Fetal Health Classification Using Different Classifier ML models

#Logistic Regression

```
import pandas as pd
import numpy as np
import matplotlib
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
df = pd.read_csv("/content/drive/MyDrive/TIP S.Y.'s/TIP 2021-2022 (3rd
yr, 1st sem)/Predictive Analytics w  Machine Learning/CPE 312 -
Machine Learning/data/fetal_health.csv")
```

```
df.head()
```

```
   baseline value  accelerations  ...  histogram_tendency
fetal_health
0           120.0          0.000  ...                 1.0
2.0
1           132.0          0.006  ...                 0.0
1.0
2           133.0          0.003  ...                 0.0
1.0
3           134.0          0.003  ...                 1.0
1.0
4           132.0          0.007  ...                 1.0
1.0

[5 rows x 22 columns]
```

```
df.info()
# No missing values and all are numerical
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2126 entries, 0 to 2125
Data columns (total 22 columns):
 #   Column                                                  Non-Null
Count  Dtype
---  ------
--------------  -----
 0   baseline value                                          2126 non-
```

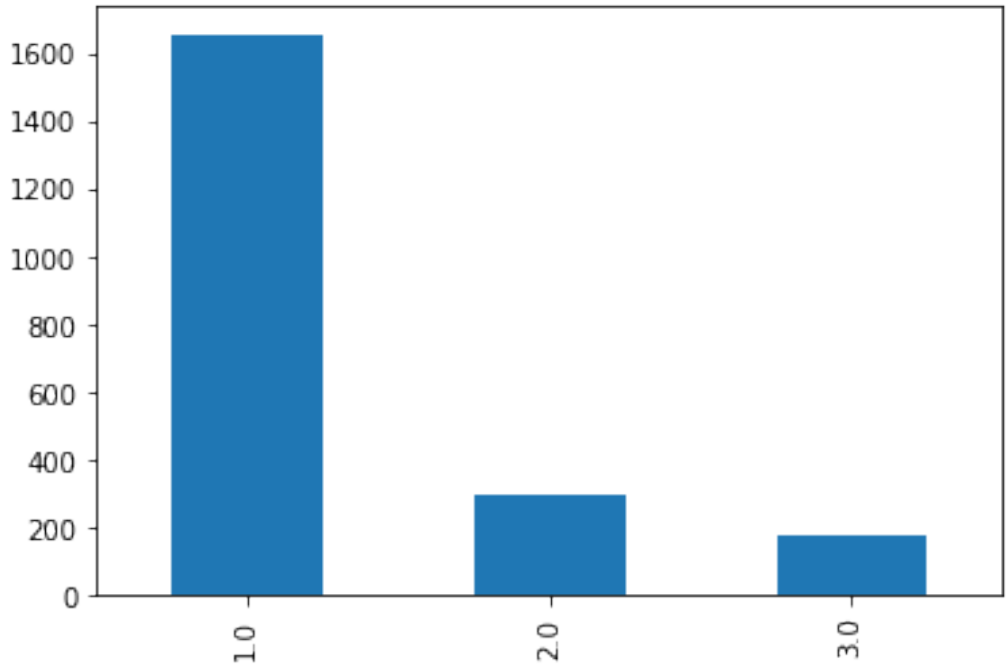
```
null   float64
 1   accelerations                                           2126 non-
null   float64
 2   fetal_movement                                          2126 non-
null   float64
 3   uterine_contractions                                    2126 non-
null   float64
 4   light_decelerations                                     2126 non-
null   float64
 5   severe_decelerations                                    2126 non-
null   float64
 6   prolongued_decelerations                                2126 non-
null   float64
 7   abnormal_short_term_variability                         2126 non-
null   float64
 8   mean_value_of_short_term_variability                    2126 non-
null   float64
 9   percentage_of_time_with_abnormal_long_term_variability  2126 non-
null   float64
 10  mean_value_of_long_term_variability                     2126 non-
null   float64
 11  histogram_width                                         2126 non-
null   float64
 12  histogram_min                                           2126 non-
null   float64
 13  histogram_max                                           2126 non-
null   float64
 14  histogram_number_of_peaks                               2126 non-
null   float64
 15  histogram_number_of_zeroes                              2126 non-
null   float64
 16  histogram_mode                                          2126 non-
null   float64
 17  histogram_mean                                          2126 non-
null   float64
 18  histogram_median                                        2126 non-
null   float64
 19  histogram_variance                                      2126 non-
null   float64
 20  histogram_tendency                                      2126 non-
null   float64
 21  fetal_health                                            2126 non-
null   float64
dtypes: float64(22)
memory usage: 365.5 KB
```

```
df["fetal_health"].unique()
```

```
array([2., 1., 3.])
```

```
# is there an imbalance in the dataset?
df["fetal_health"].value_counts().plot(kind = "bar")
print(df["fetal_health"].value_counts())

#it seems categorial 1 (normal) is more abundant than the other
categories. This may cause our model to
# better predict class 1
```

```
1.0    1655
2.0     295
3.0     176
Name: fetal_health, dtype: int64
```




```
# Let us check the correlation of each of the feature columns with the
target column

y_correl = (df["fetal_health"]).astype(int)
fields = list(df.columns[:-1])  # everything except fetal_health
correlations = df[fields].corrwith(y_correl)
correlations.sort_values(inplace=True)
correlations
```

```
accelerations                                   -0.364066
histogram_mode                                  -0.250412
histogram_mean                                  -0.226985
mean_value_of_long_term_variability             -0.226797
histogram_median                                -0.205033
uterine_contractions                            -0.204894
histogram_tendency                              -0.131976
mean_value_of_short_term_variability            -0.103382
```

```
histogram_width                                                -0.068789
histogram_max                                                  -0.045265
histogram_number_of_peaks                                      -0.023666
histogram_number_of_zeroes                                     -0.016682
light_decelerations                                             0.058870
histogram_min                                                   0.063175
fetal_movement                                                  0.088010
severe_decelerations                                            0.131934
baseline value                                                  0.148151
histogram_variance                                              0.206630
percentage_of_time_with_abnormal_long_term_variability          0.426146
abnormal_short_term_variability                                 0.471191
prolongued_decelerations                                        0.484859
dtype: float64
```

```
df.describe(percentiles = [.25, .5, .75, .90, .95, .99])
```

```
       baseline value  accelerations  ...  histogram_tendency
fetal_health
count     2126.000000    2126.000000  ...         2126.000000
2126.000000
mean       133.303857       0.003178  ...            0.320320
1.304327
std          9.840844       0.003866  ...            0.610829
0.614377
min        106.000000       0.000000  ...           -1.000000
1.000000
25%        126.000000       0.000000  ...            0.000000
1.000000
50%        133.000000       0.002000  ...            0.000000
1.000000
75%        140.000000       0.006000  ...            1.000000
1.000000
90%        146.000000       0.009000  ...            1.000000
2.000000
95%        149.000000       0.011000  ...            1.000000
3.000000
99%        158.000000       0.015000  ...            1.000000
3.000000
max        160.000000       0.019000  ...            1.000000
3.000000

[11 rows x 22 columns]
```

```
from sklearn.model_selection import train_test_split

X = df.drop(['fetal_health'], axis=1)
Y = df['fetal_health']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
train_size=0.7, test_size=0.3, random_state=100)

print("train_inputs shape is {}".format(X_train.shape))
print("train_target shape is {}".format(y_train.shape))
print("test_inputs shape is {}".format(X_test.shape))
print("test_target shape is {}".format(y_test.shape))
```

```
train_inputs shape is (1488, 21)
train_target shape is (1488,)
test_inputs shape is (638, 21)
test_target shape is (638,)
```

```
num_cols = X_train.columns.tolist()
len(num_cols)
```

```
21
```

```
# Importing libraries for scaling
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.fit_transform(X_test[num_cols])

X_train.describe().loc[["min","max"]]
```

```
     baseline value  accelerations  ...  histogram_variance
histogram_tendency
min             0.0            0.0  ...                 0.0
0.0
max             1.0            1.0  ...                 1.0
1.0

[2 rows x 21 columns]
```

```
X_test.describe().loc[["min","max"]]
```

```
     baseline value  accelerations  ...  histogram_variance
histogram_tendency
min             0.0            0.0  ...                 0.0
0.0
max             1.0            1.0  ...                 1.0
1.0

[2 rows x 21 columns]
```

```
y_train.sample(5)
```

```
879     1.0
394     1.0
1622    1.0
```

```
776     2.0
1614    1.0
Name: fetal_health, dtype: float64
```

Training the Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='liblinear')
model.fit(X_train, y_train)
weight_df = pd.DataFrame(
    {"feature":X_train.columns.tolist(),
    "weight": model.coef_.tolist()[0]}
)
weight_df.sort_values(ascending = False, by = "weight")
```

	feature	weight
1	accelerations	5.166611
3	uterine_contractions	2.126080
16	histogram_mode	1.951730
8	mean_value_of_short_term_variability	1.084694
18	histogram_median	0.952148
4	light_decelerations	0.840984
17	histogram_mean	0.482478
10	mean_value_of_long_term_variability	0.394768
11	histogram_width	0.300951
20	histogram_tendency	0.215238
12	histogram_min	-0.332072
15	histogram_number_of_zeroes	-0.433982
14	histogram_number_of_peaks	-0.554466
5	severe_decelerations	-0.761100
2	fetal_movement	-1.174326
13	histogram_max	-1.666505
9	percentage_of_time_with_abnormal_long_term_var...	-2.751685
19	histogram_variance	-2.793487
0	baseline value	-3.166935
7	abnormal_short_term_variability	-3.517308
6	prolongued_decelerations	-4.684603

```
import plotly.graph_objects as go

neg_weights = weight_df[weight_df["weight"] < 0].sort_values("weight",
ascending = False)
pos_weights = weight_df[weight_df["weight"] > 0].sort_values("weight",
ascending = False)

fig = go.Figure()
fig.add_trace(go.Bar(x=neg_weights.feature, y=neg_weights.weight,
                marker_color='crimson',
                name='negative weights'))
fig.add_trace(go.Bar(x=pos_weights.feature, y=pos_weights.weight,
```

```
                marker_color='lightslategrey',
                name='positive weights'
                ))
#fig.update_xaxes(visible = False)
fig.update_xaxes(nticks = 100)
fig.update_yaxes(nticks=20, ticks = "outside")
fig.update_layout(
    margin=dict(l=50, r=20, t=20, b=20),
    paper_bgcolor="LightSteelBlue"
)
fig.show()
```

generally the higher the coefficient the more the feature is able to accurately predict the target variable. A high negative coefficient value means that the relationship is inverse, but strong.

Making Predictions on the test dataset

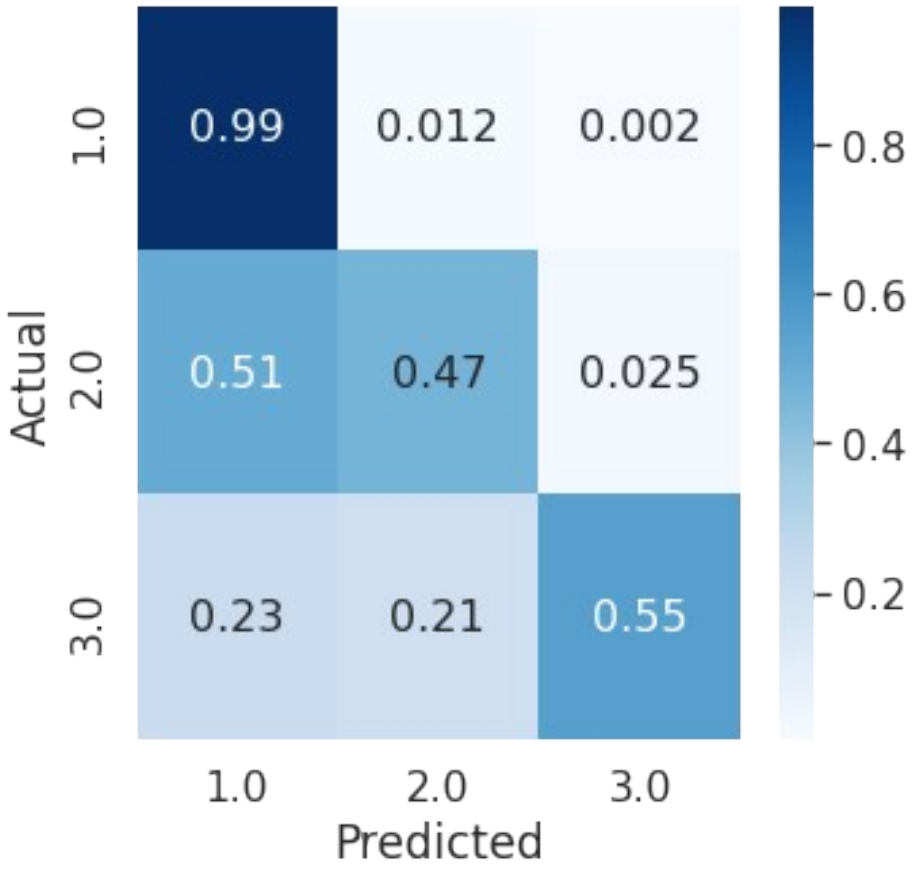
```
val_predictions = model.predict(X_test)

