Liver Disease Prediction

Content

This data set contains 416 liver patient records and 167 non liver patient records collected from North East of Andhra Pradesh, India. The "Dataset" column is a class label used to divide groups into liver patient (liver disease) or not (no disease). This data set contains 441 male patient records and 142 female patient records.

Any patient whose age exceeded 89 is listed as being of age "90".

Columns:

Age of the patient

Gender of the patient

Total Bilirubin

Direct Bilirubin

Alkaline Phosphotase

Alamine Aminotransferase

Aspartate Aminotransferase

Total Protiens

Albumin

Albumin and Globulin Ratio

Dataset: field used to split the data into two sets (patient with liver disease, or no disease)

Importing Libraries

```
# Importing Libraries:

import pandas as pd

import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt

# for displaying all feature from dataset:

pd.pandas.set_option('display.max_columns', None)
```

Read The DataSet

```
# Reading Dataset:

dataset = pd.read_csv("https://raw.githubusercontent.com/IBM-EPBL/IBM-Project-7380-1658854276/main/project/Dataset/indian_liver_patient.csv")

# Top 5 records:

dataset.head()
```

```
Age     Gender Total_Bilirubin  Direct_Bilirubin Alkaline_Phosphotase   Alamine_Aminotransferase
        Aspartate_Aminotransferase      Total_Protiens  Albumin         Albumin_and_Globulin_Ratio
        Dataset

0       65      Female 0.7      0.1     187     16      18      6.8     3.3     0.90    1

1       62      Male    10.9    5.5     699     64      100     7.5     3.2     0.74    1

2       62      Male    7.3     4.1     490     60      68      7.0     3.3     0.89    1

3       58      Male    1.0     0.4     182     14      20      6.8     3.4     1.00    1

4       72      Male    3.9     2.0     195     27      59      7.3     2.4     0.40    1
```

```
# Last 5 records:

dataset.tail()
```

```
Age     Gender Total_Bilirubin  Direct_Bilirubin Alkaline_Phosphotase   Alamine_Aminotransferase
        Aspartate_Aminotransferase      Total_Protiens  Albumin         Albumin_and_Globulin_Ratio
        Dataset

578     60      Male    0.5     0.1     500     20      34      5.9     1.6     0.37    2

579     40      Male    0.6     0.1     98      35      31      6.0     3.2     1.10    1

580     52      Male    0.8     0.2     245     48      49      6.4     3.2     1.00    1
```

```
581     31      Male    1.3     0.5     184     29      32      6.8     3.4     1.00    1
582     38      Male    1.0     0.3     216     21      24      7.3     4.4     1.50    2
```

```
# Shape of dataset:
dataset.shape
```

```
(583, 11)
```

Cheaking Missing (NaN) Values

```
# Cheaking Missing (NaN) Values:
dataset.isnull().sum()
```

```
Age                           0
Gender                        0
Total_Bilirubin               0
Direct_Bilirubin              0
Alkaline_Phosphotase          0
Alamine_Aminotransferase      0
Aspartate_Aminotransferase    0
Total_Protiens                0
Albumin                       0
Albumin_and_Globulin_Ratio    4
Dataset                       0
dtype: int64
```

