Date:10/16/2020
From: Apoorva Bairagi ,Nidhi Vijayvargiya
Subject Working on Diabetes readmission Using Machine Learning Algorithms

INTRODUCTION

Hospital readmission is a real-world challenge and an ongoing topic for enhancing the quality of health care and the experience of a patient, while guaranteeing cost-effectiveness. A substantial proportion of hospital expenses , especially those with serious medical problems, can be traced to a limited number of patients. These expenses are primarily related to frequent hospitalizations with the same illness. Around 20% of all Medicare patients hospitalized are readmitted within 30 days, and 34% are readmitted within 90 days of discharge. As one means of reducing healthcare costs, avoidance of unplanned hospital readmission has also gained growing publicity. The Medicare Premium Advisory Board, for example, has proposed reduced reimbursement rates for patients with early heart failure rehospitalization (CHF).The purpose of this project is to predict when a patient with diabetes will be readmitted within 30 days to the hospital. The goal is to define predictors of unplanned readmissions from hospitals and to explain the role of diabetes diagnosis and glycemic control. The report would then identify generic steps and strategies unique to diabetes in order to stop readmission.

DATA SET DESCRIPTION

At 130 US hospitals and advanced distribution networks, the dataset illustrates 10 years (1999-2008) of health treatment. It contains over 50 features that reflect the results of patients and hospitals. For experiences which met the following conditions, information was retrieved from the database. That is an inpatient experience (admission to the hospital). It is a diabetic experience, that is, one during which as a diagnosis, some form of diabetes has been entered into the system.'The dataset represents 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks. It includes over 50 features representing patient and hospital outcomes. Information was extracted from the database for encounters that satisfied the following criteria. It is an inpatient encounter (a hospital admission).It is a diabetic encounter, that is, one during which any kind of diabetes was entered to the system as a diagnosis.' The duration of the stay was a minimum of 1 day and 14 days at most. During the experience, laboratory experiments were carried out. During the experience, prescriptions were given.

The dataset is from **Kaggle** represents the attributes as patient number, race, gender, age, admission type, time in hospital, medical specialty of admitting physician, number of lab test performed, HbA1c test result, diagnosis, number of medication, diabetic medications, number of outpatient, inpatient, and emergency visits in the year before the hospitalization, etc.

Abstract: This data has been prepared to analyze factors related to readmission as well as other outcomes pertaining to patients with diabetes.

Data Set Characteristics:	Multivariate	**Number of Instances:**	100000	**Area:**	Life
Attribute Characteristics:	Integer	**Number of Attributes:**	55	**Date Donated**	2014-05-03
Associated Tasks:	Classification, Clustering	**Missing Values?**	Yes	**Number of Web Hits:**	322583

STEP PERFORMED TO BUILD THE PROJECT

- Familiarise with the question
- To deal with lost data, perform data scrubbing, and then scale the data.
- Check out which factors have the largest effect on patients with diabetes or who are readmitted, such as age , race, weight , blood pressure, etc.
- Using RF and Gbc to understand how diabetic patients and others that are readmitted are categorized.
- Using acceptable diagrams such as bar charts, measure and present output parameters (e.g. accuracy, etc.) of each model.
- Calculating the tuning of the hyperparameter to see which model is the best model

PROBLEM DEFINITION

Since the data is labeled. We used supervised machine learning for this project where we explored 5 machine learning model to predict if a patient with diabetes will be readmitted to the hospital within 30 days.

EXPECTATIONS : We are trying to figure out situations where the patient will not be readmitted, or if they are going to be readmitted in less than 30 days

IMPORT PYTHON LIBRARIES

```
#Loading libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc_auc_score, roc_curve, recall_score, classification_report
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from numpy import mean
from sklearn.svm import LinearSVC
```

LOAD THE DATASET

```
#Load the dataset

import pandas as pd
diabetic= pd.read_csv('C:/Users/nidhi/Desktop/diabetic_data.csv')
```

IMPORT AND EXPLORE DATA:The original dataset had 55 attributes and 101,731 observations. In order to prepare the dataset for modeling, the data was subjected to intense cleaning procedures. The final dataset created has 91,841 observations and 53 attributes

