HOSPITAL MORTALITY PREDICTION

PROJECT 1

By:
Priyanka
Chandramohan

1. Introduction

The predictors of in-hospital mortality for intensive care units (ICU)-admitted HF patients remain poorly characterized. We aimed to develop and validate a prediction model for all-cause in-hospital mortality among ICU-admitted HF patients.

- We will try to predict whether the patient mortality in a hospital.

Main Goal

- Create an analytical framework to understand
  - Key factors impacting hospital mortality.
- Develop a modelling framework
  - To estimate the whether the patient mortality in a hospital.

About Dataset

- The MIMIC-III database (version 1.4, 2016) is a publicly available critical care database containing de-identified data on 46,520 patients and 58,976 admissions to the ICU of the Beth Israel Deaconess Medical Center, Boston, USA, between 1 June, 2001 and 31 October, 2012.

This is what the dataset looks like,

	group	ID	outcome	age	gendera	BMI	hypertensive	atrialfibrillation	CHD with no MI	diabetes	deficiencyanemias	depression	Hyperlipemia	Renal failure
0	1	125047	0.0	72	1	37.588179	0	0	0	1	1	0	1	1
1	1	139812	0.0	75	2	NaN	0	0	0	0	1	0	0	0
2	1	109787	0.0	83	2	26.572634	0	0	0	0	1	0	0	1
3	1	130587	0.0	43	2	83.264629	0	0	0	0	0	0	0	0
4	1	138290	0.0	75	2	31.824842	1	0	0	0	1	0	0	1

COPD	heart rate	Systolic blood pressure	Diastolic blood pressure	Respiratory rate	temperature	SP O2	Urine output	hematocrit	RBC	MCH	MCHC	MCV	RDW
0	68.837838	155.866667	68.333333	16.621622	36.714286	98.394737	2155.0	26.272727	2.960000	28.250000	31.520000	89.900	16.220000
1	101.370370	140.000000	65.000000	20.851852	36.682540	96.923077	1425.0	30.780000	3.138000	31.060000	31.660000	98.200	14.260000
0	72.318182	135.333333	61.375000	23.640000	36.453704	95.291667	2425.0	27.700000	2.620000	34.320000	31.300000	109.800	23.820000
0	94.500000	126.400000	73.200000	21.857143	36.287037	93.846154	8760.0	36.637500	4.277500	26.062500	30.412500	85.625	17.037500
1	67.920000	156.560000	58.120000	21.360000	36.761905	99.280000	4455.0	29.933333	3.286667	30.666667	33.666667	91.000	16.266667

Table1- DataSet

It has lots of rows and columns, which is 1177 rows spread across 51 columns. Here is the list of columns this dataset has,

```
1 data.columns
```

```
Index(['group', 'ID', 'outcome', 'age', 'gendera', 'BMI', 'hypertensive',
       'atrialfibrillation', 'CHD with no MI', 'diabetes', 'deficiencyanemias',
       'depression', 'Hyperlipemia', 'Renal failure', 'COPD', 'heart rate',
       'Systolic blood pressure', 'Diastolic blood pressure',
       'Respiratory rate', 'temperature', 'SP O2', 'Urine output',
       'hematocrit', 'RBC', 'MCH', 'MCHC', 'MCV', 'RDW', 'Leucocyte',
       'Platelets', 'Neutrophils', 'Basophils', 'Lymphocyte', 'PT', 'INR',
       'NT-proBNP', 'Creatine kinase', 'Creatinine', 'Urea nitrogen',
       'glucose', 'Blood potassium', 'Blood sodium', 'Blood calcium',
       'Chloride', 'Anion gap', 'Magnesium ion', 'PH', 'Bicarbonate',
       'Lactic acid', 'PCO2', 'EF'],
      dtype='object')
```