'Albumin_and_Globulin_Ratio' feature contain 4 NaN values.

```
# Mean & Median of "Albumin_and_Globulin_Ratio" feature:
print(dataset['Albumin_and_Globulin_Ratio'].median())
print(dataset['Albumin_and_Globulin_Ratio'].mean())
```

```
0.93

0.9470639032815197

# Filling NaN Values of "Albumin_and_Globulin_Ratio" feature with Median :

dataset['Albumin_and_Globulin_Ratio'] =
dataset['Albumin_and_Globulin_Ratio'].fillna(dataset['Albumin_and_Globulin_Ratio'].median())

# Datatypes:

dataset.dtypes

Age                          int64
Gender                       object
Total_Bilirubin              float64
Direct_Bilirubin             float64
Alkaline_Phosphotase         int64
Alamine_Aminotransferase     int64
Aspartate_Aminotransferase   int64
Total_Protiens               float64
Albumin                      float64
Albumin_and_Globulin_Ratio   float64
Dataset                      int64
dtype: object

# Description:

dataset.describe()

Age     Total_Bilirubin   Direct_Bilirubin Alkaline_Phosphotase   Alamine_Aminotransferase
        Aspartate_Aminotransferase      Total_Protiens  Albumin          Albumin_and_Globulin_Ratio
        Dataset

count   583.000000      583.000000      583.000000      583.000000      583.000000      583.000000
        583.000000      583.000000      583.000000      583.000000
```

```
mean    44.746141    3.298799    1.486106    290.576329    80.713551    109.910806
        6.483190     3.141852    0.946947    1.286449

std     16.189833    6.209522    2.808498    242.937989    182.620356   288.918529
        1.085451     0.795519    0.318495    0.452490

min     4.000000     0.400000    0.100000    63.000000     10.000000    10.000000
        2.700000     0.900000    0.300000    1.000000

25%     33.000000    0.800000    0.200000    175.500000    23.000000    25.000000
        5.800000     2.600000    0.700000    1.000000

50%     45.000000    1.000000    0.300000    208.000000    35.000000    42.000000
        6.600000     3.100000    0.930000    1.000000

75%     58.000000    2.600000    1.300000    298.000000    60.500000    87.000000
        7.200000     3.800000    1.100000    2.000000

max     90.000000    75.000000   19.700000   2110.000000   2000.000000  4929.000000
        9.600000     5.500000    2.800000    2.000000
```

Data Visualization

```
# Target feature:
print("Liver Disease Patients      :", dataset['Dataset'].value_counts()[1])
print("Non Liver Disease Patients  :", dataset['Dataset'].value_counts()[2])


# Visualization:
sns.countplot(dataset['Dataset'])
plt.show()
```

```
Liver Disease Patients      : 416
Non Liver Disease Patients  : 167
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

```
# Histrogram of Age:

plt.figure(figsize=(8,5))

sns.histplot(dataset['Age'], kde=True)

plt.title('Age', fontsize=20)

plt.show()


dataset.head()
```

```
Age     Gender Total_Bilirubin   Direct_Bilirubin Alkaline_Phosphotase   Alamine_Aminotransferase
        Aspartate_Aminotransferase     Total_Protiens  Albumin           Albumin_and_Globulin_Ratio
        Dataset

0       65      Female 0.7     0.1     187     16      18      6.8     3.3     0.90    1

1       62      Male    10.9    5.5     699     64      100     7.5     3.2     0.74    1

2       62      Male    7.3     4.1     490     60      68      7.0     3.3     0.89    1

3       58      Male    1.0     0.4     182     14      20      6.8     3.4     1.00    1

4       72      Male    3.9     2.0     195     27      59      7.3     2.4     0.40    1
```

```
# Gender feature:

print("Total Male   :", dataset['Gender'].value_counts()[0])

print("Total Female :", dataset['Gender'].value_counts()[1])


# Visualization:

sns.countplot(dataset['Gender'])

plt.show()

Total Male   : 441

Total Female : 142
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

  FutureWarning
```

```
# Printing How many Unique values present in each feature:

for feature in dataset.columns:

   print(feature,":", len(dataset[feature].unique()))
```

```
Age : 72

Gender : 2

Total_Bilirubin : 113

Direct_Bilirubin : 80

Alkaline_Phosphotase : 263

Alamine_Aminotransferase : 152

Aspartate_Aminotransferase : 177

Total_Protiens : 58

Albumin : 40

Albumin_and_Globulin_Ratio : 69

Dataset : 2
```

```
# Label Encoding

dataset['Gender'] = np.where(dataset['Gender']=='Male', 1,0)

dataset.head()
```

```
Age     Gender Total_Bilirubin  Direct_Bilirubin Alkaline_Phosphotase   Alamine_Aminotransferase
        Aspartate_Aminotransferase    Total_Protiens  Albumin           Albumin_and_Globulin_Ratio
        Dataset