PERFORMING DATA SCRUBBING (HANDLE MISSING VALUES, SCALING, NORMALIZATION)

```
#Checking for missing values in the data
for col in diabetic.columns:
    if diabetic[col].dtype == object:
         print(col,diabetic[col][diabetic[col] == '?'].count())

print('gender', diabetic['gender'][diabetic['gender'] == 'Unknown/Invalid'].count())
```

- Feature weight contains approximately 98% of the missing values so there is no significance in filling those missing values so we decided to drop this feature
- Feature Payer code and medical specialty contains approximately 40% missing values so we dropped these features too.
- Features like race, diag_1,2,3 and gender contains very less missing values as compared to other attributes which we dropped so for these attributes we also decided to drop those where missing values contains.
- Features (drugs named citoglipton and examide), all records have the same value. So essentially these cannot provide any interpretive or discriminatory information for predicting readmission, so we decided to drop these two features also.

We define a function to calculate the prevalence of population that is readmitted with 30 days.

```
def calc_prevalence(y_actual):
    return (sum(y_actual)/len(y_actual))

print('Prevalence:%.3f'%calc_prevalence(diabetic['OUTPUT_LABEL'].values))
```

```
Prevalence:0.089
```

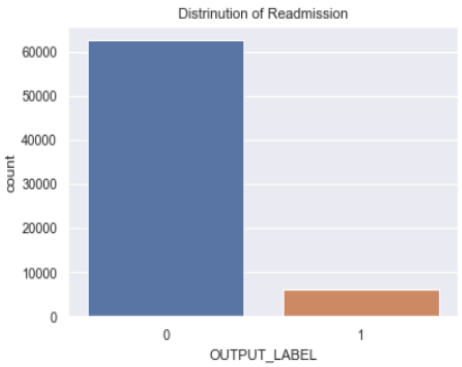
Around **9%** of the population is rehospitalized.

DATA VISUALIZATION

DISTRIBUTION OF READMISSION, PLOTTING BETWEEN TIME IN HOSPITAL AND READMISSION AND OUTPUT LABEL

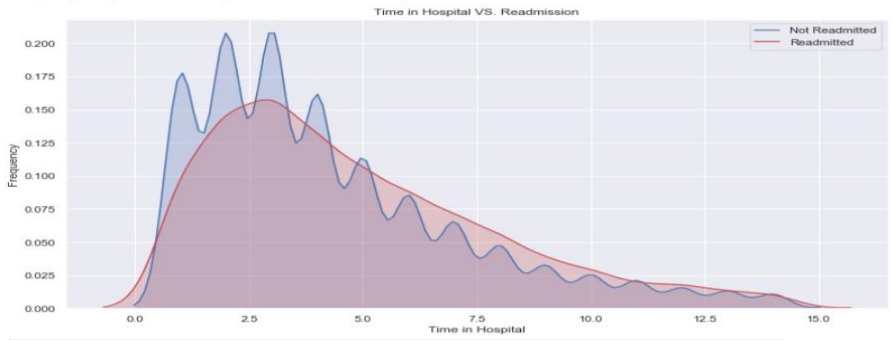
```
# Distribution of Readmission
sns.countplot(diabetic['OUTPUT_LABEL']).set_title('Distrinution of Readmission')
```

```
Text(0.5, 1.0, 'Distrinution of Readmission')
```




```
fig = plt.figure(figsize=(13,7),)
ax=sns.kdeplot(diabetic.loc[(diabetic['OUTPUT_LABEL'] == 0),'time_in_hospital'] , color='b',shade=True,label='Not Readmitted')
ax=sns.kdeplot(diabetic.loc[(diabetic['OUTPUT_LABEL'] == 1),'time_in_hospital'] , color='r',shade=True, label='Readmitted')
ax.set(xlabel='Time in Hospital', ylabel='Frequency')
plt.title('Time in Hospital VS. Readmission')
```

```
Text(0.5, 1.0, 'Time in Hospital VS. Readmission')
```




```
corr_matrix = diabetic.corr()
corr_matrix["OUTPUT_LABEL"].sort_values(ascending=False)
```

```
OUTPUT_LABEL                1.000000
number_inpatient            0.099698
discharge_disposition_id_22 0.088348
discharge_disposition_id_3  0.057535
time_in_hospital            0.052030
                              ...
diag_1                     -0.015372
discharge_disposition_id_11 -0.039080
encounter_id               -0.048829
glimepiride-pioglitazone         NaN
metformin-rosiglitazone          NaN
Name: OUTPUT_LABEL, Length: 93, dtype: float64
```