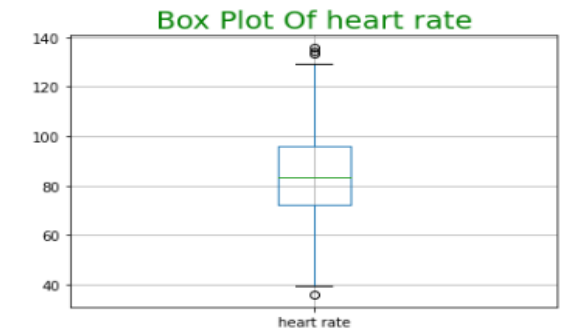
Table 2 - Columns in

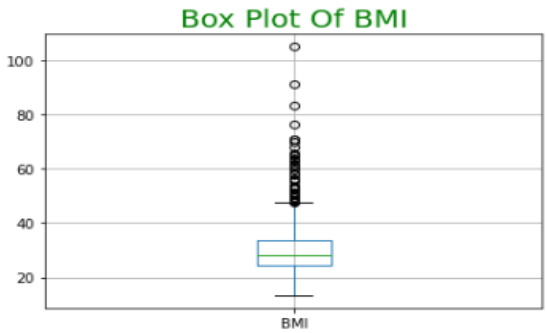
Analysis Of the Data

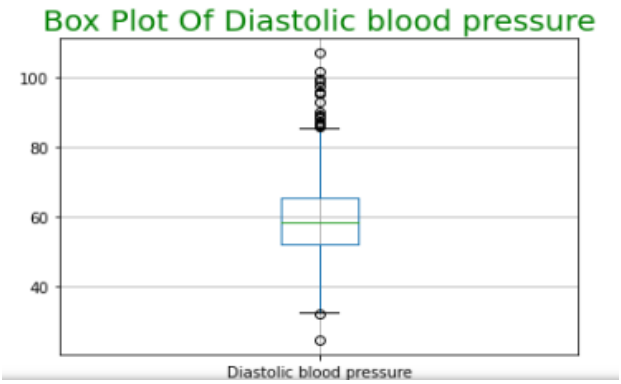
	count	mean	std	min	25%	50%	75%	max
group	1177.0	1.299065	0.458043	1.000000	1.000000	1.000000	2.000000	2.000000
ID	1177.0	150778.120646	29034.669513	100213.000000	125603.000000	151901.000000	176048.000000	199952.000000
outcome	1176.0	0.135204	0.342087	0.000000	0.000000	0.000000	0.000000	1.000000
age	1177.0	74.055225	13.434061	19.000000	65.000000	77.000000	85.000000	99.000000
gendera	1177.0	1.525064	0.499584	1.000000	1.000000	2.000000	2.000000	2.000000
BMI	962.0	30.188278	9.325997	13.346801	24.326461	28.312474	33.633509	104.970366
hypertensive	1177.0	0.717927	0.450200	0.000000	0.000000	1.000000	1.000000	1.000000
atrialfibrillation	1177.0	0.451147	0.497819	0.000000	0.000000	0.000000	1.000000	1.000000
CHD with no MI	1177.0	0.085811	0.280204	0.000000	0.000000	0.000000	0.000000	1.000000
diabetes	1177.0	0.421410	0.493995	0.000000	0.000000	0.000000	1.000000	1.000000
deficiencyanemias	1177.0	0.338997	0.473570	0.000000	0.000000	0.000000	1.000000	1.000000
depression	1177.0	0.118946	0.323863	0.000000	0.000000	0.000000	0.000000	1.000000
Hyperlipemia	1177.0	0.379779	0.485538	0.000000	0.000000	0.000000	1.000000	1.000000
Renal failure	1177.0	0.365336	0.481729	0.000000	0.000000	0.000000	1.000000	1.000000
COPD	1177.0	0.075616	0.264495	0.000000	0.000000	0.000000	0.000000	1.000000
heart rate	1164.0	84.575848	16.018701	36.000000	72.371250	83.610799	95.907143	135.708333
Systolic blood pressure	1161.0	117.995035	17.367618	75.000000	105.391304	116.128205	128.625000	203.000000
Diastolic blood pressure	1161.0	59.534497	10.684681	24.736842	52.173913	58.461538	65.464286	107.000000
Respiratory rate	1164.0	20.801511	4.002987	11.137931	17.925694	20.372308	23.391200	40.900000
temperature	1158.0	36.677286	0.607558	33.250000	36.286045	36.650794	37.021991	39.132478
SP O2	1164.0	96.272900	2.298002	75.916667	95.000000	96.452273	97.917500	100.000000
Urine output	1141.0	1899.276512	1272.363631	0.000000	980.000000	1675.000000	2500.000000	8820.000000
hematocrit	1177.0	31.914014	5.202102	20.311111	28.160000	30.800000	35.012500	55.425000
RBC	1177.0	3.575010	0.626835	2.030000	3.120000	3.490000	3.900000	6.575000
MCH	1177.0	29.539939	2.619054	18.125000	28.250000	29.750000	31.240000	40.314286
MCHC	1177.0	32.864327	1.402302	27.825000	32.011111	32.985714	33.825000	37.011111
MCV	1177.0	89.903812	6.532629	62.600000	86.250000	90.000000	93.857143	116.714286
RDW	1177.0	15.952129	2.131643	12.088889	14.460000	15.506250	16.937500	29.050000