# check the accuracy of the train_target and train_predictions
from sklearn.metrics import accuracy_score
print("accuracy is: {}".format(accuracy_score(y_test,
val_predictions)))
```

```
accuracy is: 0.8824451410658307
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, val_predictions, normalize = 'true')
```

```
array([[0.98602794, 0.01197605, 0.00199601],
       [0.50617284, 0.4691358 , 0.02469136],
       [0.23214286, 0.21428571, 0.55357143]])
```

```
data = confusion_matrix(y_test, val_predictions, normalize = 'true')
df_cm = pd.DataFrame(data, columns=np.unique(val_predictions), index =
np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```



as one can see the confusion matrix reveals that the model is better at predicting the 1 (normal) class than the other classes. A worrisome data is that we have a high false positive for 1 when the actual data is actually of class 2.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, val_predictions))
```

```
              precision    recall  f1-score   support

         1.0       0.90      0.99      0.94       501
         2.0       0.68      0.47      0.55        81
         3.0       0.91      0.55      0.69        56

    accuracy                           0.88       638
   macro avg       0.83      0.67      0.73       638
weighted avg       0.87      0.88      0.87       638
```

hypertuning the Logistic Regression Model

```
param_grid = [
    {'penalty':['l1', 'l2','elasticnet','none'],
     'solver':['liblinear','newton-cg','lbfgs','sag','saga'],
     'max_iter': [100,1000,1500,5000],
```

```
    }
]

from sklearn.model_selection import GridSearchCV

logmodel = LogisticRegression()

clf = GridSearchCV(logmodel, param_grid, cv=5, verbose=True, n_jobs=-
1)

best_clf = clf.fit(X_train, y_train)

Fitting 5 folds for each of 80 candidates, totalling 400 fits

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


180 fits failed out of a total of 400.
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:
--------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties,
got l1 penalty.

--------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 680, in _fit_and_score
```

```
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
l1 penalty.

--------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1
penalty.

--------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 459, in _check_solver
    solver
ValueError: Only 'saga' solver supports elasticnet penalty, got
solver=liblinear.
```

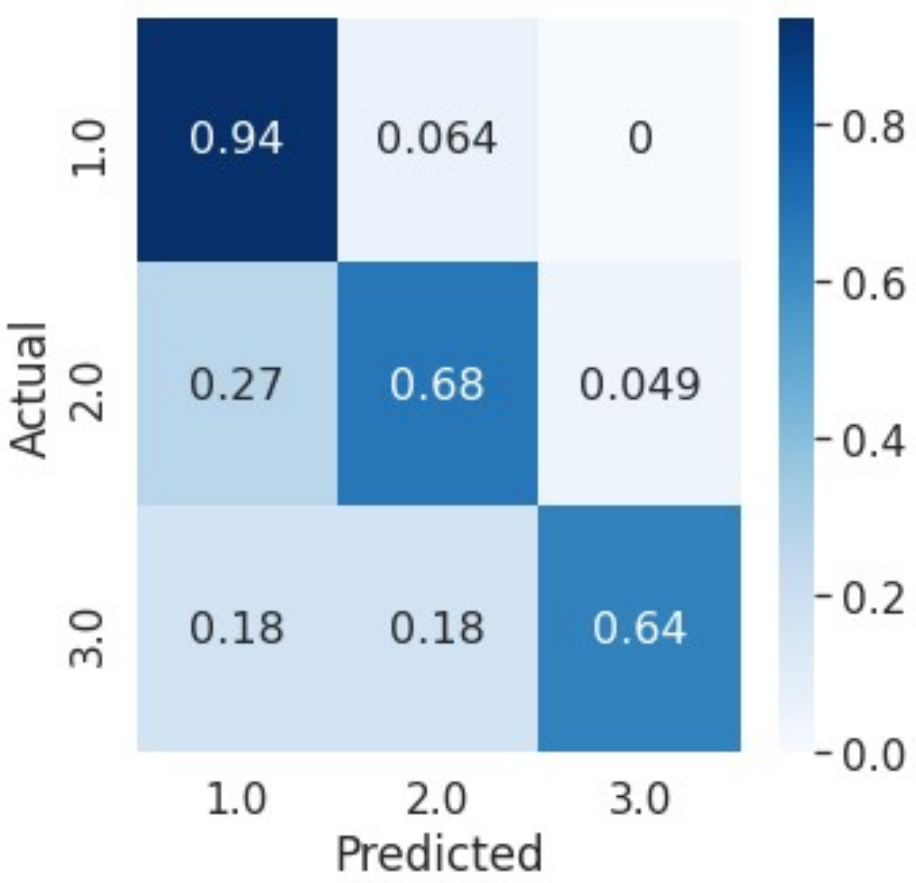
```
--------------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties,
got elasticnet penalty.

--------------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got
elasticnet penalty.

--------------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
```

```
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver sag supports only 'l2' or 'none' penalties, got
elasticnet penalty.

--------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1473, in fit
    % self.l1_ratio
ValueError: l1_ratio must be between 0 and 1; got (l1_ratio=None)

--------------------------------------------------------------------
----------
20 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_valid
ation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic
.py", line 464, in _check_solver
    raise ValueError("penalty='none' is not supported for the
liblinear solver")
ValueError: penalty='none' is not supported for the liblinear solver


/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search
.py:972: UserWarning:

One or more of the test scores are non-finite: [0.89515739        nan
nan        nan 0.90187332 0.88171649
 0.89313267 0.89313267 0.89380381 0.89380381        nan        nan
        nan        nan        nan        nan 0.89245023 0.89513705
 0.89446365 0.89378799 0.89515739        nan        nan        nan
 0.90187332 0.88171649 0.89313267 0.89313267 0.89380381 0.89380381
```

```
        nan         nan         nan         nan         nan         nan
 0.89245023 0.89245023 0.89245023 0.89177683 0.89515739         nan
        nan         nan 0.90187332 0.88171649 0.89313267 0.89313267
 0.89380381 0.89380381         nan         nan         nan         nan
        nan         nan 0.89245023 0.89245023 0.89245023 0.89177683
 0.89515739         nan         nan         nan 0.90187332 0.88171649
 0.89313267 0.89313267 0.89313267 0.89380381         nan         nan
        nan         nan         nan         nan 0.89245023 0.89245023
 0.89245023 0.89177683]

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_sag.py:35
4: ConvergenceWarning:

The max_iter was reached which means the coef_ did not converge


best_clf.best_params_

{'max_iter': 100, 'penalty': 'l1', 'solver': 'saga'}
```

Classfication Metrics of Optimized Logistic Regression Model

```
optimized_y_prediction = best_clf.predict(X_test)
print("new accuracy is: {}".format(accuracy_score(y_test,
optimized_y_prediction)))

new accuracy is: 0.877742946708464

data = confusion_matrix(y_test, optimized_y_prediction, normalize =
'true')
df_cm = pd.DataFrame(data, columns=np.unique(optimized_y_prediction),
index = np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```



Previous confusion matrix:

```
array([[0.98602794, 0.01197605, 0.00199601],
       [0.50617284, 0.4691358 , 0.02469136],
       [0.23214286, 0.21428571, 0.55357143]])
```

Through hypertuning we were able to increase our prediction power for class 2 and 3. Though class 1 decreased in accuracy/recall, the value of 0.94 is still acceptable. I would not necessarily use this model to predict accurately predict if a fetus has a suspect or pathological condition. In other words, I would not trust the model if it were to predict that the fetus is either pathological or suspect; however, I would trust the model if it predicted that the fetus is normal.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, optimized_y_prediction))
```

```
              precision    recall  f1-score   support

         1.0       0.94      0.94      0.94       501
         2.0       0.57      0.68      0.62        81
         3.0       0.90      0.64      0.75        56

    accuracy                           0.88       638
```

```
   macro avg       0.80      0.75      0.77       638
weighted avg       0.89      0.88      0.88       638
```

```
prob_a=best_clf.predict_proba(X_test)
```

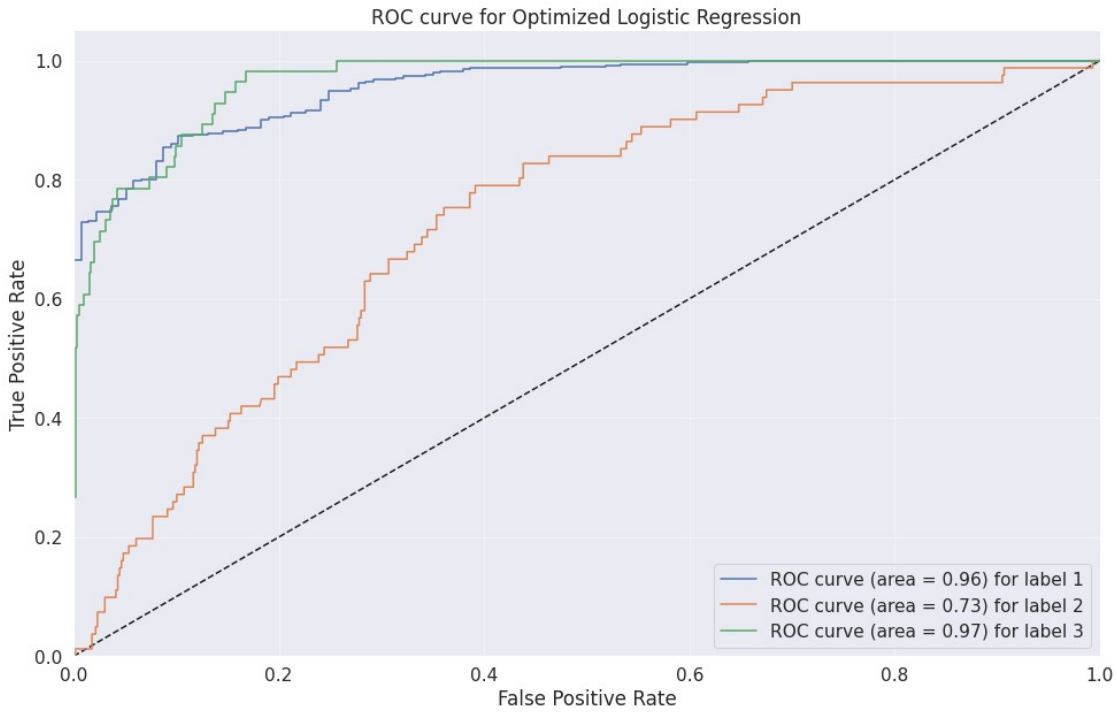
```
from sklearn.metrics import roc_curve, roc_auc_score

roc_auc_score(y_test, prob_a, multi_class='ovo', average='weighted')
```

```
0.9563605348888232
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

def plot_multiclass_roc(clf, X_test, y_test, n_classes, figsize=(17,
6)):
    try:
        y_score = clf.decision_function(X_test)
        print("Using decision_function method")
    except:
        y_score = clf.predict_proba(X_test)
        print("Using predict_proba method")

    # structures
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # calculate dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:,
i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # roc for each class
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('ROC curve for Optimized Logistic Regression')
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for
label %i' % (roc_auc[i], i+1))
    ax.legend(loc="best")
    ax.grid(alpha=.4)
```

```
    sns.despine()
    plt.show()
    return fig, ax

logmodel_roc_auc_curve = plot_multiclass_roc(best_clf, X_test, y_test,
n_classes=3, figsize=(16, 10))
```

```
Using decision_function method
```



Random Forest Model

```
# Let's check our training and test datasets again
print("train_inputs shape is {}".format(X_train.shape))
print("train_target shape is {}".format(y_train.shape))
print("test_inputs shape is {}".format(X_test.shape))
print("test_target shape is {}".format(y_test.shape))
```

```
train_inputs shape is (1488, 21)
train_target shape is (1488,)
test_inputs shape is (638, 21)
test_target shape is (638,)
```

```
# Let us check if they are still standardized
X_test.describe().loc[["min","max"]]
```

```
# X_test is used to check here but X_train is also standardized
```

```
     baseline value  accelerations  ...  histogram_variance
histogram_tendency
```

```
min             0.0            0.0  ...                 0.0
0.0
max             1.0            1.0  ...                 1.0
1.0

[2 rows x 21 columns]
```

```
from sklearn.ensemble import RandomForestClassifier

RandomForestModel = RandomForestClassifier()
RandomForestModel.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