0       65      0       0.7     0.1     187     16      18      6.8     3.3     0.90    1

1       62      1       10.9    5.5     699     64      100     7.5     3.2     0.74    1
```

```
2    62    1    7.3    4.1    490    60    68    7.0    3.3    0.89    1

3    58    1    1.0    0.4    182    14    20    6.8    3.4    1.00    1

4    72    1    3.9    2.0    195    27    59    7.3    2.4    0.40    1
```

```
# Correlation using Heatmap:

plt.figure(figsize=(12,8))

sns.heatmap(dataset.corr(), annot=True, cmap='YlGnBu')

plt.show()
```

There is Multi-Collinearity found on our dataset.

```
dataset.columns

Index(['Age', 'Gender', 'Total_Bilirubin', 'Direct_Bilirubin',

       'Alkaline_Phosphotase', 'Alamine_Aminotransferase',

       'Aspartate_Aminotransferase', 'Total_Protiens', 'Albumin',

       'Albumin_and_Globulin_Ratio', 'Dataset'],

      dtype='object')
```

Multicollinearity betwwen 'Total_Bilirubin' and 'Direct_Bilirubin' is 0.87%

Multicollinearity betwwen 'Alamine_Aminotransferase' and 'Aspartate_Aminotransferase' is 0.79%

Multicollinearity betwwen 'Total_Protiens' and 'Albumin' is 0.78%

Multicollinearity betwwen 'Albumin' and 'Albumin_and_Globulin_Ratio' is 0.69%

Usually we drop that feature which has above 0.85% multicollinearity between two independent feature. Here we have only 'Total_Bilirubin' and 'Direct_Bilirubin' feature which has 0.87% mutlicollinearity. So we drop one of the feature from them and other independent feature has less multicollinearity, less than 0.80% So we keep that feature.

```
# Droping 'Direct_Bilirubin' feature:

dataset = dataset.drop('Direct_Bilirubin', axis=1)

dataset.columns
```

```
Index(['Age', 'Gender', 'Total_Bilirubin', 'Alkaline_Phosphotase',
       'Alamine_Aminotransferase', 'Aspartate_Aminotransferase',
       'Total_Protiens', 'Albumin', 'Albumin_and_Globulin_Ratio', 'Dataset'],
      dtype='object')
```

```
sns.distplot(dataset['Albumin'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
# Calculate the boundaries of Total_Protiens feature which differentiates the outliers:
uppper_boundary=dataset['Total_Protiens'].mean() + 3* dataset['Total_Protiens'].std()
lower_boundary=dataset['Total_Protiens'].mean() - 3* dataset['Total_Protiens'].std()

print(dataset['Total_Protiens'].mean())
print(lower_boundary)
print(uppper_boundary)
```

```
6.483190394511149
3.2268359424407516
9.739544846581545
```

```
##### Calculate the boundaries of Albumin feature which differentiates the outliers:
uppper_boundary=dataset['Albumin'].mean() + 3* dataset['Albumin'].std()
lower_boundary=dataset['Albumin'].mean() - 3* dataset['Albumin'].std()

print(dataset['Albumin'].mean())
print(lower_boundary)
print(uppper_boundary)

3.141852487135506

0.7552960692434296

5.528408905027582

# Lets compute the Interquantile range of Total_Bilirubin feature to calculate the boundaries:

IQR = dataset.Total_Bilirubin.quantile(0.75)-dataset.Total_Bilirubin.quantile(0.25)


# Extreme outliers

lower_bridge = dataset['Total_Bilirubin'].quantile(0.25) - (IQR*3)

upper_bridge = dataset['Total_Bilirubin'].quantile(0.75) + (IQR*3)


print(lower_bridge)

print(upper_bridge)


# if value greater than upper bridge, we replace that value with upper_bridge value:

dataset.loc[dataset['Total_Bilirubin'] >= upper_bridge, 'Total_Bilirubin'] = upper_bridge

-4.6000000000000005

8.0

# Lets compute the Interquantile range of Alkaline_Phosphotase feature to calculate the boundaries:

IQR = dataset.Alkaline_Phosphotase.quantile(0.75) - dataset.Alkaline_Phosphotase.quantile(0.25)