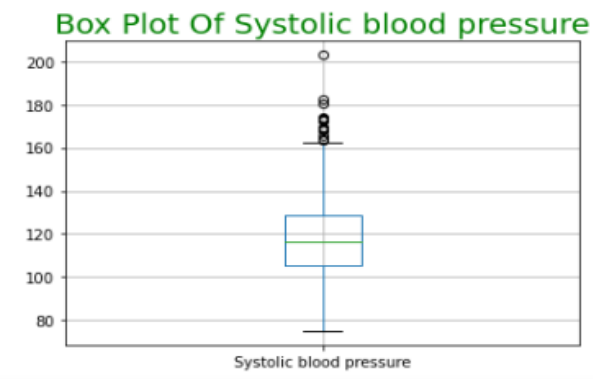
Table3- Checking of Skewness

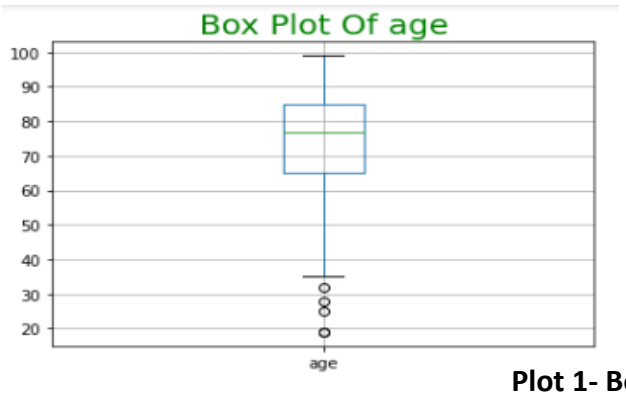
Viewing Outliers

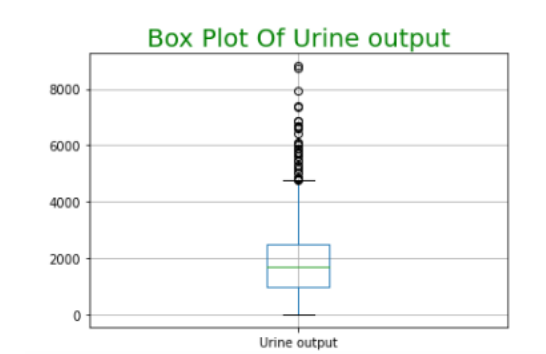










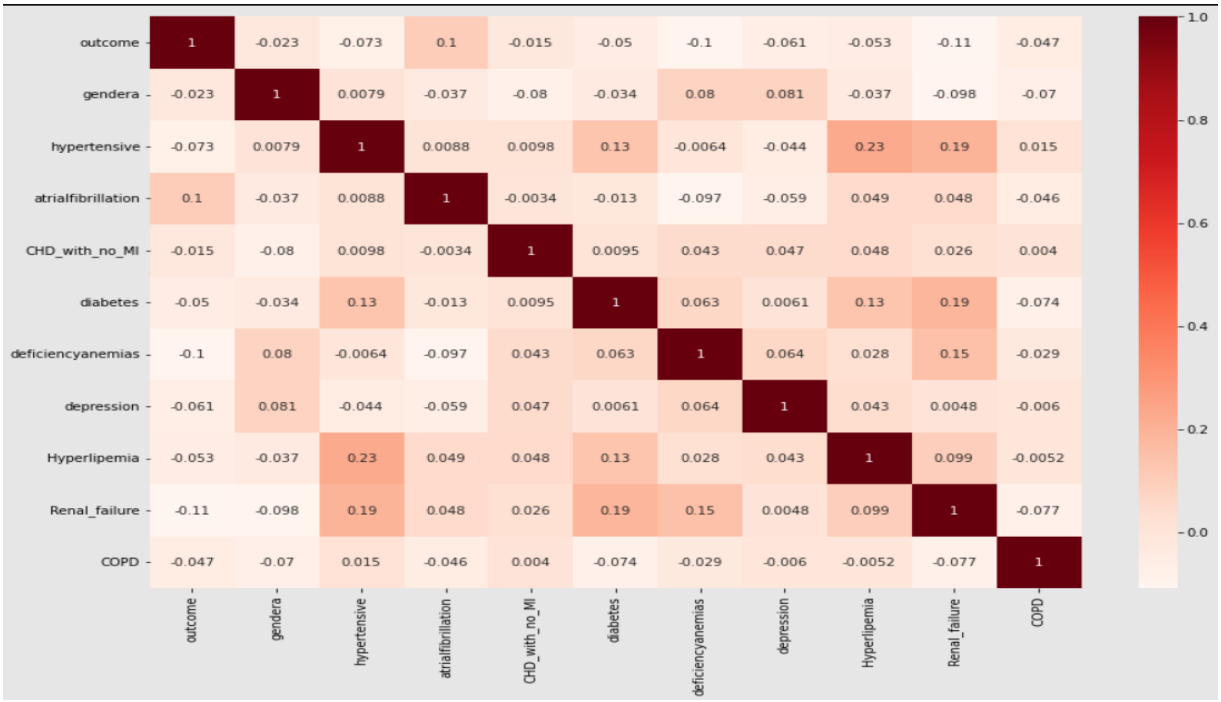


Plot 1- Boxplot

Observation:

There are lots of feature which are having outliers.

Heatmap



Plot – 2: Heatmap

2. Data Cleaning & Pre-processing

Data Cleaning is an important phase in any data science project, if our data is clean then only we can provide it to our machine learning model. Uncleaned Data can further lead our model with low accuracy. And, If data is incorrect, outcomes and algorithms are

unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset.

The approach used for identifying and treating missing values and outlier treatment:-

```
1  data.isnull().sum()
2  data.isna().sum()
```

```
group                          0
ID                             0
outcome                        1
age                            0
gendera                        0
BMI                          215
hypertensive                   0
atrialfibrillation             0
CHD_with_no_MI                 0
diabetes                       0
deficiencyanemias              0
depression                     0
Hyperlipemia                   0
Renal_failure                  0
COPD                           0
heart_rate                    13
Systolic_blood_pressure       16
Diastolic_blood_pressure      16
Respiratory rate              13
```

Code Sample 1- Null Values

To identify any missing values in our data set we have used Pandas pre built function isnull() to detect any missing values in our datasets. As we can see that column many columns has a high number of missing values. So our next step is how to handle a large number of missing values. One approach is, that we will delete the column if we don't need that column for further analysis. And, what if we need that column for further analysis then we have use an approach will is a predefined function in Pandas called fillna().

```
for i in data.columns:
    if data[i].isnull().sum()>1:
        data[i]=data[i].fillna(data[i].mean())


data = data.dropna(how='any',axis=0)
data = data.drop(['ID'],axis=1)
data = data.drop(['group'],axis=1)
```

Code Sample 2- Imputing Null Values

As you can have a look, How we have filled the missing values in a categorical variable using mode. And, How we have filled the missing values in a numerical variable using Median. This is how we have an approach for identifying and handling missing values.

