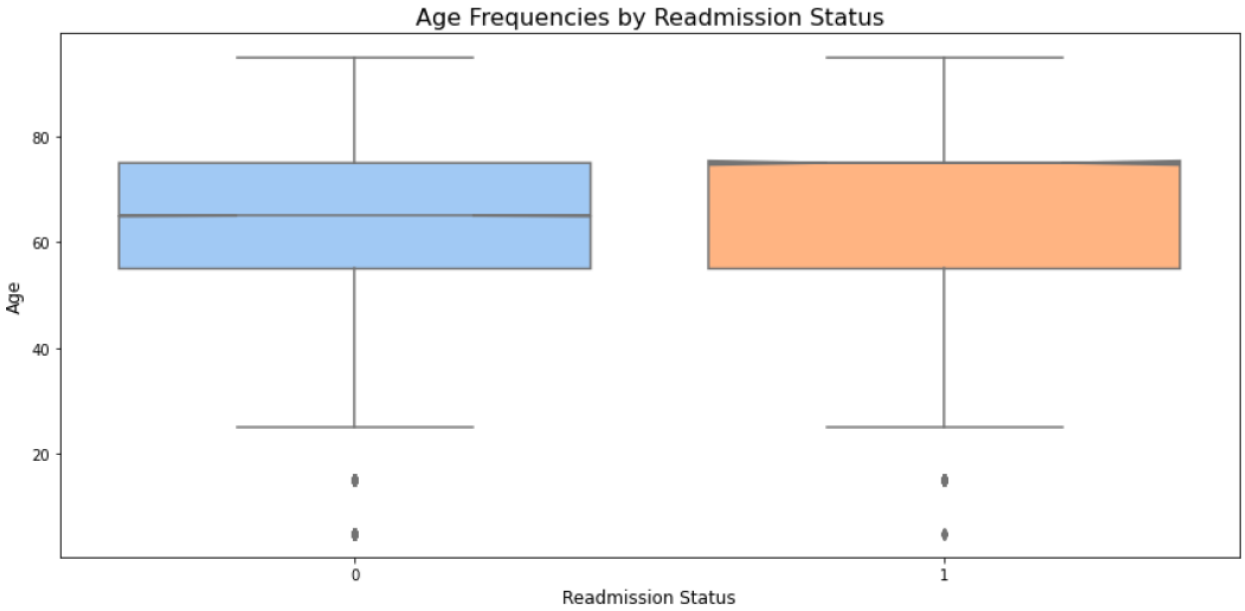

Figure 1-7 Age Frequencies by Readmission Status

Finally, we use sns.countplot to explore the distribution of readmission status. It is obvious that our data is imbalanced. We will address this issue in the following section. Kindly refer to figure 1-8 below for the imbalance in the data.

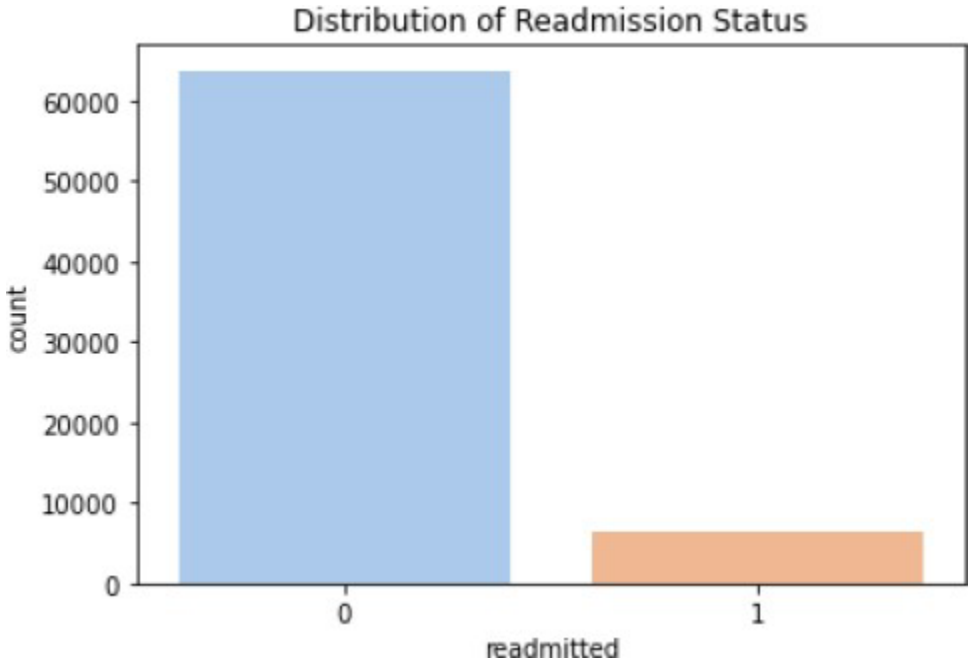

Figure 1-8 Distribution of Readmission Status

The next section in this research will address the imbalance in the data and it will consider model preparation.