# Extreme outliers

lower_bridge = dataset['Alkaline_Phosphotase'].quantile(0.25) - (IQR*3)

upper_bridge = dataset['Alkaline_Phosphotase'].quantile(0.75) + (IQR*3)
```

```
print(lower_bridge)

print(upper_bridge)


# if value greater than upper bridge, we replace that value with upper_bridge value:

dataset.loc[dataset['Alkaline_Phosphotase'] >= upper_bridge, 'Alkaline_Phosphotase'] = upper_bridge
```

```
-192.0

665.5
```

```
# Lets compute the Interquantile range of Alamine_Aminotransferase feature to calculate the boundaries:

IQR = dataset.Alamine_Aminotransferase.quantile(0.75) - dataset.Alamine_Aminotransferase.quantile(0.25)


# Extreme outliers

lower_bridge = dataset['Alamine_Aminotransferase'].quantile(0.25) - (IQR*3)

upper_bridge = dataset['Alamine_Aminotransferase'].quantile(0.75) + (IQR*3)


print(lower_bridge)

print(upper_bridge)


# if value greater than upper bridge, we replace that value with upper_bridge value:

dataset.loc[dataset['Alamine_Aminotransferase'] >= upper_bridge, 'Alamine_Aminotransferase'] = upper_bridge
```

```
-89.5

173.0
```

```
# Lets compute the Interquantile range of Aspartate_Aminotransferase feature to calculate the boundaries:
```

```
IQR = dataset.Aspartate_Aminotransferase.quantile(0.75) -
dataset.Aspartate_Aminotransferase.quantile(0.25)


# Extreme outliers

lower_bridge = dataset['Aspartate_Aminotransferase'].quantile(0.25) - (IQR*3)

upper_bridge = dataset['Aspartate_Aminotransferase'].quantile(0.75) + (IQR*3)


print(lower_bridge)

print(upper_bridge)


# if value greater than upper bridge, we replace that value with upper_bridge value:

dataset.loc[dataset['Aspartate_Aminotransferase'] >= upper_bridge, 'Aspartate_Aminotransferase'] =
upper_bridge
```

```
-161.0

273.0
```

```
# Lets compute the Interquantile range of Albumin_and_Globulin_Ratio feature to calculate the
boundaries

IQR = dataset.Albumin_and_Globulin_Ratio.quantile(0.75) -
dataset.Albumin_and_Globulin_Ratio.quantile(0.25)


# Extreme outliers

lower_bridge = dataset['Albumin_and_Globulin_Ratio'].quantile(0.25) - (IQR*3)

upper_bridge = dataset['Albumin_and_Globulin_Ratio'].quantile(0.75) + (IQR*3)


print(lower_bridge)

print(upper_bridge)
```

# if value greater than upper bridge, we replace that value with upper_bridge value:

```
dataset.loc[dataset['Albumin_and_Globulin_Ratio'] >= upper_bridge, 'Albumin_and_Globulin_Ratio'] = 
upper_bridge
```

```
-0.5000000000000004

2.3000000000000007
```

# Top 5 records:

```
dataset.head()
```

```
Age     Gender Total_Bilirubin  Alkaline_Phosphotase   Alamine_Aminotransferase
        Aspartate_Aminotransferase    Total_Protiens  Albumin         Albumin_and_Globulin_Ratio
        Dataset

0       65      0       0.7     187.0   16      18      6.8     3.3     0.90    1

1       62      1       8.0     665.5   64      100     7.5     3.2     0.74    1

2       62      1       7.3     490.0   60      68      7.0     3.3     0.89    1

3       58      1       1.0     182.0   14      20      6.8     3.4     1.00    1

4       72      1       3.9     195.0   27      59      7.3     2.4     0.40    1
```

# Description after deal with outliers by IQR:

```
dataset.describe()
```

```
Age     Gender Total_Bilirubin  Alkaline_Phosphotase   Alamine_Aminotransferase
        Aspartate_Aminotransferase    Total_Protiens  Albumin         Albumin_and_Globulin_Ratio
        Dataset

count   583.000000      583.000000      583.000000      583.000000      583.000000      583.000000
        583.000000      583.000000      583.000000      583.000000

mean    44.746141       0.756432        2.249400        266.389365      53.399657       73.041166
        6.483190        3.141852        0.945403        1.286449

std     16.189833       0.429603        2.382344        145.665460      46.059536       73.549864
        1.085451        0.795519        0.310942        0.452490

min     4.000000        0.000000        0.400000        63.000000       10.000000       10.000000
        2.700000        0.900000        0.300000        1.000000
```

```
25%     33.000000       1.000000        0.800000        175.500000      23.000000       25.000000
        5.800000        2.600000        0.700000        1.000000

50%     45.000000       1.000000        1.000000        208.000000      35.000000       42.000000
        6.600000        3.100000        0.930000        1.000000

75%     58.000000       1.000000        2.600000        298.000000      60.500000       87.000000
        7.200000        3.800000        1.100000        2.000000

max     90.000000       1.000000        8.000000        665.500000      173.000000      273.000000
        9.600000        5.500000        2.300000        2.000000
```

Independent and Dependent Split

```
# Independent and Dependent Feature:

X = dataset.iloc[:, :-1]

y = dataset.iloc[:, -1]

# top 5 records of Independent features:

X.head()
```

```
Age     Gender Total_Bilirubin  Alkaline_Phosphotase    Alamine_Aminotransferase
        Aspartate_Aminotransferase      Total_Protiens  Albumin         Albumin_and_Globulin_Ratio

0       65      0       0.7     187.0   16      18      6.8     3.3     0.90

1       62      1       8.0     665.5   64      100     7.5     3.2     0.74

2       62      1       7.3     490.0   60      68      7.0     3.3     0.89

3       58      1       1.0     182.0   14      20      6.8     3.4     1.00

4       72      1       3.9     195.0   27      59      7.3     2.4     0.40
```

```
# top 5 records of dependent features:

y.head()
```

```
0   1

1   1

2   1
```

```
3    1
4    1
Name: Dataset, dtype: int64
```

```
# SMOTE Technique:
from imblearn.combine import SMOTETomek
smote = SMOTETomek()
X_smote, y_smote = smote.fit_resample(X,y)
# Counting before and after SMOTE:
from collections import Counter
print('Before SMOTE : ', Counter(y))
print('After SMOTE  : ', Counter(y_smote))
```

```
Before SMOTE :  Counter({1: 416, 2: 167})
After SMOTE  :  Counter({1: 396, 2: 396})
```

Train Test Split

```
# Train Test Split:
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_smote,y_smote, test_size=0.3, random_state=33)
print(X_train.shape)
print(X_test.shape)
```

```
(554, 9)
(238, 9)
```

```
# Feature Importance :
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

```
### Apply SelectKBest Algorithm

ordered_rank_features=SelectKBest(score_func=chi2,k=9)

ordered_feature=ordered_rank_features.fit(X,y)


dfscores=pd.DataFrame(ordered_feature.scores_,columns=["Score"])

dfcolumns=pd.DataFrame(X.columns)


features_rank=pd.concat([dfcolumns,dfscores],axis=1)


features_rank.columns=['Features','Score']

features_rank.nlargest(9, 'Score')
```

```
Features        Score

5       Aspartate_Aminotransferase      3368.743077

3       Alkaline_Phosphotase    2385.790640

4       Alamine_Aminotransferase        1717.348297

2       Total_Bilirubin   127.476411

0       Age     64.315174

7       Albumin         3.053371

8       Albumin_and_Globulin_Ratio    1.704602

1       Gender 0.964518

6       Total_Protiens   0.129627
```