While performing Preprocessing and Data cleaning we have to also deal with outliers. Outliers are data points in a data set that is distant from all other observations.

A data point that lies outside the overall distribution of the dataset.

```
columns = ['age','gendera','BMI','heart rate','Systolic blood pressure','Diastolic blood pressure',
           'Respiratory rate','temperature','Urine output']
for i in columns:
    data.iloc[:,:].boxplot(column = i)
    plt.title(f"Box Plot Of {i}" , fontsize=20,
          color="green")
    plt.show()
```

Code Sample 3- Outlier Detection

Need For Variable Transformation:-

```
plt.figure(figsize=(12,8))
plt.subplot(1,2,1)
labels=['Live','Died']
plt.title('Pie chart', fontweight='bold', fontsize='14',fontfamily='sans-serif',color='red')
plt.pie(data['outcome'].value_counts(),labels=labels, pctdistance=0.7,autopct='%.2f%%', wedgeprops=dict(alpha=0.8, edgecolor
        textprops={'fontsize':12})
centre=plt.Circle((0, 0), 0.45, fc='white', edgecolor='black')
plt.gcf().gca().add_artist(centre)


countplt = plt.subplot(1, 2, 2)
plt.title('Histogram', fontweight='bold', fontsize=14,
          fontfamily='sans-serif', color='red')
ax = sns.countplot(x='outcome', data=data,
                   edgecolor='black', alpha=0.85)
```

Code Sample 4 – Variable Transformation

Variable transformation is a way to make the data work better in your model. Here specifically we have Replaced the unwanted symbol and nan value with NumPy nan values i.e np. nan which we are going to deal in missing value. And we have also converted the data types of few columns which will help build our model. The need for this is because we need our model to have a good score and accuracy which will make good predictions. So to feed the data to our model we must ensure to take these steps and make our data insightful.

Variables removed or added and why?

Variables are removed in such a scenario where we have a large number of null values in any columns particularly or else in our dataset, to make our data clean and ready for modelling. Whereas, variables are added in such a scenario where Consider an example where we have a column as start-date and another column as end-date so we can create a column named 'difference' where we subtract the start-date and end-date and have a number of days between them in a column named 'difference' and later drop start-date and end-date as if they are of no use.

3. Modeling Building

Model Selection and Why?

After cleaning and processing the data then comes the modeling part which includes building Machine Learning models, let's first understand in brief what Machine Learning is?

Machine Learning is a technique that analyzes past data and tries to extract meaningful insights and patterns from them which can be further used to perform predictions in future.

For example, classifying whether a tumor is benign or malignant, predicting stock prices, etc. One such application which we're using right here is predicting house prices. Before making predictions first we need to build a model and train it using past data.

First, we need to separate the dataset into two parts: features (health attributes) and labels (mortality) which is the required format for any model to be trained on.

Then the data needs to be split into 3 sets

1. Training set - This will be the part of the dataset which the model will be using to train itself, the size should be at least 60-70% of the total data we've.

2. Validation set - This set is used for validating our model's performance for a different set of hyperparameters. After taking out the train set, the remaining set can be split into validation and test set.

3. Testing set - To evaluate how the model is performing on the unseen data on which the model will be doing future predictions on, test set is used. It helps to understand how much error is there between actual and predicted values.

```
X=new_data.drop(['outcome'],axis=1)
Y= new_data['outcome']
```

```
from sklearn.model_selection import train_test_split
```

```
x_train , x_test , y_train , y_test = train_test_split(X,Y,test_size=0.1,random_state=1)
```

```
print(x_train.shape,x_test.shape)
```

```
(286, 48) (32, 48)
```

Code Sample 5 - Splitting Data

We need to build different Classification algorithms and using the validation set we can determine which model to keep for making final predictions.