Classification Metrics prior to Hypertuning

```
y_predictions_randomforest = RandomForestModel.predict(X_test)
accuracy_score(y_test, y_predictions_randomforest)
```

```
0.9278996865203761
```

```
confusion_matrix(y_test, y_predictions_randomforest, normalize =
"true")
```

```
array([[0.96407186, 0.03393214, 0.00199601],
       [0.2345679 , 0.75308642, 0.01234568],
       [0.07142857, 0.07142857, 0.85714286]])
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predictions_randomforest))
```

```
              precision    recall  f1-score   support

         1.0       0.95      0.96      0.96       501
         2.0       0.74      0.75      0.75        81
         3.0       0.96      0.86      0.91        56

    accuracy                           0.93       638
   macro avg       0.89      0.86      0.87       638
weighted avg       0.93      0.93      0.93       638
```

Overall, Random Forest Model is performing significantly better than the Logistic Regression Model

Hypertuning our Random Forest model

```
randomforest_random_grid = {'n_estimators': [100,200,300,400],
                            'max_depth': [4,7,10,15],
                            'min_samples_leaf': [1, 2, 4],
                            'criterion':['gini', 'entropy']}
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
clf_randomforest = GridSearchCV(RandomForestModel,
randomforest_random_grid, cv=2, verbose=True, n_jobs=-1)

best_clf_randomforest = clf_randomforest.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 96 candidates, totalling 192 fits
```

Classification Metrics on Optimized Random Forest Model

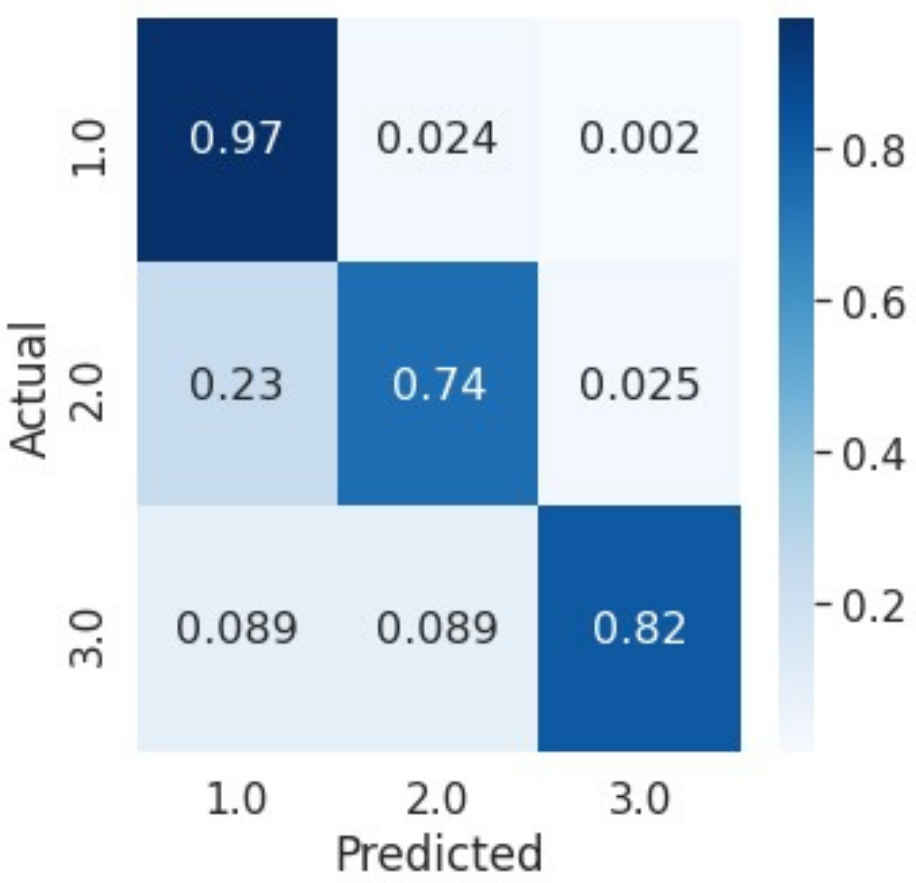
```
best_clf_randomforest.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': 7,
 'min_samples_leaf': 1,
 'n_estimators': 100}
```

```
RandomForestModel_optimized = RandomForestClassifier(criterion=
'entropy',
 max_depth= 15,
 min_samples_leaf= 1,
 n_estimators= 200)
RandomForestModel_optimized.fit(X_train, y_train)
y_predictions_randomforest_optimized =
RandomForestModel_optimized.predict(X_test)
print("old accuracy score is: {}".format(accuracy_score(y_test,
y_predictions_randomforest)))
print("optimized accuracy score is: {}".format(accuracy_score(y_test,
y_predictions_randomforest_optimized)))
```

```
old accuracy score is: 0.9278996865203761
optimized accuracy score is: 0.9310344827586207
```

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

data = confusion_matrix(y_test, y_predictions_randomforest_optimized,
normalize = 'true')
df_cm = pd.DataFrame(data,
columns=np.unique(y_predictions_randomforest_optimized), index =
np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```




```
print(classification_report(y_test,
y_predictions_randomforest_optimized))
```

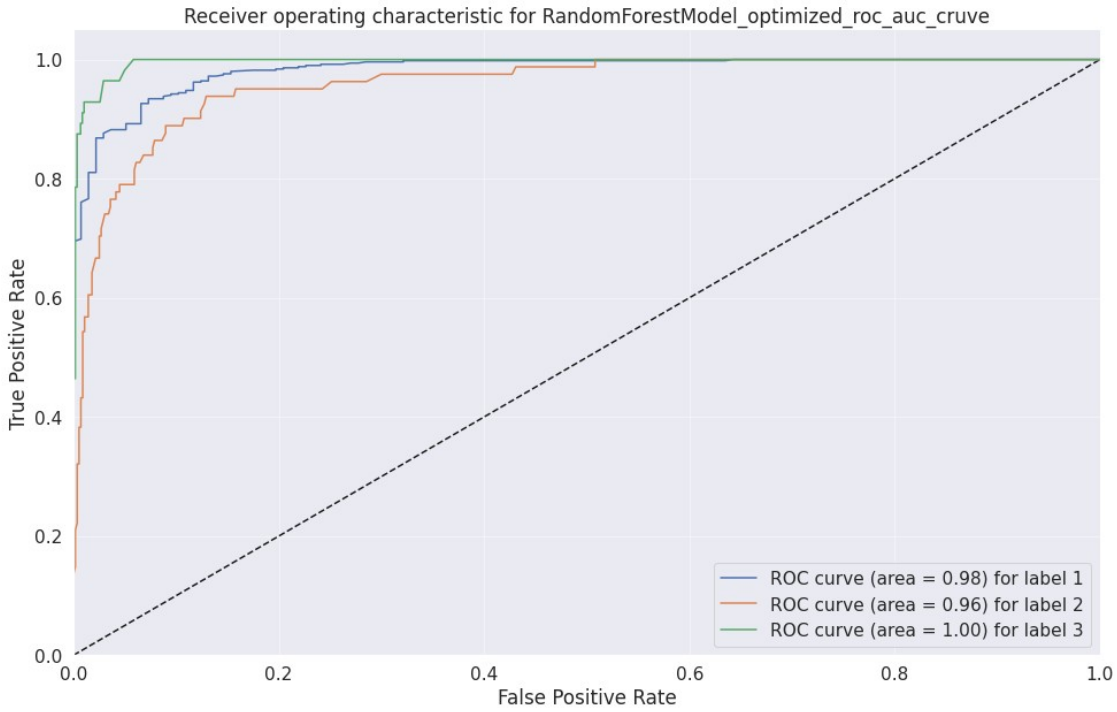
```
              precision    recall  f1-score   support

         1.0       0.95      0.97      0.96       501
         2.0       0.78      0.74      0.76        81
         3.0       0.94      0.82      0.88        56

    accuracy                           0.93       638
   macro avg       0.89      0.85      0.87       638
weighted avg       0.93      0.93      0.93       638
```

```
print("Previous ROC_AUC score {}".format(roc_auc_score(y_test,
RandomForestModel.predict_proba(X_test), multi_class='ovo',
average='weighted')
))
print("new ROC_AUC Score {}".format(roc_auc_score(y_test,
RandomForestModel_optimized.predict_proba(X_test), multi_class='ovo',
average='weighted')))
```

```
Previous ROC_AUC score 0.9798500577153402
new ROC_AUC Score 0.9788664611427491
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

def plot_multiclass_roc(clf, X_test, y_test, n_classes, figsize=(17,
6)):
    try:
        y_score = clf.decision_function(X_test)
        print("Using decision_function method")
    except:
        y_score = clf.predict_proba(X_test)
        print("Using predict_proba method")

    # structures
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # calculate dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:,
i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # roc for each class
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic for
RandomForestModel_optimized_roc_auc_cruve')
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for
label %i' % (roc_auc[i], i+1))
    ax.legend(loc="best")
    ax.grid(alpha=.4)
    sns.despine()
    plt.show()
    return fig, ax

RandomForestModel_optimized_roc_auc_cruve =
plot_multiclass_roc(RandomForestModel_optimized, X_test, y_test,
n_classes=3, figsize=(16, 10))
```

```
Using predict_proba method
```



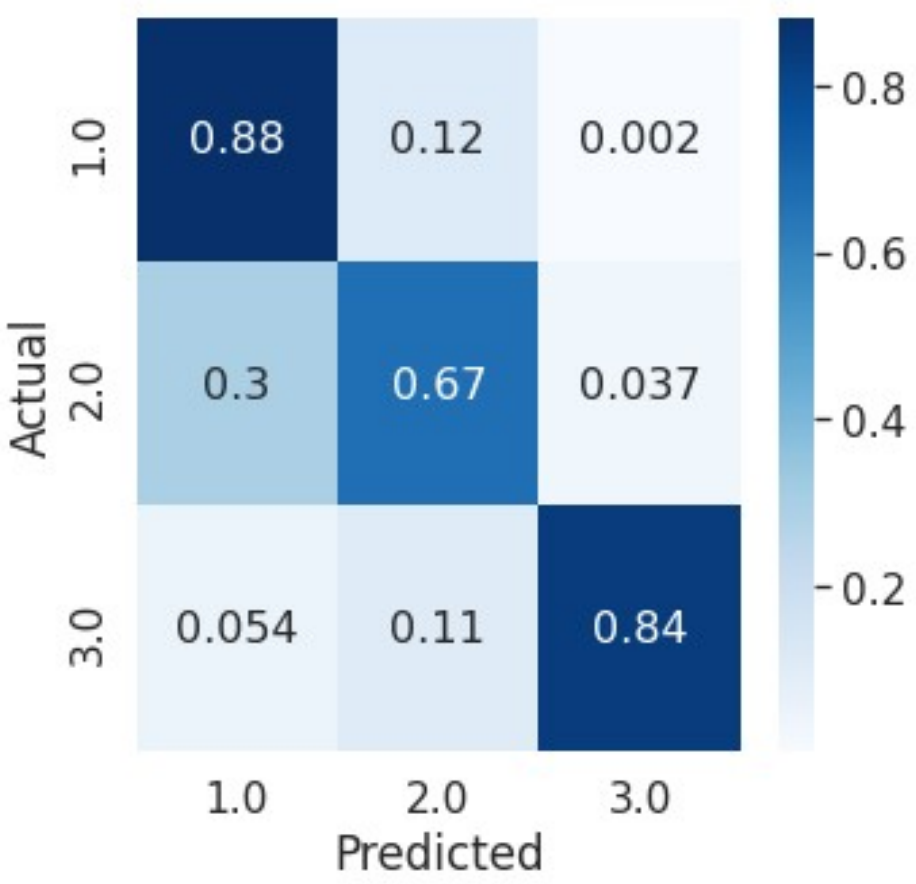
Decision Tree Model

```
from sklearn.tree import DecisionTreeClassifier

decisiontree_model = DecisionTreeClassifier()
decisiontree_model.fit(X_train, y_train)

DecisionTreeClassifier()
```

Classification Metrics prior to hypertuning

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

y_predictions_decisiontree = decisiontree_model.predict(X_test)
print("accuracy {}".format(accuracy_score(y_test,
y_predictions_decisiontree)))

accuracy 0.8510971786833855

data = confusion_matrix(y_test, y_predictions_decisiontree, normalize
= 'true')
df_cm = pd.DataFrame(data,
columns=np.unique(y_predictions_decisiontree), index =
np.unique(y_test))
df_cm.index.name = 'Actual'
```