Classification Algorithm And Run model

```
# Importing Performance Metrics:
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# RandomForestClassifier:

from sklearn.ensemble import RandomForestClassifier

RandomForest = RandomForestClassifier()

RandomForest = RandomForest.fit(X_train,y_train)


# Predictions:

y_pred = RandomForest.predict(X_test)


# Performance:

print('Accuracy:', accuracy_score(y_test,y_pred))

print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))

Accuracy: 0.8109243697478992

[[ 87  35]

 [ 10 106]]

              precision    recall  f1-score   support



           1       0.90      0.71      0.79       122

           2       0.75      0.91      0.82       116



    accuracy                           0.81       238

   macro avg       0.82      0.81      0.81       238

weighted avg       0.83      0.81      0.81       238
```

```
# AdaBoostClassifier:

from sklearn.ensemble import AdaBoostClassifier

AdaBoost = AdaBoostClassifier()

AdaBoost = AdaBoost.fit(X_train,y_train)


# Predictions:

y_pred = AdaBoost.predict(X_test)


# Performance:

print('Accuracy:', accuracy_score(y_test,y_pred))

print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))

Accuracy: 0.7857142857142857

[[ 86  36]

 [ 15 101]]

              precision    recall  f1-score   support


           1       0.85      0.70      0.77       122

           2       0.74      0.87      0.80       116


    accuracy                           0.79       238

   macro avg       0.79      0.79      0.78       238

weighted avg       0.80      0.79      0.78       238


# GradientBoostingClassifier:
```

```
from sklearn.ensemble import GradientBoostingClassifier

GradientBoost = GradientBoostingClassifier()

GradientBoost = GradientBoost.fit(X_train,y_train)


# Predictions:

y_pred = GradientBoost.predict(X_test)


# Performance:

print('Accuracy:', accuracy_score(y_test,y_pred))

print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))

Accuracy: 0.7605042016806722

[[89 33]

 [24 92]]

              precision    recall  f1-score   support



           1       0.79      0.73      0.76       122

           2       0.74      0.79      0.76       116



    accuracy                           0.76       238

   macro avg       0.76      0.76      0.76       238

weighted avg       0.76      0.76      0.76       238
```