Model Preparation

Training/Validation/Test Samples

From the Figure 1-8 above, we have an imbalanced dataset. “An imbalance occurs when one or more classes have very low proportions in the training data as compared to the other classes.” (Kuhn, 2013) According to our dataset, around 11% of the patients were readmitted within 30 days. This means that the majority of patients were either not readmitted or were readmitted after 30 days. In this case, the model might be overfit to the majority class and become oblivious to the minority class. Resampling is a widely adopted technique for dealing with highly unbalanced datasets. Usually, we need to split the data into training, test and validation sets before the resampling. And resampling is only carried out on the training set otherwise the performance measures could be skewed. In our project, we split the dataset into 70% training, 15% validation and 15% test.

There are three types of resampling: over-sampling, under-sampling and synthetic sampling. And we choose to use under-sampling, which involves removing samples from the majority class (readmitted=0). Because the sample size of our minority class (readmitted=1) is large enough (over 11,000). We first extract 30% of the data to be used as the validation/test data. Then we split the validation/test data in half. We use the remaining data as the training data. In order to get the balanced training data, we split the training data into positive rows (readmitted=1) and negative rows (readmitted=0). Then we merge the rows together, but this time we use the length of positive rows for our negative samples. In this way, we have a balanced training data with 50% positive and 50% negative.

Supervised Learning Models

Performance Metrics

Before selecting models, we set up a report function to return performance metrics. We use the following performance metrics: precision, recall, accuracy and AUC.

Precision Score: According to scikit-learn documentation, the precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1 and the worst value is 0.

Recall Score: According to scikit-learn documentation, the recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. Again, the best value is 1 and the worst value is 0.

Accuracy Score: Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted samples to the total samples. According to scikit-learn documentation, the accuracy_score function computes the accuracy, either the fraction (default) or the count (normalize=False) of correct predictions. Accuracy = (tp+tn)/(tp+fp+fn+tn)

AUC Score: AUC is used most of the time to mean AUROC. It computes Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores. Higher the AUC score, the better the model is at predicting 0s as 0s and 1s as 1s.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an optimization algorithm often used in machine learning to find the parameters corresponding to the best fit between the prediction and the actual data. Basically, gradient descent means starting with a random point and descending a slope to reach the lowest point. In addition, SGD randomly picks one data point from the whole data set at each iteration to reduce the computations of huge data enormously.

According to scikit-learn documentation, SGD is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. For instance, using SGDClassifier (loss='log') results in logistic regression, i.e., a model equivalent to LogisticRegression which is fitted via SGD instead of being fitted by one of the other solvers in LogisticRegression. The advantages of Stochastic Gradient Descent are efficiency and ease of implementation (lots of opportunities for code tuning).

We first initialize the model and then fit the training data using the following code:

```
sgdc=SGDClassifier(loss = 'log',alpha = 0.1,random_state = 40)
sgdc.fit(X_train_trans, y_train)
```

We make predictions based on training and validation datasets. And we get the following performance report.

```
Stochastic Gradient Descend
Training:
AUC:0.652
accuracy:0.608
recall:0.545
precision:0.624
specificity:0.671
prevalence:0.500

Validation:
AUC:0.635
accuracy:0.656
recall:0.523
precision:0.133
specificity:0.669
prevalence:0.088
```

Random Forest

Random Forest is a versatile machine learning method based on decision trees. It is an ensemble of decision tree algorithms. Random Forest can be used for both classification and regression problems. According to scikit-learn documentation, in random forests, each tree in the ensemble is built from a sample drawn with replacement from the training set. And only a random subset of features is taken into consideration by the algorithm when splitting a node. In addition, random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant, hence yielding an overall better model.

We first initialize the Random Forest classifier and then fit the training and validation data using the following code:

```
rf=RandomForestClassifier(max_depth = 6, random_state = 40)
rf.fit(X_train_trans, y_train)
```

We make predictions based on training and validation datasets. And we get the following performance report:

```
Random Forest
Training:
AUC:0.689
accuracy:0.629
recall:0.590
precision:0.641
specificity:0.669
prevalence:0.500

Validation:
AUC:0.633
accuracy:0.630
recall:0.550
precision:0.128
specificity:0.638
prevalence:0.088
```

Gradient Boosting Classifier

Gradient Boosting (GB) classifier takes a weak learning algorithm and makes a series of changes to it that improve the strength of the learner. The idea of gradient boosting is whether a weak learner can be gradually modified into a better one. GB consists of two parts: a weak learner and an additive component. According to scikit-learn documentation, GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage, n_classes_ regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. In addition, the number of weak learners is controlled by the parameter n_estimators; The size of each tree can be controlled either by setting the tree depth via max_depth or by setting the number of leaf nodes via max_leaf_nodes. The learning_rate is a hyper-parameter in the range (0.0, 1.0] that controls overfitting via shrinkage. GB[4] supports both binary and multi-class classification.

We first initialize the Gradient Boosting classifier and then fit the training and validation data using the following code:

```
gbc =GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=3, random_state=40)
gbc.fit(X_train_trans, y_train)
```

We make predictions based on training and validation datasets. And we get the following performance report:

[4] GB stands for Gradient Boosting.

```
Gradient Boosting Classifier
Training:
AUC:0.819
accuracy:0.734
recall:0.730
precision:0.736
specificity:0.738
prevalence:0.500

Validation:
AUC:0.591
accuracy:0.575
recall:0.563
precision:0.114
specificity:0.576
prevalence:0.088
```

Comparison of Models

We compare the results of the three models and plot the outcomes below. Here we use the AUC score to evaluate the best model, because the AUC score provides an aggregate measure of performance across all possible classification thresholds.