```
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```



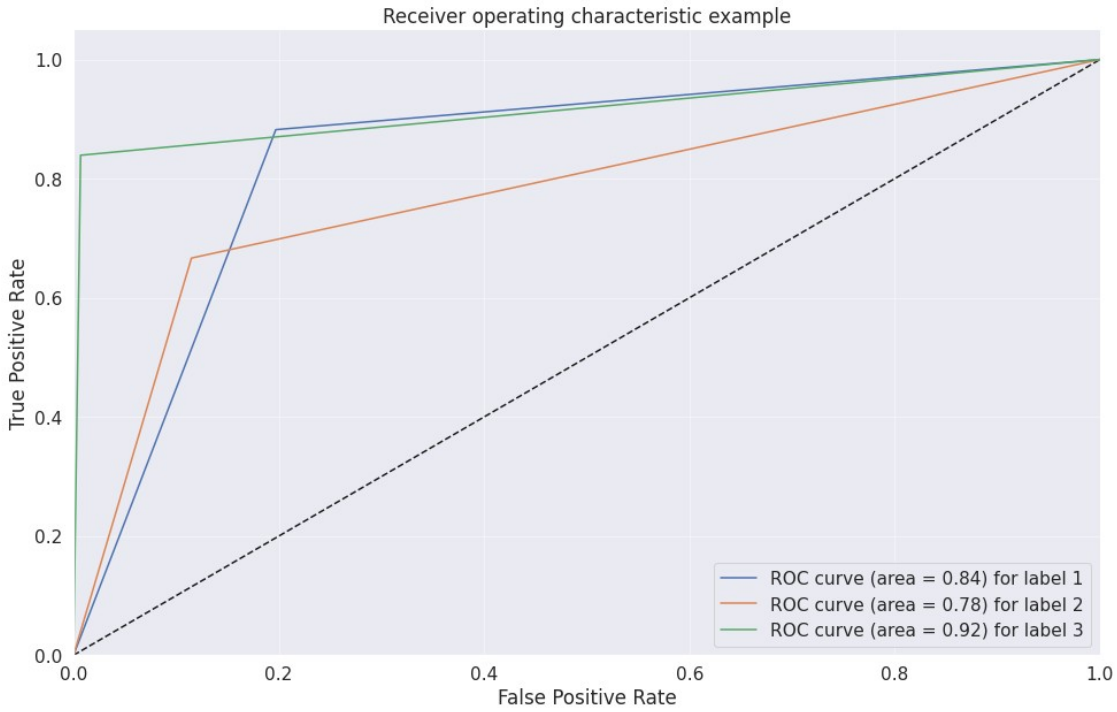

```
print(classification_report(y_test, y_predictions_decisiontree))
```

```
              precision    recall  f1-score   support

         1.0       0.94      0.88      0.91       501
         2.0       0.46      0.67      0.54        81
         3.0       0.92      0.84      0.88        56

    accuracy                           0.85       638
   macro avg       0.77      0.80      0.78       638
weighted avg       0.88      0.85      0.86       638
```

```
roc_auc_score(y_test, decisiontree_model.predict_proba(X_test),
multi_class='ovo', average='weighted')
```

```
0.8479895158493782
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

def plot_multiclass_roc(clf, X_test, y_test, n_classes, figsize=(17,
6)):
    y_score = decisiontree_model.predict_proba(X_test)

    # structures
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # calculate dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:,
i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # roc for each class
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic example')
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for
label %i' % (roc_auc[i], i+1))
    ax.legend(loc="best")
    ax.grid(alpha=.4)
    sns.despine()
    plt.show()

plot_multiclass_roc(decisiontree_model, X_test, y_test, n_classes=3,
figsize=(16, 10))
```



Tuning the Decision Tree Model

```
decisiontree_random_grid = {
                           'max_depth': [4,7,10,15],
                           'min_samples_leaf': [1, 2, 4],
                           'criterion':['gini', 'entropy'],
                           }

clf_decisiontree = GridSearchCV(decisiontree_model,
decisiontree_random_grid, cv=5, verbose=True, n_jobs=-1)

best_clf_decisiontree = clf_decisiontree.fit(X_train, y_train)

Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

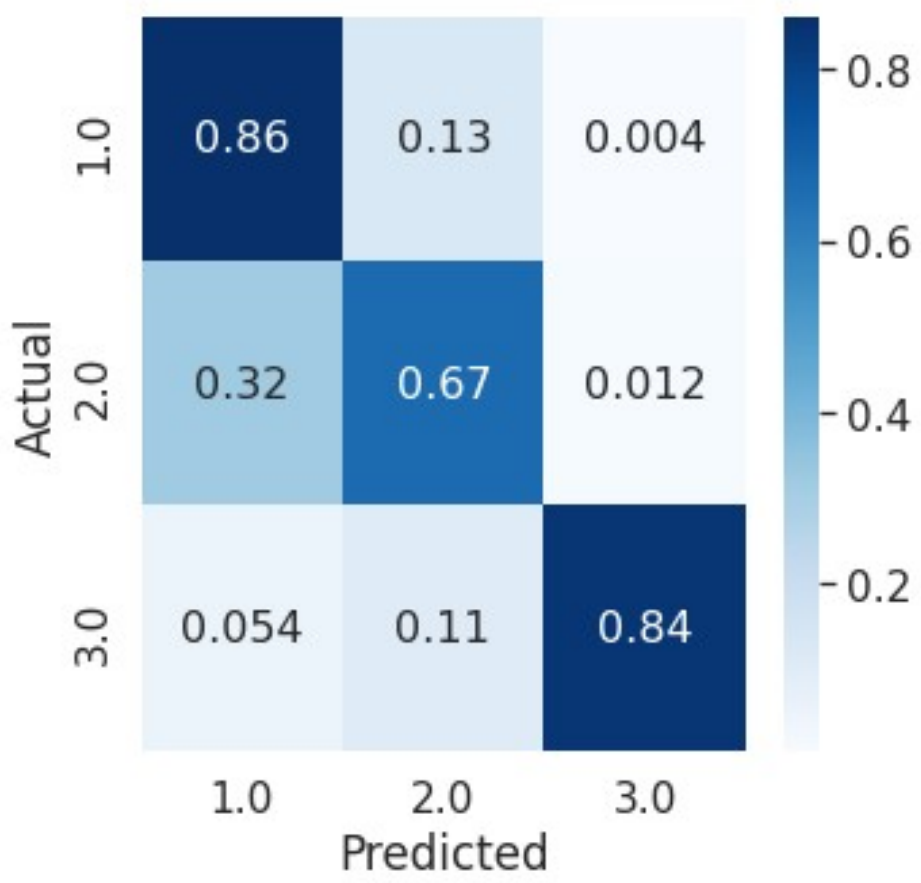
Classification Metrics on Optimized Decision Tree Model

```
best_clf_decisiontree.best_params_

{'criterion': 'gini', 'max_depth': 15, 'min_samples_leaf': 1}

decisiontree_model_optimized = DecisionTreeClassifier(criterion=
'gini', max_depth= 15,
 min_samples_leaf= 1)
decisiontree_model_optimized.fit(X_train, y_train)
y_predictions_decisiontree_optimized =
decisiontree_model_optimized.predict(X_test)
print("old accuracy score is: {}".format(accuracy_score(y_test,
y_predictions_decisiontree)))
print("optimized accuracy score is: {}".format(accuracy_score(y_test,
y_predictions_decisiontree_optimized)))
```

```
old accuracy score is: 0.8510971786833855
optimized accuracy score is: 0.835423197492163
```

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

```
data = confusion_matrix(y_test, y_predictions_decisiontree_optimized,
normalize = 'true')
df_cm = pd.DataFrame(data,
columns=np.unique(y_predictions_decisiontree_optimized), index =
np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```




```
print(classification_report(y_test,
y_predictions_decisiontree_optimized))
```

```
              precision    recall  f1-score   support
```

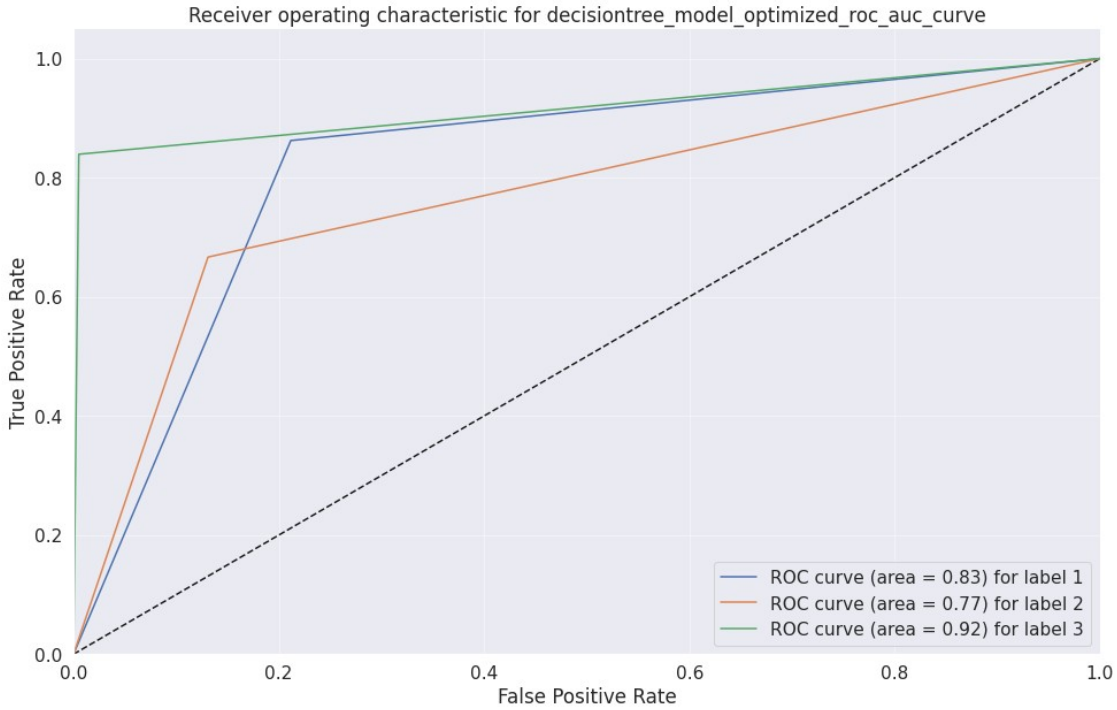
```
         1.0       0.94      0.86      0.90       501
         2.0       0.43      0.67      0.52        81
         3.0       0.94      0.84      0.89        56

    accuracy                           0.84       638
   macro avg       0.77      0.79      0.77       638
weighted avg       0.87      0.84      0.85       638
```

```
print("Previous ROC_AUC score {}".format(roc_auc_score(y_test,
decisiontree_model.predict_proba(X_test), multi_class='ovo',
average='weighted')
))
print("new ROC_AUC Score {}".format(roc_auc_score(y_test,
decisiontree_model_optimized.predict_proba(X_test), multi_class='ovo',
average='weighted')))
```

```
Previous ROC_AUC score 0.8479895158493782
new ROC_AUC Score 0.8391162690769297
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

def plot_multiclass_roc(clf, X_test, y_test, n_classes, figsize=(17,
6)):
    y_score = clf.predict_proba(X_test)

    # structures
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # calculate dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:,
i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # roc for each class
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic for
decisiontree_model_optimized_roc_auc_curve')
    for i in range(n_classes):
```

```
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for
label %i' % (roc_auc[i], i+1))
    ax.legend(loc="best")
    ax.grid(alpha=.4)
    sns.despine()
    plt.show()
    return fig, ax

decisiontree_model_optimized_roc_auc_curve =
plot_multiclass_roc(decisiontree_model_optimized, X_test, y_test,
n_classes=3, figsize=(16, 10))
```



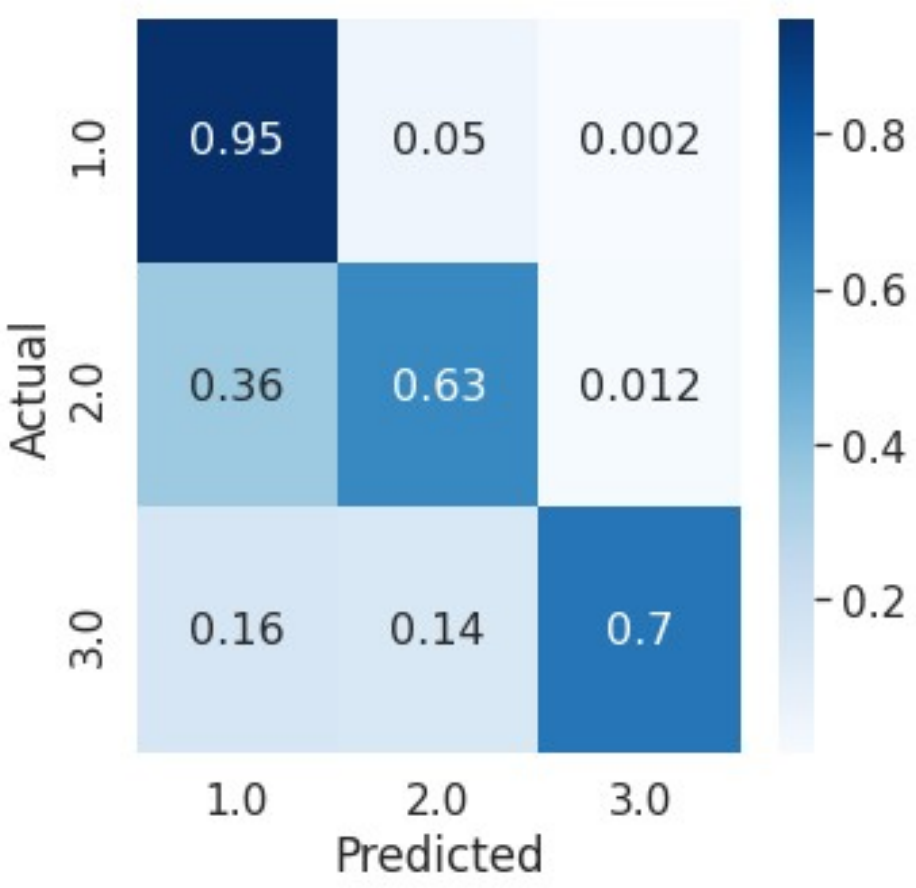
KNN model

```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=3)

knn_model = knn_model.fit(X_train, y_train)

y_pred_knn = knn_model.predict(X_test)

print(accuracy_score(y_test, y_pred_knn))
```

```
0.8855799373040752
```