Here is the list of all the algorithms we've to build and evaluated:

Here is the list of all the algorithms we've to build and evaluated:

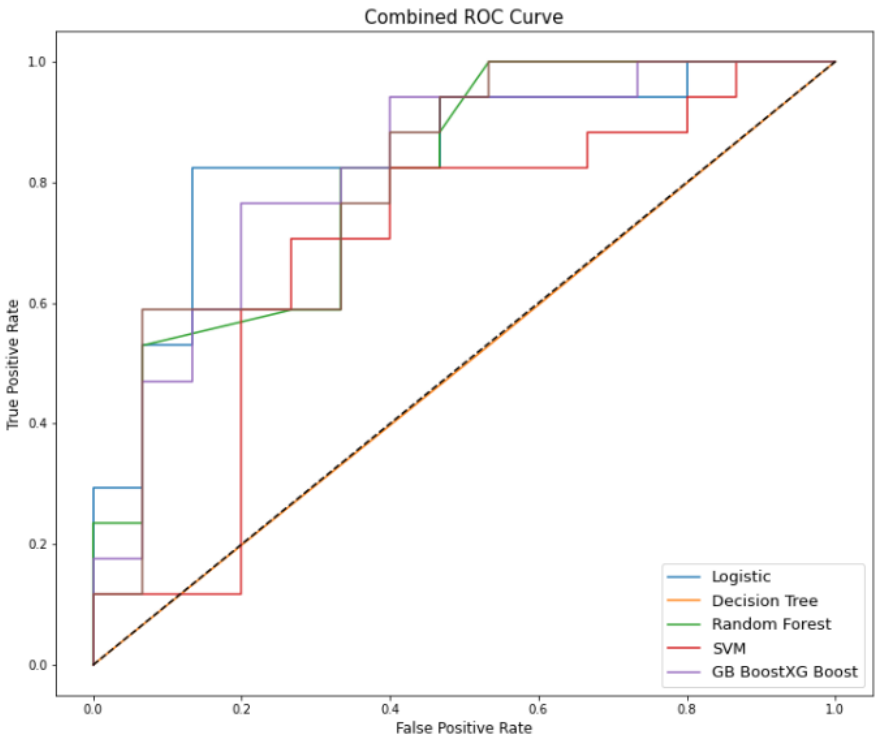
1. Logistic Regression
2. Logistic Regression with Grid Search CV
3. **SVC**
4. SVC with Random Search CV
5. Decision Tree
6. Decision Tree with Grid Search CV
7. Random Forest
8. Random Forest with Grid Search CV
9. Gradient Boosting
10. Gradient Boosting with Grid Search CV
11. XGboost
12. Xgboost with Random Search CV

	Classifier	Train Accuracy	Test Accuracy	Precision	recall_score	f1_score	auc_score
0	Logistic Regression	0.769	0.812	0.824	0.824	0.824	0.812
4	Gradient Boosting	1.000	0.719	0.824	0.700	0.757	0.725
2	Random Forest	1.000	0.688	0.824	0.667	0.737	0.697
5	XG Boosting	1.000	0.688	0.765	0.684	0.722	0.688
1	Decision Tree	1.000	0.500	0.529	0.529	0.529	0.498
3	SVM	0.601	0.500	0.235	0.571	0.333	0.526

Here we can see that Random forest classifier shows highest test accuracy and F1 score.