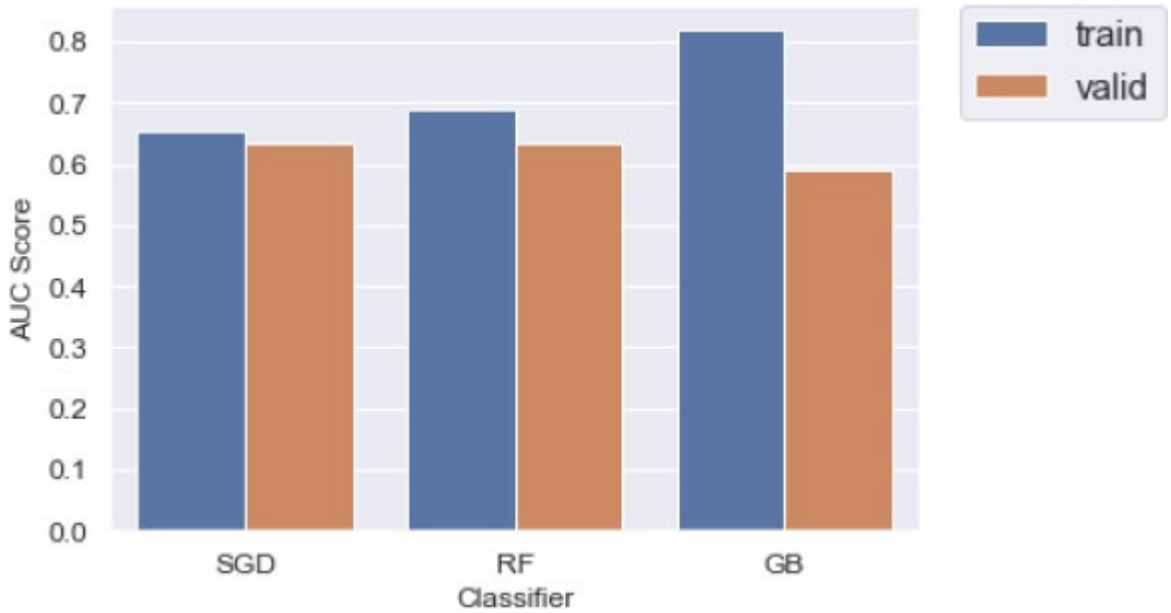

Figure 1-9: Comparison of Model Performances

According to the chart, we can see that Gradient Boosting classifier has the best performance, especially on the training dataset. In addition, Stochastic Gradient Descent and Random Forest have similar performances.

We can take a closer look at the most important features selected by better performance models (RF and GB) to get more insights into our datasets. Sklearn provides a useful tool named feature_importances for us to easily measure the relative importance of each feature based on the prediction. Here, the level of importance is measured based on the fact that the feature contributes enough to the prediction process.