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

```
import numpy as np

data_confusion = confusion_matrix(y_test, y_pred_knn, normalize =
'true')
df_cm = pd.DataFrame(data_confusion, columns=np.unique(y_pred_knn),
index = np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```



Tuning KNN hyperparameters

```
knn_random_grid = {
                            'leaf_size':[n for n in range(1,20)],
                            'n_neighbors':[n for n in range(1,20)],
                            'p':[1,2]
                            }

clf_knn = GridSearchCV(knn_model, knn_random_grid, cv=2, verbose=True,
n_jobs=-1)

best_clf_knn = clf_knn.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 722 candidates, totalling 1444 fits
```

Classifier Metrics of Optimized KNN Model

```
best_clf_knn.best_params_
```

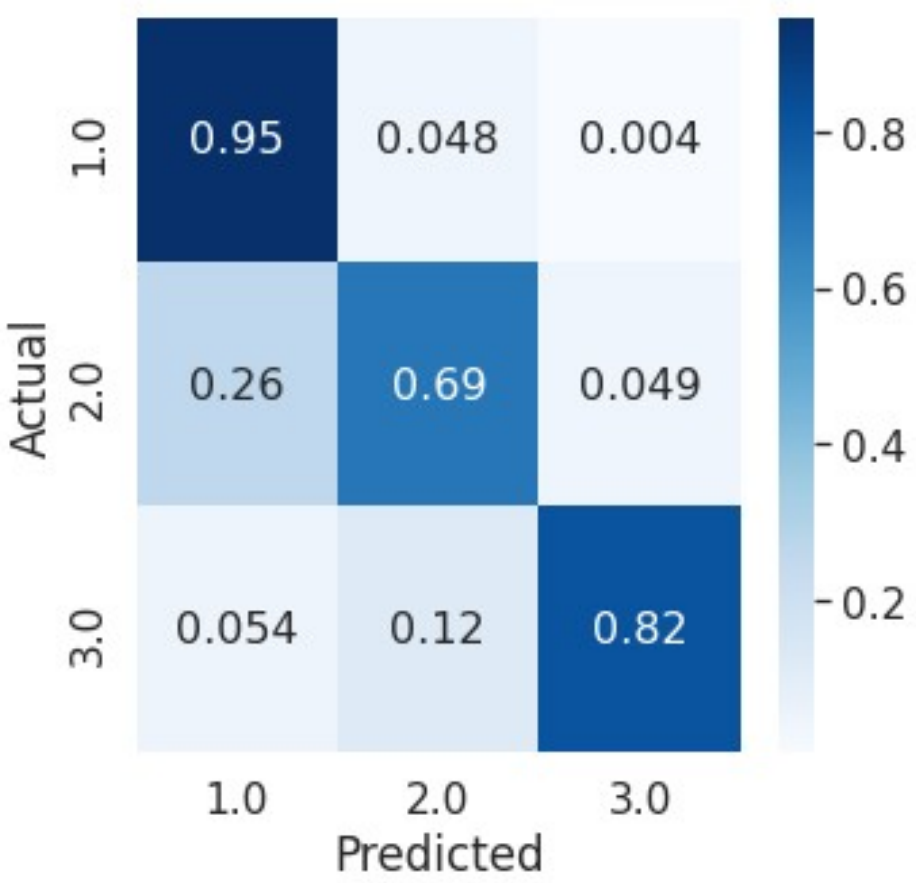
```
{'leaf_size': 1, 'n_neighbors': 1, 'p': 2}
```

```
knn_model_optimized = KNeighborsClassifier(leaf_size = 1,
n_neighbors=1, p = 2)
knn_model_optimized.fit(X_train, y_train)
y_predictions_knn_optimized = knn_model_optimized.predict(X_test)
print("old accuracy score is: {}".format(accuracy_score(y_test,
y_pred_knn)))
print("optimized accuracy score is: {}".format(accuracy_score(y_test,
y_predictions_knn_optimized)))
```

```
old accuracy score is: 0.8855799373040752
optimized accuracy score is: 0.9043887147335423
```

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

```
data = confusion_matrix(y_test, y_predictions_knn_optimized, normalize
= 'true')
df_cm = pd.DataFrame(data,
columns=np.unique(y_predictions_knn_optimized), index =
np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```




```
print(classification_report(y_test, y_predictions_knn_optimized))
```

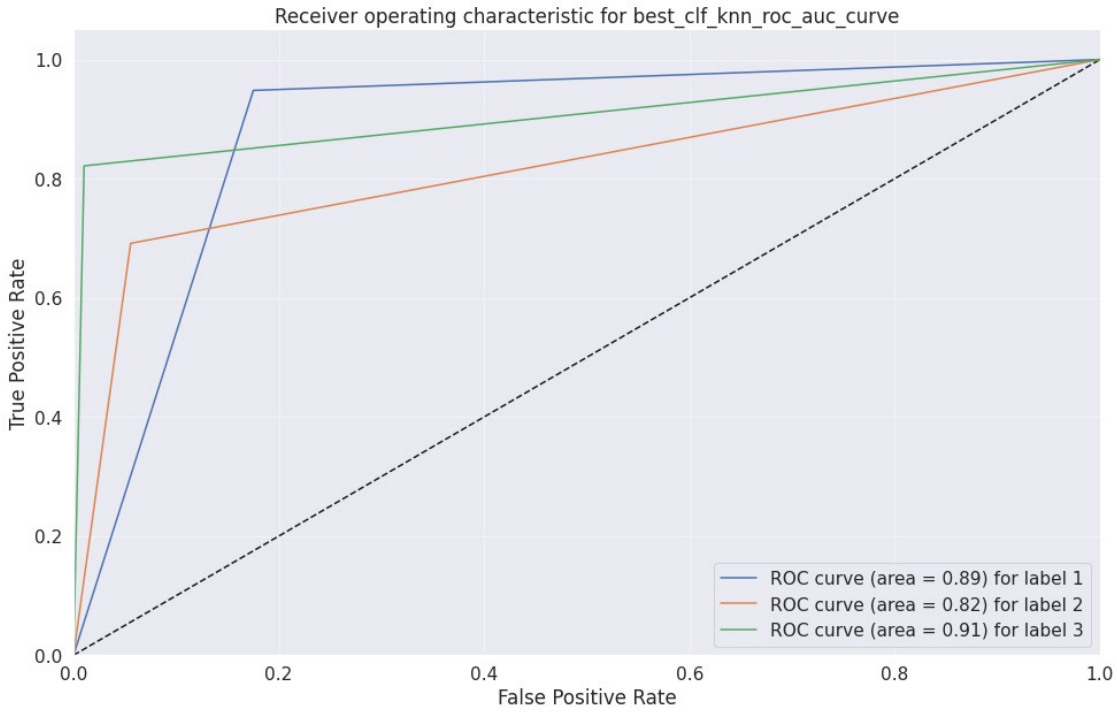
```
              precision    recall  f1-score   support

         1.0       0.95      0.95      0.95       501
         2.0       0.64      0.69      0.67        81
         3.0       0.88      0.82      0.85        56

    accuracy                           0.90       638
   macro avg       0.83      0.82      0.82       638
weighted avg       0.91      0.90      0.91       638
```

```
print("Previous ROC_AUC score {}".format(roc_auc_score(y_test,
knn_model.predict_proba(X_test), multi_class='ovo',
average='weighted')
))
print("new ROC_AUC Score {}".format(roc_auc_score(y_test,
knn_model_optimized.predict_proba(X_test), multi_class='ovo',
average='weighted')))
```

```
Previous ROC_AUC score 0.9271871045667204
new ROC_AUC Score 0.8746719570470775
```

New ROC_AUC score lowered, which could be a bad sign

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

def plot_multiclass_roc(clf, X_test, y_test, n_classes, figsize=(17,
6)):
    y_score = clf.predict_proba(X_test)

    # structures
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # calculate dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:,
i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # roc for each class
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic for
best_clf_knn_roc_auc_curve')
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for
label %i' % (roc_auc[i], i+1))
    ax.legend(loc="best")
    ax.grid(alpha=.4)
    sns.despine()
    plt.show()
    return fig, ax

best_clf_knn_roc_auc_curve = plot_multiclass_roc(best_clf_knn, X_test,
y_test, n_classes=3, figsize=(16, 10))
```



Support Vector Machines

Training Support Vector Classifier without tuning

```
from sklearn.svm import LinearSVC

svc_model = LinearSVC()
svc_model.fit(X_train, y_train)

LinearSVC()

y_predictions_svc = svc_model.predict(X_test)

y_test.value_counts()

1.0    501
2.0     81
3.0     56
Name: fetal_health, dtype: int64

svc_model.decision_function(X_test)[0]

array([ 1.33935466, -1.21777251, -1.96631963])
```

Classification Metrics w/o hyperparameter tuning

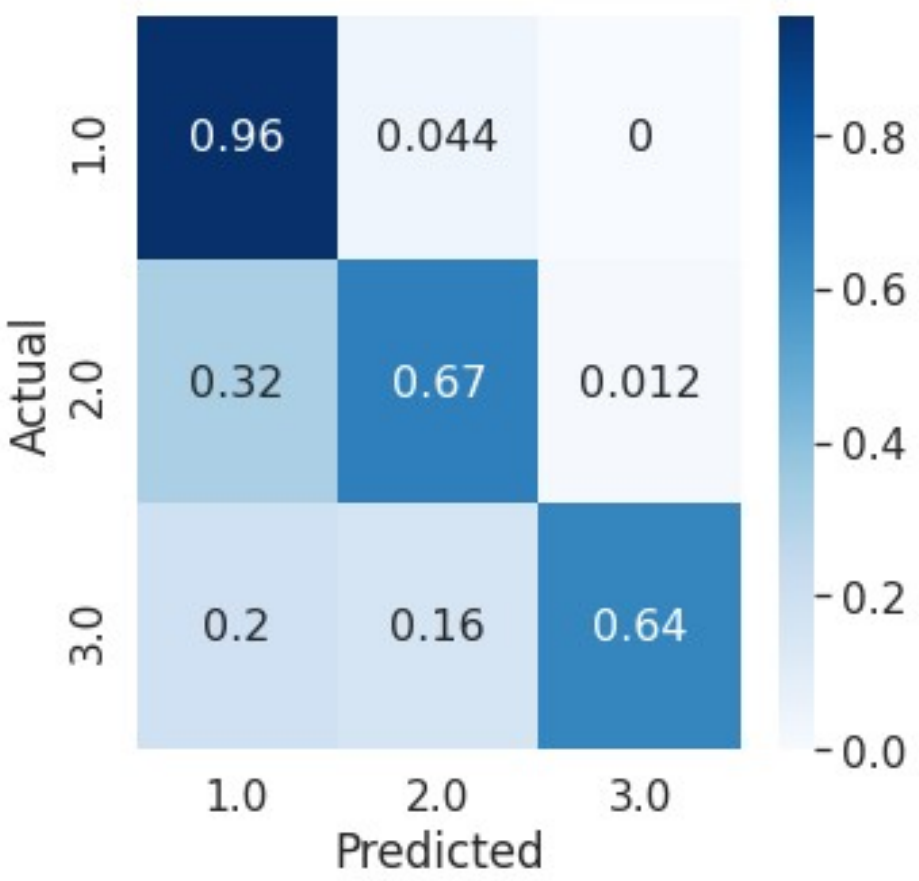
```
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score, roc_curve, roc_auc_score

print(classification_report(y_test, y_predictions_svc))
```

```
              precision    recall  f1-score   support

         1.0       0.93      0.96      0.94       501
         2.0       0.64      0.67      0.65        81
         3.0       0.97      0.64      0.77        56

    accuracy                           0.89       638
   macro avg       0.85      0.76      0.79       638
weighted avg       0.90      0.89      0.89       638
```