RandomizedSearchCV

```
# Importing RandomizedSearchCV:
```

```
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest:

n_estimators = [int(x) for x in np.linspace(start = 100, stop = 2000, num = 20)]


# Number of features to consider at every split:

max_features = ['auto', 'sqrt','log2']


# Maximum number of levels in tree:

max_depth = [int(x) for x in np.linspace(100, 100,20)]


# Minimum number of samples required to split a node:

min_samples_split = [1,2,3,4,5,6,7,8,9,10,12,14,16,18,20]


# Minimum number of samples required at each leaf node:

min_samples_leaf = [1,2,3,4,5,6,7,8,9,10,12,14,16,18,20]

# Create the random grid:

random_grid = {'n_estimators': n_estimators,

              'max_features': max_features,

              'max_depth': max_depth,

              'min_samples_split': min_samples_split,

              'min_samples_leaf': min_samples_leaf,

             'criterion':['entropy','gini']}

print(random_grid)

{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600,
1700, 1800, 1900, 2000], 'max_features': ['auto', 'sqrt', 'log2'], 'max_depth': [100, 100, 100, 100, 100,
100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100], 'min_samples_split': [1, 2, 3,
4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20], 'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20],
'criterion': ['entropy', 'gini']}

rf = RandomForestClassifier()

rf_randomcv = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100,
cv = 5, verbose = 2,

                              random_state = 0, n_jobs = -1)


# fit the randomized model:

rf_randomcv.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:

40 fits failed out of a total of 500.

The score on these train-test partitions for these parameters will be set to nan.

If these failures are not expected, you can try to debug them by setting error_score='raise'.


Below are more details about the failures:

--------------------------------------------------------------------------------

40 fits failed with the following error:

Traceback (most recent call last):

  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py", line 680, in
_fit_and_score

    estimator.fit(X_train, y_train, **fit_params)

  File "/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py", line 467, in fit

    for i, t in enumerate(trees)

  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 1085, in __call__

    if self.dispatch_one_batch(iterator):

  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 901, in dispatch_one_batch
```

```
    self._dispatch(tasks)

  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 819, in _dispatch

    job = self._backend.apply_async(batch, callback=cb)

  File "/usr/local/lib/python3.7/dist-packages/joblib/_parallel_backends.py", line 208, in apply_async

    result = ImmediateResult(func)

  File "/usr/local/lib/python3.7/dist-packages/joblib/_parallel_backends.py", line 597, in __init__

    self.results = batch()

  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 289, in __call__

    for func, args, kwargs in self.items]

  File "/usr/local/lib/python3.7/dist-packages/joblib/parallel.py", line 289, in

    for func, args, kwargs in self.items]

  File "/usr/local/lib/python3.7/dist-packages/sklearn/utils/fixes.py", line 216, in __call__

    return self.function(*args, **kwargs)

  File "/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_forest.py", line 185, in
_parallel_build_trees

    tree.fit(X, y, sample_weight=curr_sample_weight, check_input=False)

  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", line 942, in fit

    X_idx_sorted=X_idx_sorted,

  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", line 254, in fit

    % self.min_samples_split

ValueError: min_samples_split must be an integer greater than 1 or a float in (0.0, 1.0]; got the integer 1



  warnings.warn(some_fits_failed_message, FitFailedWarning)

/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:972: UserWarning: One or
more of the test scores are non-finite: [       nan 0.6950041  0.74733825 0.68958231 0.68599509
0.71847666

 0.73829648 0.7040131  0.73290745 0.72029484 0.69320229 0.72209664
 0.6950041  0.69141687 0.6986077  0.73113841 0.68781327 0.74733825
 0.72211302 0.75094185        nan 0.71665848 0.74194922 0.71667486
 0.70044226 0.69678952 0.72026208 0.71665848 0.74552007 0.71846028
 0.69138411        nan 0.72208026 0.72568387 0.69316953 0.72568387
 0.72027846 0.75099099 0.74552007 0.73467649 0.71665848 0.69140049
 0.69682228 0.70042588 0.68782965 0.73112203        nan 0.70584767
        nan 0.72748567 0.71667486 0.6968059  0.70763309 0.72750205
 0.71847666 0.74735463 0.75276003 0.73290745 0.6968059  0.72748567
 0.71485667 0.71841114 0.68781327 0.72031122 0.75276003 0.6968059
 0.73289107 0.72208026 0.73108927 0.7004095        nan 0.73829648
 0.7040131  0.72751843 0.73651106 0.72751843 0.72568387 0.72209664
 0.70583129 0.68963145 0.73651106 0.69502048 0.72209664        nan
 0.68959869 0.74915643 0.72930385 0.74917281 0.73108927 0.74191646
 0.74191646 0.74373464        nan 0.73290745 0.6968059  0.68959869
 0.71667486 0.69498771 0.73832924 0.71305487]
  category=UserWarning,
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'criterion': ['entropy', 'gini'],
                                        'max_depth': [100, 100, 100, 100, 100,
                                                      100, 100, 100, 100, 100,
                                                      100, 100, 100, 100, 100,
                                                      100, 100, 100, 100, 100],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                              'min_samples_leaf': [1, 2, 3, 4, 5, 6,
                                                   7, 8, 9, 10, 12,
                                                   14, 16, 18, 20],
                              'min_samples_split': [1, 2, 3, 4, 5, 6,
                                                    7, 8, 9, 10, 12,
                                                    14, 16, 18, 20],
                              'n_estimators': [100, 200, 300, 400,
                                               500, 600, 700, 800,
                                               900, 1000, 1100, 1200,
                                               1300, 1400, 1500, 1600,
                                               1700, 1800, 1900,
                                               2000]},
                   random_state=0, verbose=2)
# Best parameter of RandomizedSearchCV:
rf_randomcv.best_params_
{'n_estimators': 2000,
 'min_samples_split': 3,
 'min_samples_leaf': 2,
 'max_features': 'log2',
 'max_depth': 100,
 'criterion': 'entropy'}
# Creating model using best parameter of RandomizedSearchCV:
RandomForest_RandomCV = RandomForestClassifier(criterion = 'entropy', n_estimators = 2000, max_depth = 100, max_features = 'log2',
                                 min_samples_split = 3, min_samples_leaf = 2)
RandomForest_RandomCV = RandomForest_RandomCV.fit(X_train,y_train)
```

```
# Predictions:

y_pred = RandomForest_RandomCV.predict(X_test)


# Performance:

print('Accuracy:', accuracy_score(y_test,y_pred))

print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))

Accuracy: 0.8067226890756303

[[ 86  36]

 [ 10 106]]

              precision    recall  f1-score   support



           1       0.90      0.70      0.79       122

           2       0.75      0.91      0.82       116



    accuracy                           0.81       238

   macro avg       0.82      0.81      0.81       238

weighted avg       0.82      0.81      0.80       238
```