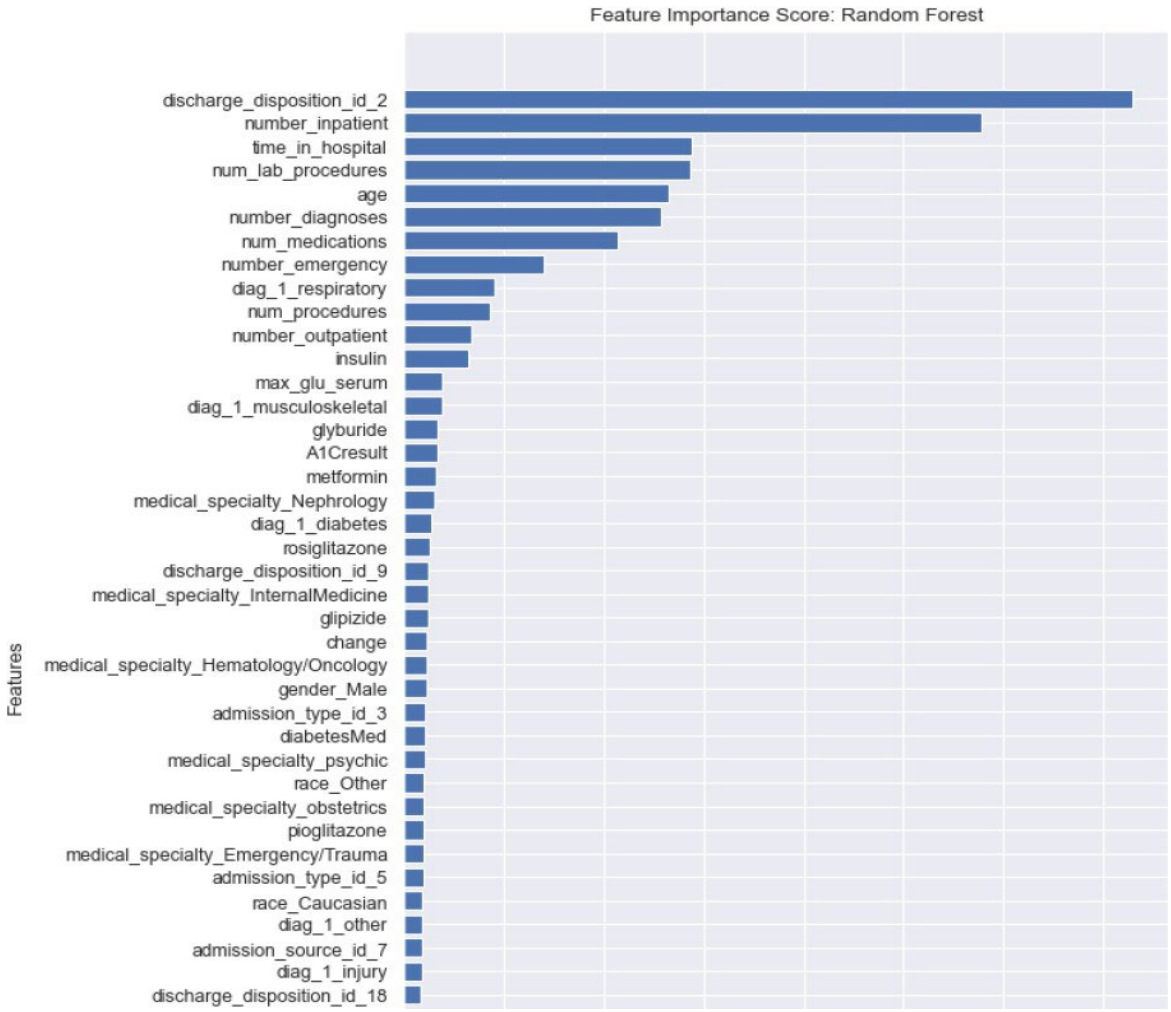

Figure 1-10: Feature Importance Score: Random Forest

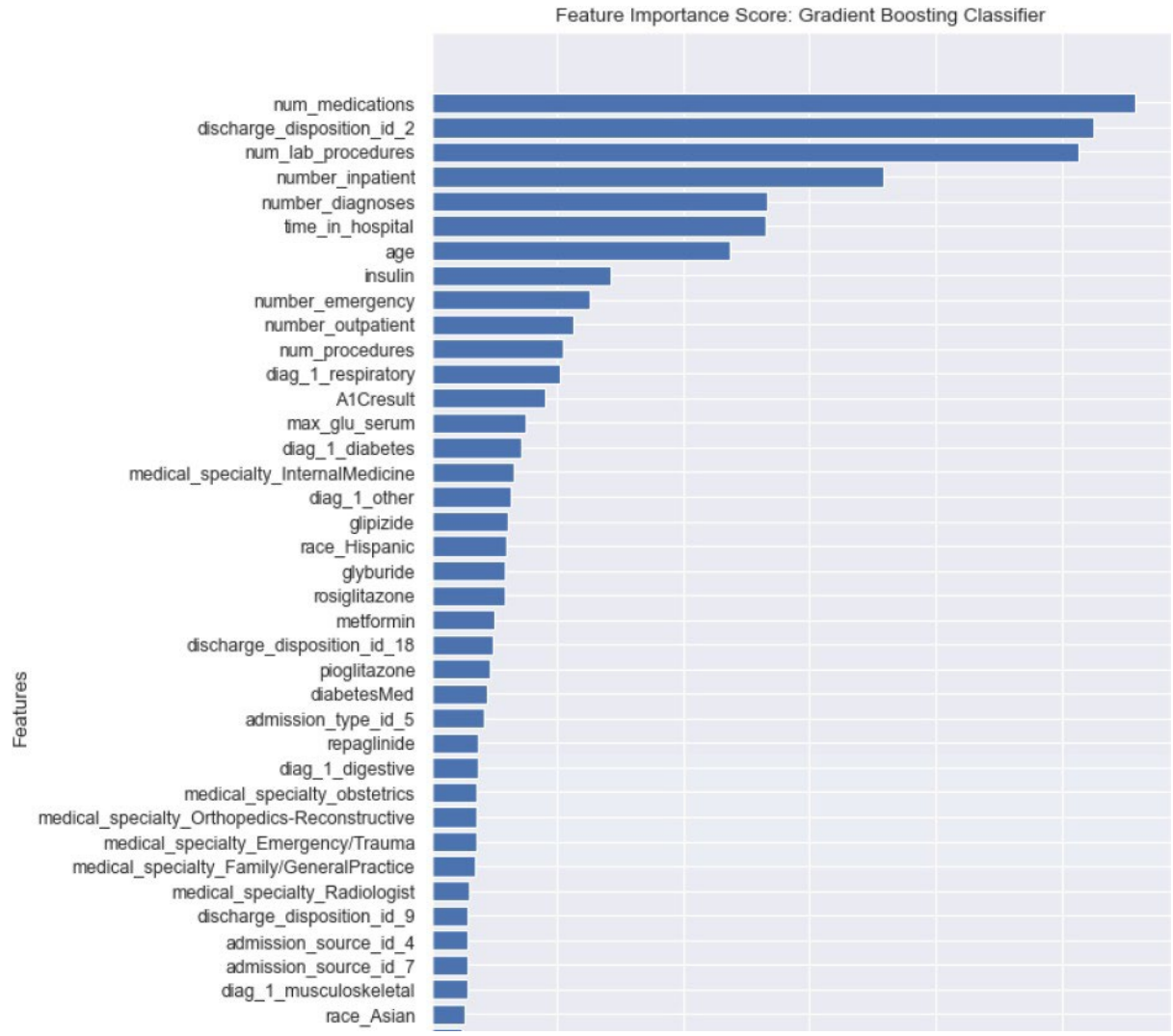

Figure 1-11: Feature Importance Score: Gradient Boosting Classifier

Based on the above charts, we can find the following important features: discharge_disposition_id_2, number of inpatient visits, time in hospital, number of lab procedures, number of diagnoses, number of medications and age. According to the IDs_mapping document, discharge_disposition_id_2 means that the patients are discharged or transferred to another hospital. The findings are highly consistent with the Figure 1-1, absolute Correlation with Readmission within 30 days in the EDA section.

Fine-tuning Hyperparameters

Hyperparameter tuning is the process of determining the right values of parameters for the optimization of the models. Hyperparameters determine the model architecture instead of the model itself. Grid search is the most basic hyperparameter tuning method, which is very

computationally intensive. The alternative method is random search, which is a randomized search on hyper parameters. We use the RandomizedSearchCV function for this project. According to scikit-learn documentation, in contrast to GridSearchCV, not all parameter values are tried out by this method, but rather a fixed number of parameter settings is sampled from the specified distributions. The number of parameter settings that are tried is given by n_iter.

We first set up the list of hyperparameters using the following code:

```
n_estimators = range(200,1000,200)
max_features = ['auto','sqrt']
max_depth = range(1,10,1)
min_samples_split = range(2,10,2)