```
print(accuracy_score(y_test, y_predictions_svc))
```

```
0.8918495297805643
```

```
data = confusion_matrix(y_test, y_predictions_svc, normalize = 'true')
df_cm = pd.DataFrame(data, columns=np.unique(y_predictions_svc), index
= np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```

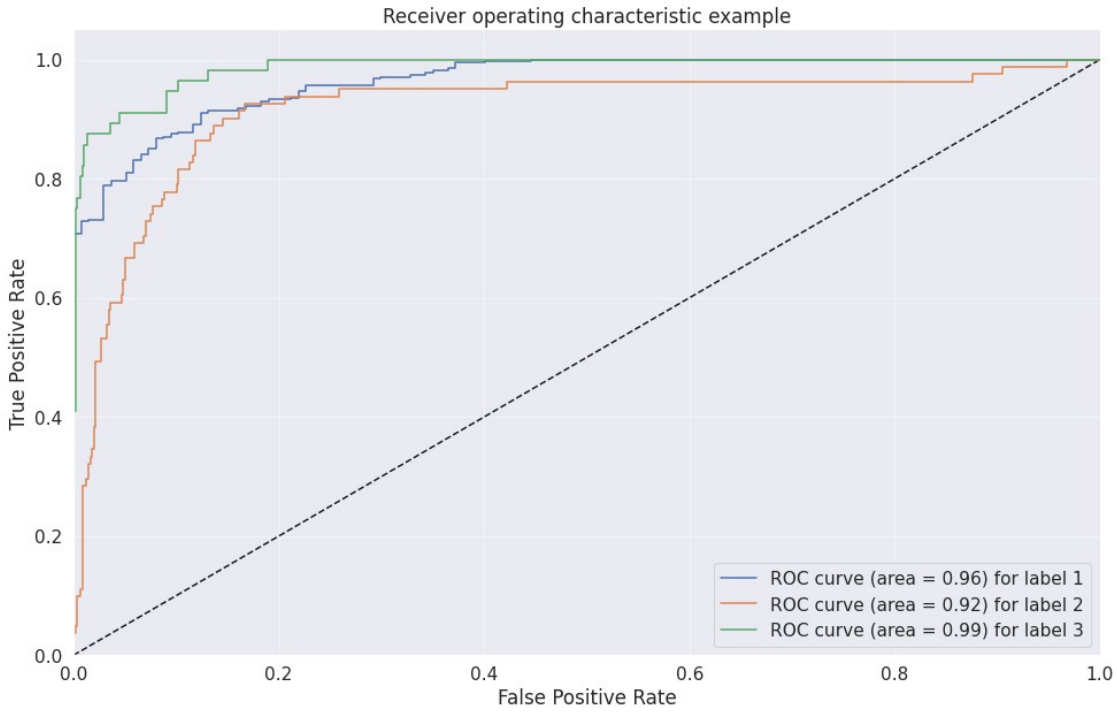


```
print("ROC_AUC score for unoptimized SVC model
{}".format(roc_auc_score(y_test, svc_model.predict_proba(X_test),
multi_class='ovo', average='weighted')
))
```

```
ROC_AUC score for unoptimized SVC model 0.964437590688322
```

New ROC_AUC score lowered, which could be a bad sign

```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

def plot_multiclass_roc(clf, X_test, y_test, n_classes, figsize=(17,
6)):
    y_score = clf.decision_function(X_test)

    # structures
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # calculate dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:,
i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # roc for each class
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic example')
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for
label %i' % (roc_auc[i], i+1))
    ax.legend(loc="best")
    ax.grid(alpha=.4)
    sns.despine()
    plt.show()

plot_multiclass_roc(svc_model, X_test, y_test, n_classes=3,
figsize=(16, 10))
```