MODEL SELECTION:BASELINE MODELS

Here we are comparing the performance of 4 machine learning models using default hyperparameters:

- K-nearest neighbors
- Gradient Boosting Classifier
- Naive Bayes
- Random forest

K nearest neighbors (KNN)

KNN is one the simplest machine learning models. For a given sample point, the model looks at the k closest datapoints and determines the probability by counting the number of positive labels divided by K. This model is easy to implement and understand, but comes at the disadvantage of being sensitivity to K and takes a long time to evaluate if the number of trained samples is large. We can fit KNN using the following code from scikit-learn.

```
#k-nearest neighbours
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=100)
knn.fit(X_train_sc, y_train)
```

```
KNeighborsClassifier(n_neighbors=100)
```

```
y_train_preds = knn.predict_proba(X_train_sc)[:,1]
y_valid_preds = knn.predict_proba(X_valid_sc)[:,1]

print('knn')
print('Training:')
knn_train_auc, knn_train_accuracy, knn_train_recall, knn_train_precision, knn_train_specificity =print_report(y_train,y_train_preds, thresh)
print('Validation:')
knn_valid_auc, knn_valid_accuracy, knn_valid_recall, knn_valid_precision, knn_valid_specificity= print_report(y_valid,y_valid_preds, thresh)
```

```
knn
Training:
AUC:0.651
accuracy:0.604
recall:0.455
precision:0.648
specificity:0.713

Validation:
AUC:0.654
accuracy:0.697
recall:0.525
precision:0.146
specificity:0.671
```

Gradient boosting classifier

Another approach to improving decision trees is using a technique called boosting. In this method, you create a bunch of shallow trees that try to improve on the errors of the previously trained trees. One model that uses this technique paired with a gradient descent algorithm (to control the learning rate) is known as gradient boosting classifier. To fit gradient boosting classifier, we can use the following code**.**

```
#Gradient Boosting Classifier
```

```
from sklearn.ensemble import GradientBoostingClassifier
gbc =GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
     max_depth=3, random_state=42)
gbc.fit(X_train_sc, y_train)
```

```
GradientBoostingClassifier(learning_rate=1.0, random_state=42)
```

```
y_train_preds = gbc.predict_proba(X_train_sc)[:,1]
y_valid_preds = gbc.predict_proba(X_valid_sc)[:,1]

print('Gradient Boosting Classifier')
print('Training:')
gbc_train_auc, gbc_train_accuracy, gbc_train_recall, gbc_train_precision, gbc_train_specificity= print_report(y_train,y_train_preds, thresh)
print('Validation:')
gbc_valid_auc, gbc_valid_accuracy, gbc_valid_recall, gbc_valid_precision, gbc_valid_specificity = print_report(y_valid,y_valid_preds, thresh)
```

```
Gradient Boosting Classifier
Training:
AUC:0.809
accuracy:0.726
recall:0.708
precision:0.734
specificity:0.743

Validation:
AUC:0.673
accuracy:0.587
recall:0.646
precision:0.127
specificity:0.582
```