criterion = ['gini','entropy']

random_grid = {'n_estimators':n_estimators,
               'max_features':max_features,
               'max_depth':max_depth,
               'min_samples_split':min_samples_split,
               'criterion':criterion}

print(random_grid)
```

Then we create the RandomizedSearch CV and fit the randomized search model:

```
# Create the RandomizedSearchCV
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,
                               n_iter = 20, cv = 2, scoring=auc_score,
                               verbose = 1, random_state = 40)
```

```
# Fit the randomized search model
rf_random.fit(X_train_trans, y_train)
```

We perform similar steps to optimize Stochastic Gradient Descent and Gradient Boosting classifiers. And we compare the performance of the optimized models to that of the unoptimized models based on the validation set. We can see from the following chart the Gradient Boosting classifier performs better after the optimization. As for Stochastic Gradient Descent and Random Forest, their performance hasn't improved very much.

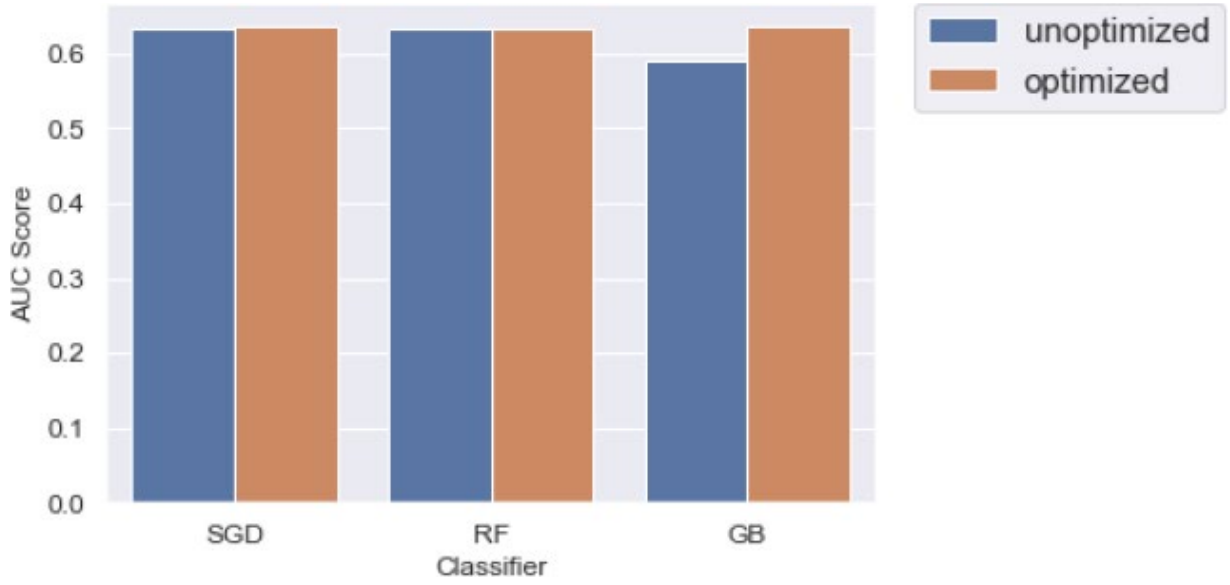

Figure 1-12 Comparison of Unoptimized and Optimized Models

Use the Best Classifier

Based on the above analysis, we decide to use the best performance model, which is the Gradient Boosting classifier. We first standardize our testing dataset and then use the best model to make predictions:

```
scaler = pickle.load(open('scaler.sav', 'rb'))
X_test_trans = scaler.transform(X_test)
y_train_pred = best_model.predict_proba(X_train_trans)[:,1]
y_valid_pred = best_model.predict_proba(X_valid_trans)[:,1]
y_test_pred = best_model.predict_proba(X_test_trans)[:,1]
```