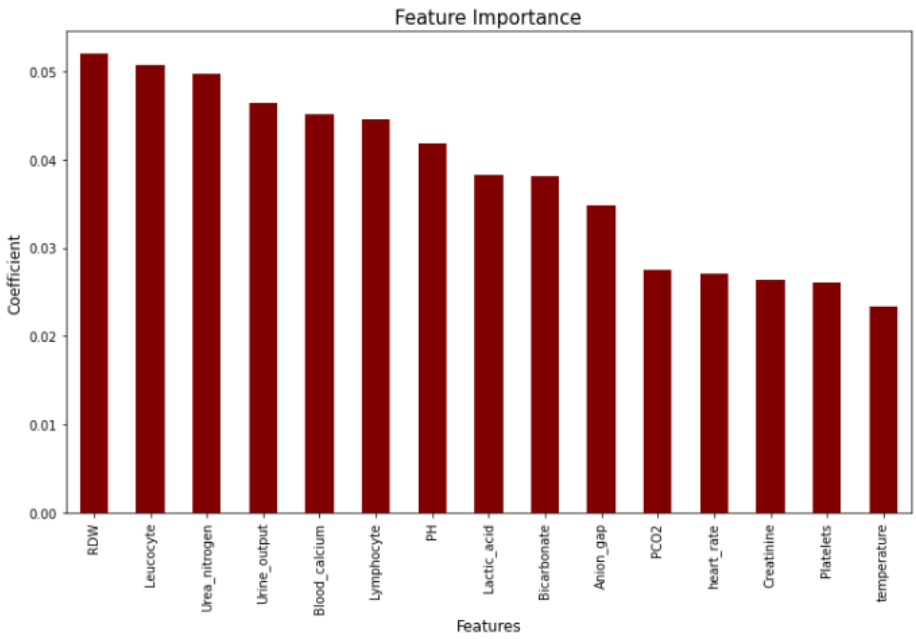
Table 3 : Classification Result

ROC CURVE



Plot 3 - ROC AUC CURVE

4. Feature Importance



Plot 4 – Feature Importance

Efforts to improve model performance

Hyperparameter Tuning

Hyperparameter tuning is the process of trying out a different set of model parameters, actually, they are algorithm's parameter for example theta1 and theta2 in the hypothesis function for Linear Regression is model parameter, but lambda which is a factor that decides the amount of regularization is algorithm's parameter called as a hyperparameter.

Tuning hyperparameters helps us to find out the optimal parameter values for which the model is giving less error and a better overall score. We've performed tuning for Gradient

Boosting Regressor with both the methods RandomizedSearchCV as well as GridSearchCV. What random search does is from the given set of parameter values, it tries out n number of combinations whereas grid search builds a model with all the possible combinations from the given set of parameters and gives us the optimal values.

Grid Search CV

These are the parameters that we've given to Grid Search to find out the best one.

We have performed hyper parameter tuning on the following algorithms:

1. Logistic Regression
2. SVC
3. Decision Tree
4. Random Forest
5. Gradient Boost
6. XGBoost

Logistic Regression

```
2  lr_grid.best_params_
```

```
{'C': 5, 'max_iter': 10000, 'penalty': 'l2'}
```

Code Sample 5 - Best parameter grid Logistic Regression

```
# Create an instance of the Logistic Regression
lr = LogisticRegression()

# Grid search
lr_grid = GridSearchCV(estimator=lr,
                       param_grid = param_dict,
                       cv = 5, verbose=3, n_jobs = -1, scoring='roc_auc')
# fitting model
lr_grid.fit(x_train,y_train)
```

Code Sample 6 – Hyper parameter Tuning Logistic

Regression

```
1  dtc_grid.best_params_
```

```
{'max_depth': 20, 'min_samples_leaf': 40, 'min_samples_split': 0.001}
```

Code Sample 7 - Best parameter grid Decision Tree

```
# Create an instance of the decision tree
dtc = DecisionTreeClassifier()

# Grid search
dtc_grid = GridSearchCV(estimator=dtc,
                        param_grid = param_dict,
                        cv = 5, verbose=3, n_jobs = -1, scoring='roc_auc')
# fitting model
dtc_grid.fit(x_train, y_train)
```

Code Sample 8 – Hyper parameter Tuning Decision Tree

```
1  svm_grid.best_params_
```

```
{'kernel': 'rbf', 'C': 10}
```

Code Sample 9 - Best parameter grid SVC

```
# Create an instance of the support vector classifier
svm=SVC(probability=True)

# Grid search
svm_grid = RandomizedSearchCV(estimator = svm, param_distributions = param_dict,
                      cv = 2, verbose=2, n_jobs = -1, scoring= 'roc_auc')
# fitting model
svm_grid.fit(x_train, y_train)
```

Code Sample 10 – Hyper parameter Tuning SVC

```
1  rfc_grid.best_params_
```

```
{'max_depth': 10,
 'min_samples_leaf': 40,
 'min_samples_split': 100,
 'n_estimators': 150}
```