Naive Bayes

Naive Bayes is another model occasionally used in machine learning. In Naive Bayes, we utilize Bayes Rule to calculate the probabilities. The “naive” part of this model is that it assumes all the features are independent (which is generally not the case). This works well for natural language processing models, but let’s try it out here anyways. We can fit Naive Bayes with the following code.

```
#Naive Bayes
from sklearn.naive_bayes import GaussianNB
nb= GaussianNB()
nb.fit(X_train_sc, y_train)
```

```
GaussianNB()
```

```
y_train_preds = nb.predict_proba(X_train_sc)[:,1]
y_valid_preds = nb.predict_proba(X_valid_sc)[:,1]

print('Naive Bayes')
print('Training:')
nb_train_auc, nb_train_accuracy, nb_train_recall, nb_train_precision, nb_train_specificity =print_report(y_train,y_train_preds, thresh)
print('Validation:')
nb_valid_auc, nb_valid_accuracy, nb_valid_recall, nb_valid_precision, nb_valid_specificity= print_report(y_valid,y_valid_preds, thresh)
```

```
Naive Bayes
Training:
AUC:0.519
accuracy:0.514
recall:0.992
precision:0.507
specificity:0.036

Validation:
AUC:0.500
accuracy:0.914
recall:0.000
precision:0.000
specificity:1.000
```

Random forest

One disadvantage of decision trees is that they tend overfit very easily by memorizing the training data. As a result, random forests were created to reduce the overfitting. In random forest models, multiple trees are created and the results are aggregated. The trees in a forest are decorrelated by using a random set of samples and random number of features in each tree. In most cases, random forests work better than decision trees because they are able to generalize more easily. To fit random forests, we can use the following code.

```
#Random Forest
```

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
from sklearn.ensemble import RandomForestClassifier
#rf = RandomForestClassifier(n_estimators = 10, max_depth=25, criterion = "gini", min_samples_split=10)
rf=RandomForestClassifier(max_depth = 6, random_state = 42)
model=rf.fit(X_train_sc, y_train)
```

```
y_train_preds = rf.predict_proba(X_train_sc)[:,1]
y_valid_preds = rf.predict_proba(X_valid_sc)[:,1]

print('Random Forest')
print('Training:')
rf_train_auc, rf_train_accuracy, rf_train_recall, rf_train_precision, rf_train_specificity =print_report(y_train,y_train_preds, thresh)
print('Validation:')
rf_valid_auc, rf_valid_accuracy, rf_valid_recall, rf_valid_precision, rf_valid_specificity = print_report(y_valid,y_valid_preds, thresh)
```

```
Random Forest
Training:
AUC:0.684
accuracy:0.626
recall:0.558
precision:0.646
specificity:0.694