GridSearchCV

```
# Importing GridSearchCV:

from sklearn.model_selection import GridSearchCV

# Best parameter:

rf_randomcv.best_params_
```

```
{'n_estimators': 2000,
 'min_samples_split': 3,
 'min_samples_leaf': 2,
 'max_features': 'log2',
 'max_depth': 100,
 'criterion': 'entropy'}
```

```
param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'max_depth': [rf_randomcv.best_params_['max_depth']-50,
                  rf_randomcv.best_params_['max_depth'],
                 rf_randomcv.best_params_['max_depth']+50],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf']-1,
                         rf_randomcv.best_params_['min_samples_leaf'],
                         rf_randomcv.best_params_['min_samples_leaf']+1],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split'] - 1,
                          rf_randomcv.best_params_['min_samples_split'],
                          rf_randomcv.best_params_['min_samples_split'] +1],
    'n_estimators': [rf_randomcv.best_params_['n_estimators'] - 50,
                     rf_randomcv.best_params_['n_estimators'],
                     rf_randomcv.best_params_['n_estimators'] + 50]
}


print(param_grid)
```

```
{'criterion': ['entropy'], 'max_features': ['log2'], 'max_depth': [50, 100, 150], 'min_samples_leaf': [1, 2, 3],
'min_samples_split': [2, 3, 4], 'n_estimators': [1950, 2000, 2050]}
```

```
# Fit the grid_search to the data:

rf = RandomForestClassifier()

grid_search = GridSearchCV(estimator = rf, param_grid = param_grid, cv=5 , n_jobs = -1, verbose = 2)

grid_search.fit(X_train,y_train)

Fitting 5 folds for each of 81 candidates, totalling 405 fits

GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,

             param_grid={'criterion': ['entropy'], 'max_depth': [50, 100, 150],

                         'max_features': ['log2'],

                         'min_samples_leaf': [1, 2, 3],

                         'min_samples_split': [2, 3, 4],

                         'n_estimators': [1950, 2000, 2050]},

             verbose=2)

# Best Parameter of GridSearchCV:

grid_search.best_params_

{'criterion': 'entropy',

 'max_depth': 50,

 'max_features': 'log2',

 'min_samples_leaf': 2,

 'min_samples_split': 2,

 'n_estimators': 2050}

# Creating model using best parameter of GridSearchCV:

RandomForest_gridCV = RandomForestClassifier(criterion='entropy', n_estimators=1950,
max_depth=150, max_features='log2',

                                  min_samples_split=2, min_samples_leaf=1)

RandomForest_gridCv = RandomForest_gridCV.fit(X_train,y_train)
```

```
# Predictions:

y_pred = RandomForest_gridCV.predict(X_test)


# Performance:

print('Accuracy:', accuracy_score(y_test,y_pred))

print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))

Accuracy: 0.8025210084033614

[[ 85  37]

 [ 10 106]]

              precision    recall  f1-score   support



           1       0.89      0.70      0.78       122

           2       0.74      0.91      0.82       116



    accuracy                           0.80       238

   macro avg       0.82      0.81      0.80       238

weighted avg       0.82      0.80      0.80       238
```

Save Model

```
# Creating a pickle file for the classifier

import pickle

filename = 'Liver.pkl'

pickle.dump(RandomForestClassifier, open(filename, 'wb'))
```