Optimizing the SVC model by tuning its parameters

```
param_grid = {'C': [0.1,1, 10, 100, 1000],
              'gamma': [1,0.1,0.01,0.001,0.0001],
              'kernel': ['rbf', 'poly', 'sigmoid', 'linear']}

from sklearn.model_selection import GridSearchCV

clf_svc = GridSearchCV(SVC(probability =
True),param_grid,refit=True,verbose=3)

clf_svc.fit(X_train, y_train)

Fitting 5 folds for each of 100 candidates, totalling 500 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.872 total
time=   0.3s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.849 total
time=   0.3s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.842 total
time=   0.2s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.845 total
time=   0.3s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.845 total
time=   0.2s
[CV 1/5] END .......C=0.1, gamma=1, kernel=poly;, score=0.923 total
time=   0.1s
[CV 2/5] END .......C=0.1, gamma=1, kernel=poly;, score=0.926 total
time=   0.1s
[CV 3/5] END .......C=0.1, gamma=1, kernel=poly;, score=0.903 total
time=   0.1s
```

```
[CV 4/5] END .......C=0.1, gamma=1, kernel=poly;, score=0.902 total
time=   0.1s
[CV 5/5] END .......C=0.1, gamma=1, kernel=poly;, score=0.896 total
time=   0.1s
[CV 1/5] END ....C=0.1, gamma=1, kernel=sigmoid;, score=0.775 total
time=   0.4s
[CV 2/5] END ....C=0.1, gamma=1, kernel=sigmoid;, score=0.772 total
time=   0.3s
[CV 3/5] END ....C=0.1, gamma=1, kernel=sigmoid;, score=0.775 total
time=   0.3s
[CV 4/5] END ....C=0.1, gamma=1, kernel=sigmoid;, score=0.778 total
time=   0.4s
[CV 5/5] END ....C=0.1, gamma=1, kernel=sigmoid;, score=0.774 total
time=   0.4s
[CV 1/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.879 total
time=   0.1s
[CV 2/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.849 total
time=   0.1s
[CV 3/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.839 total
time=   0.1s
[CV 4/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.852 total
time=   0.1s
[CV 5/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.852 total
time=   0.1s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.775 total
time=   0.3s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.775 total
time=   0.3s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.778 total
time=   0.2s
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END .....C=0.1, gamma=0.1, kernel=poly;, score=0.775 total
time=   0.2s
[CV 2/5] END .....C=0.1, gamma=0.1, kernel=poly;, score=0.775 total
time=   0.2s
[CV 3/5] END .....C=0.1, gamma=0.1, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END .....C=0.1, gamma=0.1, kernel=poly;, score=0.778 total
time=   0.2s
[CV 5/5] END .....C=0.1, gamma=0.1, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.775 total
time=   0.3s
[CV 2/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.775 total
time=   0.3s
[CV 3/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.775 total
time=   0.3s
```

```
[CV 4/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.778 total
time=   0.3s
[CV 5/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.774 total
time=   0.3s
[CV 1/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.879 total
time=   0.1s
[CV 2/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.849 total
time=   0.1s
[CV 3/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.839 total
time=   0.1s
[CV 4/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.852 total
time=   0.1s
[CV 5/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.852 total
time=   0.1s
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.775 total
time=   0.3s
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.778 total
time=   0.2s
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END ....C=0.1, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ....C=0.1, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END ....C=0.1, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END ....C=0.1, gamma=0.01, kernel=poly;, score=0.778 total
time=   0.2s
[CV 5/5] END ....C=0.1, gamma=0.01, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END .C=0.1, gamma=0.01, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 2/5] END .C=0.1, gamma=0.01, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 3/5] END .C=0.1, gamma=0.01, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 4/5] END .C=0.1, gamma=0.01, kernel=sigmoid;, score=0.778 total
time=   0.2s
[CV 5/5] END .C=0.1, gamma=0.01, kernel=sigmoid;, score=0.774 total
time=   0.2s
[CV 1/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.879 total
time=   0.1s
[CV 2/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.849 total
time=   0.1s
[CV 3/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.839 total
time=   0.1s
```

```
[CV 4/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.852 total
time=   0.1s
[CV 5/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.852 total
time=   0.1s
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.778 total
time=   0.2s
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END ...C=0.1, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.2s
[CV 2/5] END ...C=0.1, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.2s
[CV 3/5] END ...C=0.1, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.2s
[CV 4/5] END ...C=0.1, gamma=0.001, kernel=poly;, score=0.778 total
time=   0.2s
[CV 5/5] END ...C=0.1, gamma=0.001, kernel=poly;, score=0.774 total
time=   0.2s
[CV 1/5] END C=0.1, gamma=0.001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 2/5] END C=0.1, gamma=0.001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 3/5] END C=0.1, gamma=0.001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 4/5] END C=0.1, gamma=0.001, kernel=sigmoid;, score=0.778 total
time=   0.2s
[CV 5/5] END C=0.1, gamma=0.001, kernel=sigmoid;, score=0.774 total
time=   0.2s
[CV 1/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.879 total
time=   0.1s
[CV 2/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.849 total
time=   0.1s
[CV 3/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.839 total
time=   0.1s
[CV 4/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.852 total
time=   0.1s
[CV 5/5] END .C=0.1, gamma=0.001, kernel=linear;, score=0.852 total
time=   0.1s
[CV 1/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 2/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 3/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
```

```
[CV 4/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.778 total
time=   0.2s
[CV 5/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END ..C=0.1, gamma=0.0001, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END C=0.1, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 2/5] END C=0.1, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 3/5] END C=0.1, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 4/5] END C=0.1, gamma=0.0001, kernel=sigmoid;, score=0.778 total
time=   0.2s
[CV 5/5] END C=0.1, gamma=0.0001, kernel=sigmoid;, score=0.774 total
time=   0.2s
[CV 1/5] END C=0.1, gamma=0.0001, kernel=linear;, score=0.879 total
time=   0.1s
[CV 2/5] END C=0.1, gamma=0.0001, kernel=linear;, score=0.849 total
time=   0.1s
[CV 3/5] END C=0.1, gamma=0.0001, kernel=linear;, score=0.839 total
time=   0.1s
[CV 4/5] END C=0.1, gamma=0.0001, kernel=linear;, score=0.852 total
time=   0.1s
[CV 5/5] END C=0.1, gamma=0.0001, kernel=linear;, score=0.852 total
time=   0.1s
[CV 1/5] END .........C=1, gamma=1, kernel=rbf;, score=0.909 total
time=   0.2s
[CV 2/5] END .........C=1, gamma=1, kernel=rbf;, score=0.923 total
time=   0.2s
[CV 3/5] END .........C=1, gamma=1, kernel=rbf;, score=0.889 total
time=   0.2s
[CV 4/5] END .........C=1, gamma=1, kernel=rbf;, score=0.896 total
time=   0.2s
[CV 5/5] END .........C=1, gamma=1, kernel=rbf;, score=0.896 total
time=   0.2s
[CV 1/5] END .........C=1, gamma=1, kernel=poly;, score=0.940 total
time=   0.2s
[CV 2/5] END .........C=1, gamma=1, kernel=poly;, score=0.923 total
time=   0.2s
[CV 3/5] END .........C=1, gamma=1, kernel=poly;, score=0.913 total
time=   0.2s
```

```
[CV 4/5] END .........C=1, gamma=1, kernel=poly;, score=0.906 total
time=   0.2s
[CV 5/5] END .........C=1, gamma=1, kernel=poly;, score=0.909 total
time=   0.1s
[CV 1/5] END ......C=1, gamma=1, kernel=sigmoid;, score=0.671 total
time=   0.3s
[CV 2/5] END ......C=1, gamma=1, kernel=sigmoid;, score=0.691 total
time=   0.3s
[CV 3/5] END ......C=1, gamma=1, kernel=sigmoid;, score=0.674 total
time=   0.4s
[CV 4/5] END ......C=1, gamma=1, kernel=sigmoid;, score=0.673 total
time=   0.4s
[CV 5/5] END ......C=1, gamma=1, kernel=sigmoid;, score=0.704 total
time=   0.3s
[CV 1/5] END .......C=1, gamma=1, kernel=linear;, score=0.916 total
time=   0.1s
[CV 2/5] END .......C=1, gamma=1, kernel=linear;, score=0.919 total
time=   0.1s
[CV 3/5] END .......C=1, gamma=1, kernel=linear;, score=0.879 total
time=   0.1s
[CV 4/5] END .......C=1, gamma=1, kernel=linear;, score=0.882 total
time=   0.1s
[CV 5/5] END .......C=1, gamma=1, kernel=linear;, score=0.886 total
time=   0.1s
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.893 total
time=   0.2s
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.883 total
time=   0.2s
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.875 total
time=   0.2s
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.869 total
time=   0.2s
[CV 1/5] END .......C=1, gamma=0.1, kernel=poly;, score=0.795 total
time=   0.2s
[CV 2/5] END .......C=1, gamma=0.1, kernel=poly;, score=0.795 total
time=   0.1s
[CV 3/5] END .......C=1, gamma=0.1, kernel=poly;, score=0.799 total
time=   0.1s
[CV 4/5] END .......C=1, gamma=0.1, kernel=poly;, score=0.805 total
time=   0.2s
[CV 5/5] END .......C=1, gamma=0.1, kernel=poly;, score=0.788 total
time=   0.1s
[CV 1/5] END ....C=1, gamma=0.1, kernel=sigmoid;, score=0.862 total
time=   0.3s
[CV 2/5] END ....C=1, gamma=0.1, kernel=sigmoid;, score=0.836 total
time=   0.3s
[CV 3/5] END ....C=1, gamma=0.1, kernel=sigmoid;, score=0.839 total
time=   0.3s
```

```
[CV 4/5] END ....C=1, gamma=0.1, kernel=sigmoid;, score=0.845 total
time=   0.3s
[CV 5/5] END ....C=1, gamma=0.1, kernel=sigmoid;, score=0.845 total
time=   0.3s
[CV 1/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.916 total
time=   0.1s
[CV 2/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.919 total
time=   0.1s
[CV 3/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.879 total
time=   0.1s
[CV 4/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.882 total
time=   0.1s
[CV 5/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.886 total
time=   0.1s
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.775 total
time=   0.3s
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.775 total
time=   0.3s
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.775 total
time=   0.3s
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.778 total
time=   0.3s
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END ......C=1, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ......C=1, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END ......C=1, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END ......C=1, gamma=0.01, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END ......C=1, gamma=0.01, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END ...C=1, gamma=0.01, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 2/5] END ...C=1, gamma=0.01, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 3/5] END ...C=1, gamma=0.01, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 4/5] END ...C=1, gamma=0.01, kernel=sigmoid;, score=0.778 total
time=   0.2s
[CV 5/5] END ...C=1, gamma=0.01, kernel=sigmoid;, score=0.774 total
time=   0.2s
[CV 1/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.916 total
time=   0.1s
[CV 2/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.919 total
time=   0.1s
[CV 3/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.879 total
time=   0.1s
```

```
[CV 4/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.882 total
time=   0.1s
[CV 5/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.886 total
time=   0.1s
[CV 1/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 2/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 3/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.775 total
time=   0.3s
[CV 4/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.778 total
time=   0.3s
[CV 5/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END .....C=1, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END .....C=1, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END .....C=1, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END .....C=1, gamma=0.001, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END .....C=1, gamma=0.001, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END ..C=1, gamma=0.001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 2/5] END ..C=1, gamma=0.001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 3/5] END ..C=1, gamma=0.001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 4/5] END ..C=1, gamma=0.001, kernel=sigmoid;, score=0.778 total
time=   0.2s
[CV 5/5] END ..C=1, gamma=0.001, kernel=sigmoid;, score=0.774 total
time=   0.2s
[CV 1/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.916 total
time=   0.1s
[CV 2/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.919 total
time=   0.1s
[CV 3/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.879 total
time=   0.1s
[CV 4/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.882 total
time=   0.1s
[CV 5/5] END ...C=1, gamma=0.001, kernel=linear;, score=0.886 total
time=   0.1s
[CV 1/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 2/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 3/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
```

```
[CV 4/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.778 total
time=   0.2s
[CV 5/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END ....C=1, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ....C=1, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END ....C=1, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END ....C=1, gamma=0.0001, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END ....C=1, gamma=0.0001, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END .C=1, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 2/5] END .C=1, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 3/5] END .C=1, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 4/5] END .C=1, gamma=0.0001, kernel=sigmoid;, score=0.778 total
time=   0.2s
[CV 5/5] END .C=1, gamma=0.0001, kernel=sigmoid;, score=0.774 total
time=   0.2s
[CV 1/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.916 total
time=   0.1s
[CV 2/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.919 total
time=   0.1s
[CV 3/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.879 total
time=   0.1s
[CV 4/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.882 total
time=   0.1s
[CV 5/5] END ..C=1, gamma=0.0001, kernel=linear;, score=0.886 total
time=   0.1s
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.936 total
time=   0.2s
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.936 total
time=   0.2s
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.896 total
time=   0.2s
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.909 total
time=   0.2s
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.896 total
time=   0.2s
[CV 1/5] END ........C=10, gamma=1, kernel=poly;, score=0.919 total
time=   0.3s
[CV 2/5] END ........C=10, gamma=1, kernel=poly;, score=0.916 total
time=   0.3s
[CV 3/5] END ........C=10, gamma=1, kernel=poly;, score=0.906 total
time=   0.3s
```

```
[CV 4/5] END ........C=10, gamma=1, kernel=poly;, score=0.912 total
time=   0.3s
[CV 5/5] END ........C=10, gamma=1, kernel=poly;, score=0.896 total
time=   0.3s
[CV 1/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.748 total
time=   0.3s
[CV 2/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.587 total
time=   0.3s
[CV 3/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.721 total
time=   0.3s
[CV 4/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.758 total
time=   0.3s
[CV 5/5] END .....C=10, gamma=1, kernel=sigmoid;, score=0.657 total
time=   0.3s
[CV 1/5] END ......C=10, gamma=1, kernel=linear;, score=0.913 total
time=   0.1s
[CV 2/5] END ......C=10, gamma=1, kernel=linear;, score=0.916 total
time=   0.1s
[CV 3/5] END ......C=10, gamma=1, kernel=linear;, score=0.899 total
time=   0.1s
[CV 4/5] END ......C=10, gamma=1, kernel=linear;, score=0.875 total
time=   0.1s
[CV 5/5] END ......C=10, gamma=1, kernel=linear;, score=0.892 total
time=   0.1s
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.916 total
time=   0.2s
[CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.933 total
time=   0.2s
[CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.903 total
time=   0.2s
[CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.889 total
time=   0.2s
[CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.896 total
time=   0.2s
[CV 1/5] END ......C=10, gamma=0.1, kernel=poly;, score=0.903 total
time=   0.1s
[CV 2/5] END ......C=10, gamma=0.1, kernel=poly;, score=0.899 total
time=   0.1s
[CV 3/5] END ......C=10, gamma=0.1, kernel=poly;, score=0.872 total
time=   0.1s
[CV 4/5] END ......C=10, gamma=0.1, kernel=poly;, score=0.872 total
time=   0.1s
[CV 5/5] END ......C=10, gamma=0.1, kernel=poly;, score=0.892 total
time=   0.1s
[CV 1/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.909 total
time=   0.2s
[CV 2/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.919 total
time=   0.2s
[CV 3/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.876 total
time=   0.2s
```

```
[CV 4/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.879 total
time=   0.2s
[CV 5/5] END ...C=10, gamma=0.1, kernel=sigmoid;, score=0.875 total
time=   0.2s
[CV 1/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.913 total
time=   0.1s
[CV 2/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.916 total
time=   0.1s
[CV 3/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.899 total
time=   0.1s
[CV 4/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.875 total
time=   0.1s
[CV 5/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.892 total
time=   0.1s
[CV 1/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.903 total
time=   0.2s
[CV 2/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.879 total
time=   0.2s
[CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.879 total
time=   0.2s
[CV 5/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.872 total
time=   0.2s
[CV 1/5] END .....C=10, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END .....C=10, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END .....C=10, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END .....C=10, gamma=0.01, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END .....C=10, gamma=0.01, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.879 total
time=   0.2s
[CV 2/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.849 total
time=   0.2s
[CV 3/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.836 total
time=   0.2s
[CV 4/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.852 total
time=   0.2s
[CV 5/5] END ..C=10, gamma=0.01, kernel=sigmoid;, score=0.852 total
time=   0.2s
[CV 1/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.913 total