Validation:
AUC:0.663
accuracy:0.655
recall:0.549
precision:0.133
specificity:0.665
```

MODEL SELECTION: FEATURE IMPORTANCE

We can get the feature importance from logistic regression using the following

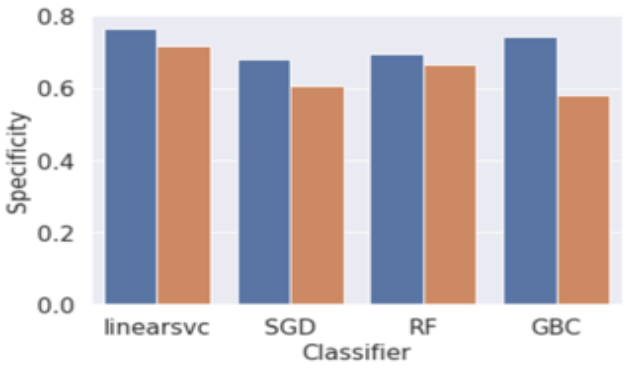
```
#Feature Importance: random forest
```

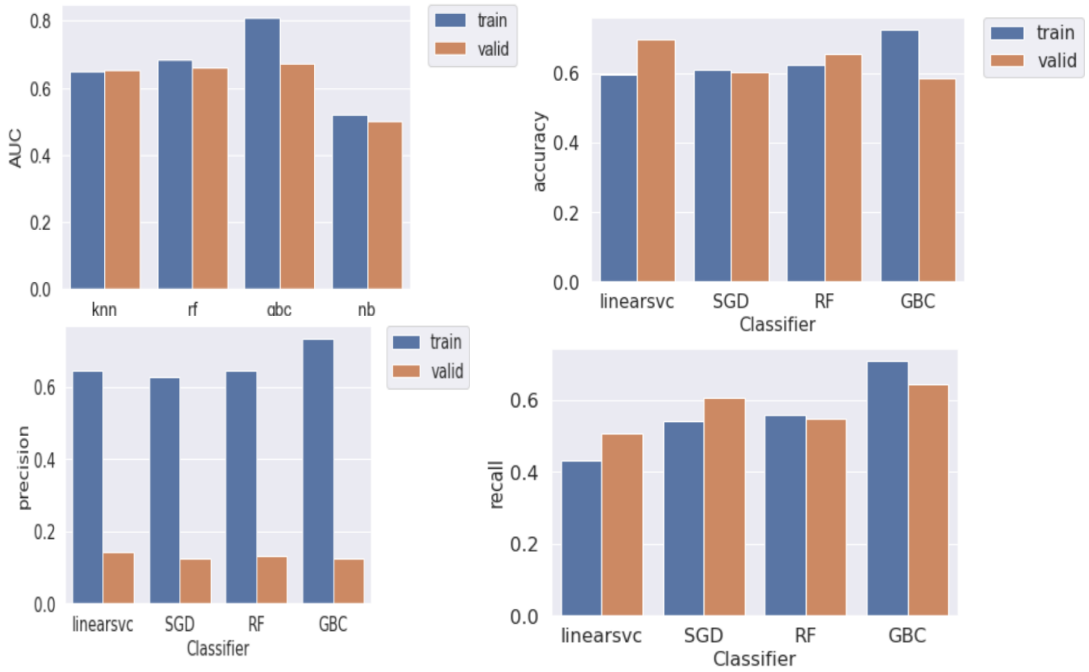
```
feature_importances = pd.DataFrame(rf.feature_importances_,
                                   index = feature,
                                    columns=['importance']).sort_values('importance',
                                                                        ascending=False)
feature_importances.head()
```

	importance
number_inpatient	0.158744
discharge_disposition_id_22	0.109983
discharge_disposition_id_3	0.089603
time_in_hospital	0.078903
discharge_disposition_id_11	0.071552

ANALYSIS OF BASELINE MODELS

With the results of all the baseline models, we have created a data frame and mapped the results using a package called seaborn. We have used the field under the ROC curve (AUC) in this project to evaluate the best model. This is a good value indicator for data science to choose the best model because it captures the trade off between the true positive and the false positive and does not need a threshold to be selected.





MODEL SELECTION: LEARNING CURVE

By following a learning curve, we can diagnose how our models are doing. In this section, with a slight adjustment to plotting the AUC instead of accuracy, we will use the learning curve code from the sci-kit-learn website. We can see that the training and validation scores are close in the case of random forests, but they both have low AUC scores. This is called high bias and is an indicator of underfitting. There are a few strategies we can use to develop the models, depending on the learning curve.

High Bias:

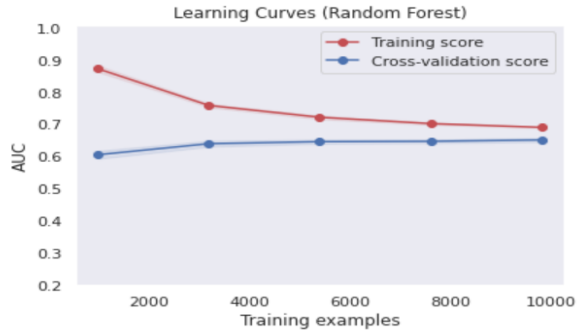
- Add new features
- Increase model complexity
- Reduce regularization
- Change model architecture

High Variance:

- Add more samples
- Add regularization
- Reduce number of features
- Decrease model complexity
- Add better features
- Change model architecture

```
title = "Learning Curves (Random Forest)"
# Cross validation with 5 iterations to get smoother mean test and train
# score curves, each time with 20% data randomly selected as a validation set.
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
estimator = RandomForestClassifier(max_depth = 6, random_state = 42)
plot_learning_curve(estimator, title, X_train_sc, y_train, ylim=(0.2, 1.01), cv=cv, n_jobs=4)

plt.show()
```



MODEL SELECTION: HYPERPARAMETER TUNING

Hyperparameter tuning are essentially the design decisions that you made when you set up the machine learning model. Each of the hyperparameters can be optimized to improve the model.Here we are only optimize the hyper parameters for random forest and gradient boosting classifier. We will not optimize KNN since it took a while to train. A Grid search, where we test all possible combinations over a grid of values, is one technique for hyperparameter tuning. The other choice is to randomly evaluate them for a permutation. In scikit-learn, this method called Random Search is also implemented.