After this, we print out the performance report as follows:

```
Training:
AUC:0.736
accuracy:0.669
recall:0.618
precision:0.688
specificity:0.719
prevalence:0.500

Validation:
AUC:0.635
accuracy:0.631
recall:0.547
precision:0.128
specificity:0.640
prevalence:0.088

Test:
AUC:0.637
accuracy:0.627
recall:0.561
precision:0.137
specificity:0.634
prevalence:0.094
```

We also create the ROC curves for the training, test and validation sets as follows:

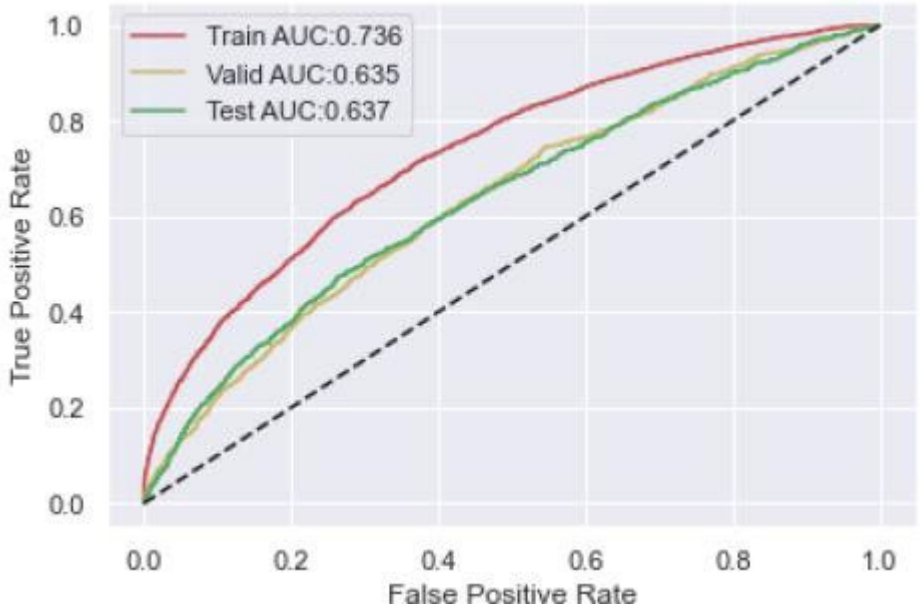

Figure 1-13 ROC Curves of the Best Classifier

Unsupervised Learning

According to IBM Cloud Education (2020), unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Clustering is the most important unsupervised learning problem. K-means is one of the simplest and the most used unsupervised learning algorithms. Compared to other clustering methods, the k-means clustering is fast and efficient. For k-means clustering, SSE (sum of the

squared error) is used as a measure of performance. Also, we need to decide the number of clusters in order to implement the clustering model.

First, we implement the k-means clustering model with k=2, which is a randomly selected number. We use k-means++ to optimize the procedure to initialize the cluster centers.

```
# Initially implement k-means with k=2
kmeans = KMeans(init='k-means++',n_clusters=2,n_init=10,max_iter=300,random_state=40)
```

Then we fit it to the standardized data and get the following statistical results.

```
# The intracluster distance
kmeans.inertia_
```

```
5602905.14669472
```

```
# Final locations of the centroid
kmeans.cluster_centers_
```

```
array([[ 8.38524345e-02,  2.97339414e-03, -1.49221453e-01,
        -2.27352429e-02,  6.28068848e-02,  1.56286887e-01,
         6.55998195e-02,  1.59784757e-01, -1.62693923e-01,
         2.07241632e+00, -1.58375523e-01, -7.87838262e-02,
        -6.57584528e-02, -6.54165058e-02, -5.88323478e-02,
        -5.29548159e-02, -3.78040089e-03,  5.87946990e-02,
         1.03813405e-02,  1.16484587e-02,  1.70797209e-02,
         1.44465689e-02, -7.23859577e-03,  1.05806949e-02,
        -1.25305824e-01, -2.34785882e-02,  1.87255237e-01,
         7.00250445e-02, -5.71341462e-03,  2.09863015e-02,
        -6.54793999e-03, -3.74228052e-01,  3.68176726e-01,
        -2.51441984e-02, -3.24735013e-02, -2.80579314e-02,
         2.40856706e-01, -3.34059823e-02, -1.04500893e-01,
```

```
# The number of iterations required to converge
kmeans.n_iter_
```

```
7
```

According to the results, especially the intracluster distance, we can find that k=2 is not a good choice. We want to minimize the intracluster distance as much as possible. Then we use the elbow method to find the appropriate number of clusters:

```
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(X_std)
    sse.append(kmeans.inertia_)
```

And we plot the sum of squared errors as follows:

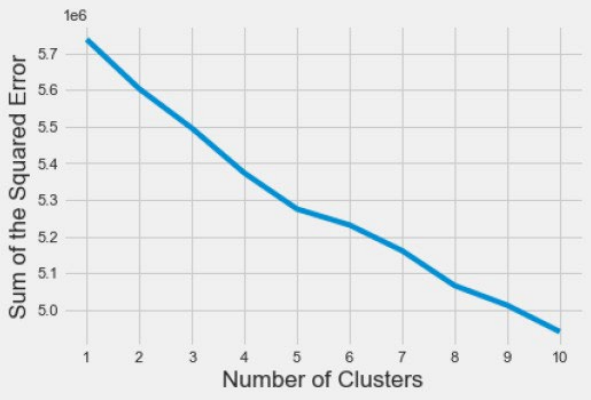

Figure 1-14 Elbow Method: Number of Clusters

Looking at the above elbow curve, we can't easily find the optimum number of clusters. It is probably between 5 and 8. Therefore, we try another method: silhouette coefficients. According to scikit-learn documentation, the Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters.

```
## Use the silhouette coefficients
silhouette_coefficients = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(X_std)
    score = silhouette_score(X_std, kmeans.labels_)
    silhouette_coefficients.append(score)
```

We plot the silhouette coefficients as follows:

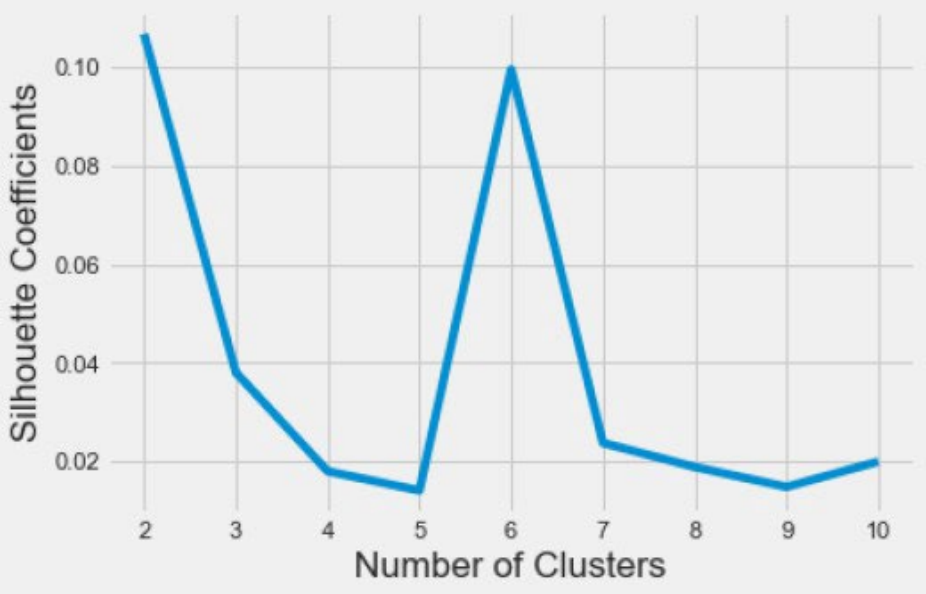

Figure 1-15: Silhouette Coefficients: Number of Clusters

Displaying the average silhouette scores for each k shows that the best choice for k is 6 since it has the highest sore.

After deciding the k value, we create a preprocessing pipeline (standardization and PCA) and a clustering pipeline. In the preprocessing pipeline, we use Principal Component Analysis (PCA) to reduce dimensionality. In the clustering pipeline, we use the best k value and also increase the number of initializations as well as the number of iterations.

```
# Build a preprocessing pipeline
preprocessor = Pipeline([('scaler', StandardScaler()),('pca', PCA(n_components=2, random_state=40)),])
```

```
# Build a clustering pipeline
clusterer = Pipeline([('kmeans',KMeans(n_clusters=6,init='k-means++',n_init=50,max_iter=500,random_state=40,),),])
```

```
# Build a large pipeline
pipe = Pipeline([('preprocessor', preprocessor),('clusterer', clusterer)])
```

We chain those two pipelines into a larger one. Then we perform the large pipeline steps on our unlabeled input data. We get the following silhouette coefficient:

```
silhouette_score(preprocessed_data, predicted_labels)
```

```
0.346826969459858
```

In order to visually demonstrate the result, we create the following plot using the seaborn library:

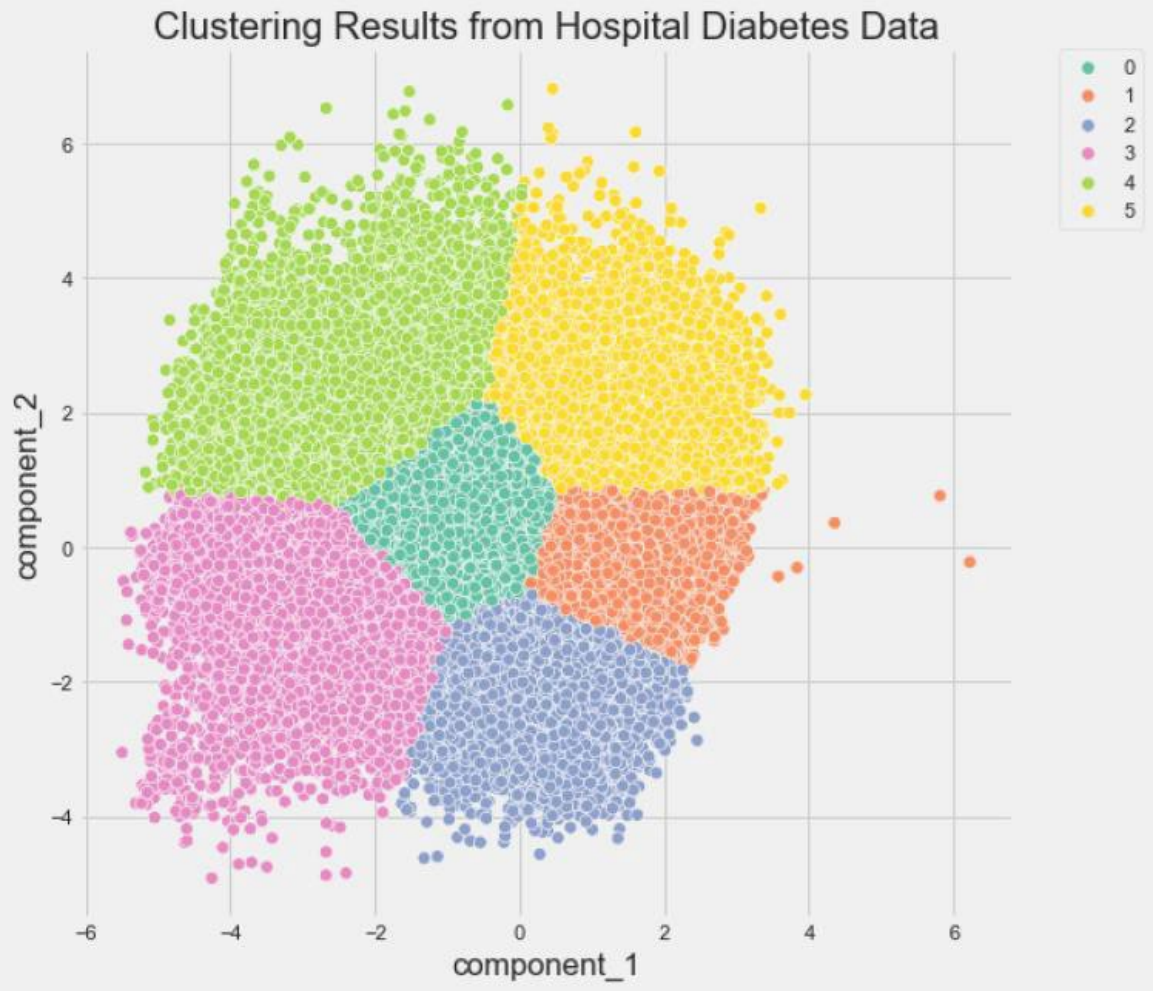

Figure 1-16: Clustering Results from Hospital Diabetes Data

Finally, we predict the clusters by using the following code:

```
y_pred = pipe.fit_predict(X_clu)
```

And we see the value count of points in each of the formed clusters:

```
1    17192
2    14897
0    11701
5    11339
3     8274
4     6570
Name: cluster, dtype: int64
```

So, there are 17,192 data points belonging to cluster 2 (index 1), 11,701 data points in cluster 1 (index 0), and so on.

Conclusion

The aim of this report was to determine the factor(s) that significantly affect readmission of people with diabetes to the hospital within 30 days after discharge and in order to come to valuable conclusion, the project has considered both supervised and unsupervised learning models. Based on the analytical examination of the data, the major factor that significantly impacts readmission to the hospital is discharge disposition and number of medication (please see figures 1.10 and 1.11). Other factors that contribute to readmissions are time in hospital, number of lab procedures, age and number of inpatients. The report is also supported with a high accuracy reading with the Gradient Boosting Classifiers.

What is the aim of these report and how can it help hospitals in dealing with patients suffering from diabetes? This report can provide a framework for hospitals to deals with certain factors at the beginning of (a) treatment or (b) visit to the hospital. It will reduce the number of return visit and it will improve the quality of patient care administered to people with diabetes.

References

Brownlee, J. (2017, July 12). *Machine Learning Mastery*. From https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/

Centers For Disease Control and Prevention. (2019). Diabetes. *What is diabetes?*, 1-2.

Diabetes 130-US hospitals for years 1999-2008 Data Set. (n.d.). From https://archive.ics.uci.edu/ml/datasets/diabetes+130-us+hospitals+for+years+1999-2008

IBM. (2020, Septmeber 21). *IBM Cloud Education-Unsupervised Learning*. From IBM: http://www.ibm.com/cloud/learn/unsupervised-learning

Kuhn, M. J. (2013). Applied Predictive Modeling. Springer Science Business Media.

Laerd Statistics. (2018). *statistics.laerd.com*. From Spearman's Rank-Order Correlation: https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide-2.php

Strack, B. D. (2014). Impact of HbA1c Measurement on Hospital Readmission Rates. *Analysis of 70,000 Clinical Database Patient Records. BioMed Research International, vol. 2014, Article ID 781670, 11 pages*.