Code Sample 11 - Best parameter grid Random Forest

```
# Create an instance of the RandomForestClassifier
rfc = RandomForestClassifier()

# Grid search
rfc_grid = GridSearchCV(estimator=rfc,
                        param_grid = param_dict,
                        cv = 5, verbose=2, scoring='roc_auc')
# fitting model
rfc_grid.fit(x_train,y_train)
```

Code Sample 12 – Hyper parameter Tuning Random Forest

```
gbc_grid.best_params_
```

```
{'n_estimators': 200,
 'min_samples_split': 50,
 'min_samples_leaf': 50,
 'max_depth': 10,
 'learning_rate': 0.05}
```

Code Sample 13 - Best parameter grid GBC

```
# Create an instance of the RandomForestClassifier
gbc = GradientBoostingClassifier(random_state=42)

# Grid search
gbc_grid = RandomizedSearchCV(estimator=gbc,
                      param_distributions = param_dict,
                      cv = 2, verbose=2, scoring='roc_auc')
# fitting model
gbc_grid.fit(x_train,y_train)
```

Code Sample 14 – Hyper parameter Tuning GBC

```
xgb_grid.best_params_
```

```
{'n_estimators': 200,
 'min_samples_leaf': 50,
 'min_child_weight': 3,
 'max_depth': 20,
 'learning_rate': 0.1,
 'gamma': 0.1}
```

Code Sample 15 - Best parameter grid XGB

```
# Create an instance of the RandomForestClassifier
xgb = XGBClassifier()

# Grid search
xgb_grid = RandomizedSearchCV(estimator=xgb,
                      param_distributions = param_dict,
                      n_jobs=-1, n_iter=5, cv = 3,
                      verbose=2, scoring='roc_auc')
# fitting model
xgb_grid.fit(x_train,y_train)
```

Code Sample 16 – Hyper parameter Tuning XGB

Performance Metrices

Just building is not enough, as we need to evaluate it using different metrics based on the problem we're solving. Model Validation helps us to understand how well the model is generalizing on the real-world data, the data which it has not seen during the training phase. For classification problems the evaluation metrics we've used are:

- **Precision**
- **Recall**
- **Accuracy**
- **F-1 score**
- **AUC Score**

Classification accuracy, which measures the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to turn it into a percentage.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

- AUC-ROC is the valued metric used for evaluating the performance in classification models. The AUC-ROC metric clearly helps determine and tell us about the capability of a model in distinguishing the classes. The judging criteria being - Higher the AUC, better the model.

	Classifier	Train Accuracy	Test Accuracy	Precision	recall_score	f1_score	auc_score
11	Optimal XG Boosting	1.000	0.781	0.941	0.727	0.821	0.814
0	Logistic Regression	0.769	0.812	0.824	0.824	0.824	0.812
6	Optimal Logistic Regression	0.762	0.781	0.824	0.778	0.800	0.782
4	Gradient Boosting	1.000	0.719	0.824	0.700	0.757	0.725
8	Optimal Random Forest	0.766	0.719	0.706	0.750	0.727	0.719
10	Optimal Gradient Boosting	0.972	0.719	0.765	0.722	0.743	0.718
2	Random Forest	1.000	0.688	0.824	0.667	0.737	0.697
5	XG Boosting	1.000	0.688	0.765	0.684	0.722	0.688
7	Optimal Decision Tree	0.710	0.656	0.706	0.667	0.686	0.655
9	Optimal SVM	0.671	0.625	0.529	0.692	0.600	0.636
3	SVM	0.601	0.500	0.235	0.571	0.333	0.526
1	Decision Tree	1.000	0.500	0.529	0.529	0.529	0.498

Table4 – Algorithm Performance

Conclusion

- From all baseline model, Random forest classifier shows highest test accuracy, F1 score and AUC.
- Baseline model of decision tree shows huge difference in train and test accuracy which shows overfitting
- After cross validation and hyperparameter tunning, random forest shows highest test accuracy score of 96.1% and AUC is 0.969.
- Cross validation and hyperparameter tunning certainly reduces chances of overfitting and also increases performance of model.