```
y_train_preds = rf.predict_proba(X_train_sc)[:,1]
y_valid_preds = rf.predict_proba(X_valid_sc)[:,1]

print('Baseline Random Forest')
rf_train_auc_base = roc_auc_score(y_train, y_train_preds)
rf_valid_auc_base = roc_auc_score(y_valid, y_valid_preds)

print('Training AUC:%.3f'%(rf_train_auc_base))
print('Validation AUC:%.3f'%(rf_valid_auc_base))

print('Optimized Random Forest')
y_train_preds_random = rf_random.best_estimator_.predict_proba(X_train_sc)[:,1]
y_valid_preds_random = rf_random.best_estimator_.predict_proba(X_valid_sc)[:,1]

rf_train_auc = roc_auc_score(y_train, y_train_preds_random)
rf_valid_auc = roc_auc_score(y_valid, y_valid_preds_random)

print('Training AUC:%.3f'%(rf_train_auc))
print('Validation AUC:%.3f'%(rf_valid_auc))
```

```
Baseline Random Forest
Training AUC:0.684
Validation AUC:0.663
Optimized Random Forest
Training AUC:0.716
Validation AUC:0.681
```

```
y_train_preds = gbc.predict_proba(X_train_sc)[:,1]
y_valid_preds = gbc.predict_proba(X_valid_sc)[:,1]

print('Baseline gbc')
gbc_train_auc_base = roc_auc_score(y_train, y_train_preds)
gbc_valid_auc_base = roc_auc_score(y_valid, y_valid_preds)

print('Training AUC:%.3f'%(gbc_train_auc_base))
print('Validation AUC:%.3f'%(gbc_valid_auc_base))

print('Optimized gbc')
y_train_preds_random = gbc_random.best_estimator_.predict_proba(X_train_sc)[:,1]
y_valid_preds_random = gbc_random.best_estimator_.predict_proba(X_valid_sc)[:,1]
gbc_train_auc = roc_auc_score(y_train, y_train_preds_random)
gbc_valid_auc = roc_auc_score(y_valid, y_valid_preds_random)

print('Training AUC:%.3f'%(gbc_train_auc))
print('Validation AUC:%.3f'%(gbc_valid_auc))
```

```
Baseline gbc
Training AUC:0.809
Validation AUC:0.673
Optimized gbc
Training AUC:0.692
Validation AUC:0.677
```

SELECTION MODEL: BEST CLASSIFIER

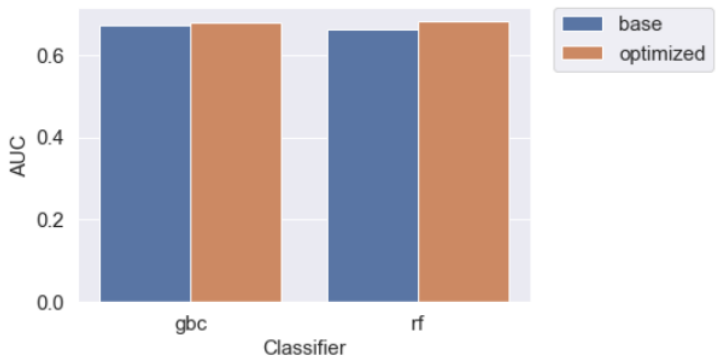
We will pick the gradient boosting classifier here because it has the best AUC in the validation set. Any time you try to run new predictions, you may not want to train your best classifier. Therefore, we need the classifier to be saved. We're going to use the package pickle.

HYPERPARAMETER TUNNING RESULTS

	classifier	data_set	auc
0	gbc	base	0.672682
1	gbc	optimized	0.676998
2	rf	base	0.662602
3	rf	optimized	0.680851

We can aggregate the results and compare to the baseline models

```
ax = sns.barplot(x="classifier", y="auc", hue="data_set", data=df_results)
ax.set_xlabel('Classifier',fontsize = 15)
ax.set_ylabel('AUC', fontsize = 15)
ax.tick_params(labelsize=15)
# Put the legend out of the figure
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., fontsize = 15)
plt.show()
```



We can see that the hyperparameter tuning, but not by much, enhanced the models. This is most definitely due to the fact that we are in a situation with high bias. If we had higher variance, more change would be expected.

MODEL EVALUATION

We have now chosen our best (optimized gradient boosting classifier) model. Let's evaluate the test set 's results.

```
#Model Evaluation
#Now that we have selected our best model. Let's evaluate the performance of the test set.
X_test = df_test[feature].values
y_test = df_test['OUTPUT_LABEL'].values
X_test_sc = scaler.fit_transform(X_test)
```

```
best_model= gbc_random.best_estimator_
```

```
y_train_preds = best_model.predict_proba(X_train_sc)[:,1]
y_valid_preds = best_model.predict_proba(X_valid_sc)[:,1]
y_test_preds = best_model.predict_proba(X_test_sc)[:,1]
```

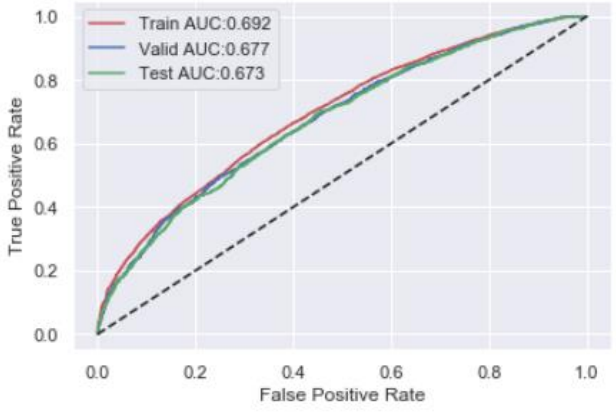
The final evaluation as shown:

```
Training:
AUC:0.692
accuracy:0.631
recall:0.558
precision:0.653
specificity:0.703

Validation:
AUC:0.677
accuracy:0.669
recall:0.557
precision:0.140
specificity:0.680

Test:
AUC:0.673
accuracy:0.674
recall:0.548
precision:0.150
specificity:0.687
```

We can also create the ROC curve for the 3 datasets as shown below:



RESULTS

Through this project, we have developed a machine learning model that can estimate the highest risk of re-admission within 30 days for patients with diabetes. A gradient boosting classifier with optimized hyperparameters was the best one. The model was able to catch 58% of readmissions which is around 1.5 times better than choosing patients at random. Overall , I agree that many healthcare data scientists are focusing on hospital readmission predictive models.

CONCLUSION

We examined how the effects of prediction modeling using readmission for patients with a diabetes diagnosis as the context for the research would influence different data preprocessing techniques such as feature selection, missing value imputation and class balancing techniques. This improved dataset (after data preprocessing) was then applied to different predictive models such as KNN algorithm, Naïve Bayes and Random Forest, Gradient boosting classifier, to obtain the risk of accuracy of readmission predictions. In conjunction with other medical diseases, the real-world hospital data of hospital patients with diabetes is analyzed as an existing condition. The goal was to build a predictive model to identify patients with a higher risk of being readmitted. The impact of different pre-processing choices was assessed on various performance metrics like Area under Curve (AUC), Precision, Recall and Accuracy. We will pick the gradient boosting classifier here because it has the best AUC in the validation set.

FUTURE WORK

There are many ways to enhance the model above. Here are some definitions.

- Bucketing of input features should help, i.e. create new variables in a specific range for blood pressure, glucose levels in a particular size, and so on.

- Data cleaning may also be strengthened by replacing 0 values with the mean value.

- Reading a little information on what measures physicians depend on to diagnose a patient with diabetes and develop new features accordingly.