time=   0.1s
[CV 2/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.916 total
time=   0.1s
[CV 3/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.899 total
time=   0.1s
```

```
[CV 4/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.875 total
time=   0.1s
[CV 5/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.892 total
time=   0.1s
[CV 1/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 2/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 3/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 4/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.778 total
time=   0.3s
[CV 5/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END ....C=10, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ....C=10, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END ....C=10, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END ....C=10, gamma=0.001, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END ....C=10, gamma=0.001, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 2/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 3/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 4/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.778 total
time=   0.2s
[CV 5/5] END .C=10, gamma=0.001, kernel=sigmoid;, score=0.774 total
time=   0.2s
[CV 1/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.913 total
time=   0.1s
[CV 2/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.916 total
time=   0.1s
[CV 3/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.899 total
time=   0.1s
[CV 4/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.875 total
time=   0.1s
[CV 5/5] END ..C=10, gamma=0.001, kernel=linear;, score=0.892 total
time=   0.1s
[CV 1/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.3s
[CV 2/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 3/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
```

```
[CV 4/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.778 total
time=   0.2s
[CV 5/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END ...C=10, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ...C=10, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.2s
[CV 3/5] END ...C=10, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END ...C=10, gamma=0.0001, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END ...C=10, gamma=0.0001, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END C=10, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 2/5] END C=10, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 3/5] END C=10, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 4/5] END C=10, gamma=0.0001, kernel=sigmoid;, score=0.778 total
time=   0.2s
[CV 5/5] END C=10, gamma=0.0001, kernel=sigmoid;, score=0.774 total
time=   0.2s
[CV 1/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.913 total
time=   0.1s
[CV 2/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.916 total
time=   0.1s
[CV 3/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.899 total
time=   0.1s
[CV 4/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.875 total
time=   0.1s
[CV 5/5] END .C=10, gamma=0.0001, kernel=linear;, score=0.892 total
time=   0.1s
[CV 1/5] END ........C=100, gamma=1, kernel=rbf;, score=0.936 total
time=   0.2s
[CV 2/5] END ........C=100, gamma=1, kernel=rbf;, score=0.933 total
time=   0.2s
[CV 3/5] END ........C=100, gamma=1, kernel=rbf;, score=0.909 total
time=   0.2s
[CV 4/5] END ........C=100, gamma=1, kernel=rbf;, score=0.916 total
time=   0.2s
[CV 5/5] END ........C=100, gamma=1, kernel=rbf;, score=0.906 total
time=   0.2s
[CV 1/5] END .......C=100, gamma=1, kernel=poly;, score=0.909 total
time=   0.7s
[CV 2/5] END .......C=100, gamma=1, kernel=poly;, score=0.919 total
time=   0.8s
[CV 3/5] END .......C=100, gamma=1, kernel=poly;, score=0.896 total
time=   0.8s
```

```
[CV 4/5] END .......C=100, gamma=1, kernel=poly;, score=0.902 total
time=   0.8s
[CV 5/5] END .......C=100, gamma=1, kernel=poly;, score=0.875 total
time=   0.8s
[CV 1/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.735 total
time=   0.3s
[CV 2/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.715 total
time=   0.2s
[CV 3/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.721 total
time=   0.3s
[CV 4/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.754 total
time=   0.3s
[CV 5/5] END ....C=100, gamma=1, kernel=sigmoid;, score=0.653 total
time=   0.2s
[CV 1/5] END .....C=100, gamma=1, kernel=linear;, score=0.909 total
time=   0.2s
[CV 2/5] END .....C=100, gamma=1, kernel=linear;, score=0.913 total
time=   0.2s
[CV 3/5] END .....C=100, gamma=1, kernel=linear;, score=0.896 total
time=   0.2s
[CV 4/5] END .....C=100, gamma=1, kernel=linear;, score=0.882 total
time=   0.2s
[CV 5/5] END .....C=100, gamma=1, kernel=linear;, score=0.869 total
time=   0.2s
[CV 1/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.926 total
time=   0.2s
[CV 2/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.936 total
time=   0.2s
[CV 3/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.909 total
time=   0.2s
[CV 4/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.902 total
time=   0.2s
[CV 5/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.892 total
time=   0.2s
[CV 1/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.923 total
time=   0.1s
[CV 2/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.926 total
time=   0.1s
[CV 3/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.903 total
time=   0.1s
[CV 4/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.902 total
time=   0.1s
[CV 5/5] END .....C=100, gamma=0.1, kernel=poly;, score=0.896 total
time=   0.1s
[CV 1/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.889 total
time=   0.2s
[CV 2/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.903 total
time=   0.2s
[CV 3/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.869 total
time=   0.2s
```

```
[CV 4/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.862 total
time=   0.2s
[CV 5/5] END ..C=100, gamma=0.1, kernel=sigmoid;, score=0.862 total
time=   0.2s
[CV 1/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.909 total
time=   0.2s
[CV 2/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.913 total
time=   0.2s
[CV 3/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.896 total
time=   0.2s
[CV 4/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.882 total
time=   0.2s
[CV 5/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.869 total
time=   0.2s
[CV 1/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.913 total
time=   0.2s
[CV 2/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.933 total
time=   0.2s
[CV 3/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.893 total
time=   0.2s
[CV 4/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.886 total
time=   0.2s
[CV 5/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.892 total
time=   0.2s
[CV 1/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.2s
[CV 3/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.775 total
time=   0.2s
[CV 4/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END ....C=100, gamma=0.01, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.916 total
time=   0.2s
[CV 2/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.919 total
time=   0.2s
[CV 3/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.879 total
time=   0.2s
[CV 4/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.882 total
time=   0.2s
[CV 5/5] END .C=100, gamma=0.01, kernel=sigmoid;, score=0.886 total
time=   0.2s
[CV 1/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.909 total
time=   0.2s
[CV 2/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.913 total
time=   0.2s
[CV 3/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.896 total
time=   0.2s
```

```
[CV 4/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.882 total
time=   0.2s
[CV 5/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.869 total
time=   0.2s
[CV 1/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.903 total
time=   0.2s
[CV 2/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.879 total
time=   0.2s
[CV 3/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.852 total
time=   0.2s
[CV 4/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.875 total
time=   0.2s
[CV 5/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.872 total
time=   0.2s
[CV 1/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.879 total
time=   0.2s
[CV 2/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.846 total
time=   0.2s
[CV 3/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.839 total
time=   0.2s
[CV 4/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.852 total
time=   0.2s
[CV 5/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.852 total
time=   0.2s
[CV 1/5] END .C=100, gamma=0.001, kernel=linear;, score=0.909 total
time=   0.2s
[CV 2/5] END .C=100, gamma=0.001, kernel=linear;, score=0.913 total
time=   0.2s
[CV 3/5] END .C=100, gamma=0.001, kernel=linear;, score=0.896 total
time=   0.2s
[CV 4/5] END .C=100, gamma=0.001, kernel=linear;, score=0.882 total
time=   0.2s
[CV 5/5] END .C=100, gamma=0.001, kernel=linear;, score=0.869 total
time=   0.2s
[CV 1/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 2/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.2s
[CV 3/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.775 total
time=   0.3s
```

```
[CV 4/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.778 total
time=   0.2s
[CV 5/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.774 total
time=   0.2s
[CV 1/5] END ..C=100, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ..C=100, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.2s
[CV 3/5] END ..C=100, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.2s
[CV 4/5] END ..C=100, gamma=0.0001, kernel=poly;, score=0.778 total
time=   0.2s
[CV 5/5] END ..C=100, gamma=0.0001, kernel=poly;, score=0.774 total
time=   0.2s
[CV 1/5] END C=100, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 2/5] END C=100, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 3/5] END C=100, gamma=0.0001, kernel=sigmoid;, score=0.775 total
time=   0.2s
[CV 4/5] END C=100, gamma=0.0001, kernel=sigmoid;, score=0.778 total
time=   0.2s
[CV 5/5] END C=100, gamma=0.0001, kernel=sigmoid;, score=0.774 total
time=   0.2s
[CV 1/5] END C=100, gamma=0.0001, kernel=linear;, score=0.909 total
time=   0.2s
[CV 2/5] END C=100, gamma=0.0001, kernel=linear;, score=0.913 total
time=   0.2s
[CV 3/5] END C=100, gamma=0.0001, kernel=linear;, score=0.896 total
time=   0.2s
[CV 4/5] END C=100, gamma=0.0001, kernel=linear;, score=0.882 total
time=   0.2s
[CV 5/5] END C=100, gamma=0.0001, kernel=linear;, score=0.869 total
time=   0.2s
[CV 1/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.926 total
time=   0.2s
[CV 2/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.919 total
time=   0.2s
[CV 3/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.919 total
time=   0.2s
[CV 4/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.919 total
time=   0.2s
[CV 5/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.902 total
time=   0.2s
[CV 1/5] END ......C=1000, gamma=1, kernel=poly;, score=0.916 total
time=   1.7s
[CV 2/5] END ......C=1000, gamma=1, kernel=poly;, score=0.903 total
time=   1.3s
[CV 3/5] END ......C=1000, gamma=1, kernel=poly;, score=0.886 total
time=   1.3s
```

```
[CV 4/5] END ......C=1000, gamma=1, kernel=poly;, score=0.889 total
time=   1.3s
[CV 5/5] END ......C=1000, gamma=1, kernel=poly;, score=0.896 total
time=   1.5s
[CV 1/5] END ...C=1000, gamma=1, kernel=sigmoid;, score=0.732 total
time=   0.3s
[CV 2/5] END ...C=1000, gamma=1, kernel=sigmoid;, score=0.711 total
time=   0.2s
[CV 3/5] END ...C=1000, gamma=1, kernel=sigmoid;, score=0.718 total
time=   0.2s
[CV 4/5] END ...C=1000, gamma=1, kernel=sigmoid;, score=0.754 total
time=   0.2s
[CV 5/5] END ...C=1000, gamma=1, kernel=sigmoid;, score=0.653 total
time=   0.3s
[CV 1/5] END ....C=1000, gamma=1, kernel=linear;, score=0.919 total
time=   1.1s
[CV 2/5] END ....C=1000, gamma=1, kernel=linear;, score=0.916 total
time=   1.0s
[CV 3/5] END ....C=1000, gamma=1, kernel=linear;, score=0.906 total
time=   1.0s
[CV 4/5] END ....C=1000, gamma=1, kernel=linear;, score=0.886 total
time=   1.0s
[CV 5/5] END ....C=1000, gamma=1, kernel=linear;, score=0.879 total
time=   0.8s
[CV 1/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.936 total
time=   0.2s
[CV 2/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.926 total
time=   0.3s
[CV 3/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.899 total
time=   0.2s
[CV 4/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.912 total
time=   0.2s
[CV 5/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.896 total
time=   0.2s
[CV 1/5] END ....C=1000, gamma=0.1, kernel=poly;, score=0.940 total
time=   0.1s
[CV 2/5] END ....C=1000, gamma=0.1, kernel=poly;, score=0.923 total
time=   0.2s
[CV 3/5] END ....C=1000, gamma=0.1, kernel=poly;, score=0.913 total
time=   0.2s
[CV 4/5] END ....C=1000, gamma=0.1, kernel=poly;, score=0.906 total
time=   0.1s
[CV 5/5] END ....C=1000, gamma=0.1, kernel=poly;, score=0.909 total
time=   0.1s
[CV 1/5] END .C=1000, gamma=0.1, kernel=sigmoid;, score=0.862 total
time=   0.2s
[CV 2/5] END .C=1000, gamma=0.1, kernel=sigmoid;, score=0.866 total
time=   0.2s
[CV 3/5] END .C=1000, gamma=0.1, kernel=sigmoid;, score=0.842 total
time=   0.2s
```

```
[CV 4/5] END .C=1000, gamma=0.1, kernel=sigmoid;, score=0.842 total
time=   0.2s
[CV 5/5] END .C=1000, gamma=0.1, kernel=sigmoid;, score=0.838 total
time=   0.2s
[CV 1/5] END ..C=1000, gamma=0.1, kernel=linear;, score=0.919 total
time=   1.1s
[CV 2/5] END ..C=1000, gamma=0.1, kernel=linear;, score=0.916 total
time=   1.1s
[CV 3/5] END ..C=1000, gamma=0.1, kernel=linear;, score=0.906 total
time=   0.9s
[CV 4/5] END ..C=1000, gamma=0.1, kernel=linear;, score=0.886 total
time=   1.1s
[CV 5/5] END ..C=1000, gamma=0.1, kernel=linear;, score=0.879 total
time=   0.7s
[CV 1/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.916 total
time=   0.2s
[CV 2/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.926 total
time=   0.2s
[CV 3/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.909 total
time=   0.2s
[CV 4/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.892 total
time=   0.2s
[CV 5/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.892 total
time=   0.2s
[CV 1/5] END ...C=1000, gamma=0.01, kernel=poly;, score=0.795 total
time=   0.2s
[CV 2/5] END ...C=1000, gamma=0.01, kernel=poly;, score=0.795 total
time=   0.1s
[CV 3/5] END ...C=1000, gamma=0.01, kernel=poly;, score=0.799 total
time=   0.2s
[CV 4/5] END ...C=1000, gamma=0.01, kernel=poly;, score=0.805 total
time=   0.1s
[CV 5/5] END ...C=1000, gamma=0.01, kernel=poly;, score=0.788 total
time=   0.1s
[CV 1/5] END C=1000, gamma=0.01, kernel=sigmoid;, score=0.913 total
time=   0.2s
[CV 2/5] END C=1000, gamma=0.01, kernel=sigmoid;, score=0.916 total
time=   0.2s
[CV 3/5] END C=1000, gamma=0.01, kernel=sigmoid;, score=0.899 total
time=   0.2s
[CV 4/5] END C=1000, gamma=0.01, kernel=sigmoid;, score=0.875 total
time=   0.2s
[CV 5/5] END C=1000, gamma=0.01, kernel=sigmoid;, score=0.889 total
time=   0.2s
[CV 1/5] END .C=1000, gamma=0.01, kernel=linear;, score=0.919 total
time=   1.0s
[CV 2/5] END .C=1000, gamma=0.01, kernel=linear;, score=0.916 total
time=   1.1s
[CV 3/5] END .C=1000, gamma=0.01, kernel=linear;, score=0.906 total
time=   0.9s
```

```
[CV 4/5] END .C=1000, gamma=0.01, kernel=linear;, score=0.886 total
time=   0.9s
[CV 5/5] END .C=1000, gamma=0.01, kernel=linear;, score=0.879 total
time=   0.8s
[CV 1/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.913 total
time=   0.2s
[CV 2/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.933 total
time=   0.2s
[CV 3/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.896 total
time=   0.2s
[CV 4/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.882 total
time=   0.2s
[CV 5/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.886 total
time=   0.2s
[CV 1/5] END ..C=1000, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END ..C=1000, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END ..C=1000, gamma=0.001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END ..C=1000, gamma=0.001, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END ..C=1000, gamma=0.001, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END C=1000, gamma=0.001, kernel=sigmoid;, score=0.916 total
time=   0.2s
[CV 2/5] END C=1000, gamma=0.001, kernel=sigmoid;, score=0.919 total
time=   0.2s
[CV 3/5] END C=1000, gamma=0.001, kernel=sigmoid;, score=0.879 total
time=   0.2s
[CV 4/5] END C=1000, gamma=0.001, kernel=sigmoid;, score=0.882 total
time=   0.2s
[CV 5/5] END C=1000, gamma=0.001, kernel=sigmoid;, score=0.886 total
time=   0.2s
[CV 1/5] END C=1000, gamma=0.001, kernel=linear;, score=0.919 total
time=   1.3s
[CV 2/5] END C=1000, gamma=0.001, kernel=linear;, score=0.916 total
time=   0.9s
[CV 3/5] END C=1000, gamma=0.001, kernel=linear;, score=0.906 total
time=   0.9s
[CV 4/5] END C=1000, gamma=0.001, kernel=linear;, score=0.886 total
time=   0.9s
[CV 5/5] END C=1000, gamma=0.001, kernel=linear;, score=0.879 total
time=   0.9s
[CV 1/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.903 total
time=   0.2s
[CV 2/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.879 total
time=   0.2s
[CV 3/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.852 total
time=   0.2s
```

```
[CV 4/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.875 total
time=   0.2s
[CV 5/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.872 total
time=   0.2s
[CV 1/5] END .C=1000, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 2/5] END .C=1000, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 3/5] END .C=1000, gamma=0.0001, kernel=poly;, score=0.775 total
time=   0.1s
[CV 4/5] END .C=1000, gamma=0.0001, kernel=poly;, score=0.778 total
time=   0.1s
[CV 5/5] END .C=1000, gamma=0.0001, kernel=poly;, score=0.774 total
time=   0.1s
[CV 1/5] END C=1000, gamma=0.0001, kernel=sigmoid;, score=0.879 total
time=   0.2s
[CV 2/5] END C=1000, gamma=0.0001, kernel=sigmoid;, score=0.849 total
time=   0.2s
[CV 3/5] END C=1000, gamma=0.0001, kernel=sigmoid;, score=0.839 total
time=   0.2s
[CV 4/5] END C=1000, gamma=0.0001, kernel=sigmoid;, score=0.852 total
time=   0.2s
[CV 5/5] END C=1000, gamma=0.0001, kernel=sigmoid;, score=0.852 total
time=   0.2s
[CV 1/5] END C=1000, gamma=0.0001, kernel=linear;, score=0.919 total
time=   1.1s
[CV 2/5] END C=1000, gamma=0.0001, kernel=linear;, score=0.916 total
time=   1.0s
[CV 3/5] END C=1000, gamma=0.0001, kernel=linear;, score=0.906 total
time=   1.1s
[CV 4/5] END C=1000, gamma=0.0001, kernel=linear;, score=0.886 total
time=   1.4s
[CV 5/5] END C=1000, gamma=0.0001, kernel=linear;, score=0.879 total
time=   0.9s
```

```
GridSearchCV(estimator=SVC(probability=True),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf', 'poly', 'sigmoid',
'linear']},
             verbose=3)
```

```
clf_svc.best_params_
```

```
{'C': 100, 'gamma': 1, 'kernel': 'rbf'}
```

```
svc_model_optimized = SVC(probability=True, C= 100, gamma= 1, kernel=
'rbf')
svc_model_optimized.fit(X_train, y_train)
```

```
SVC(C=100, gamma=1, probability=True)
```

```
y_predictions_svc_optimized = svc_model_optimized.predict(X_test)
```

Classification Metrics for Optimized SVC model

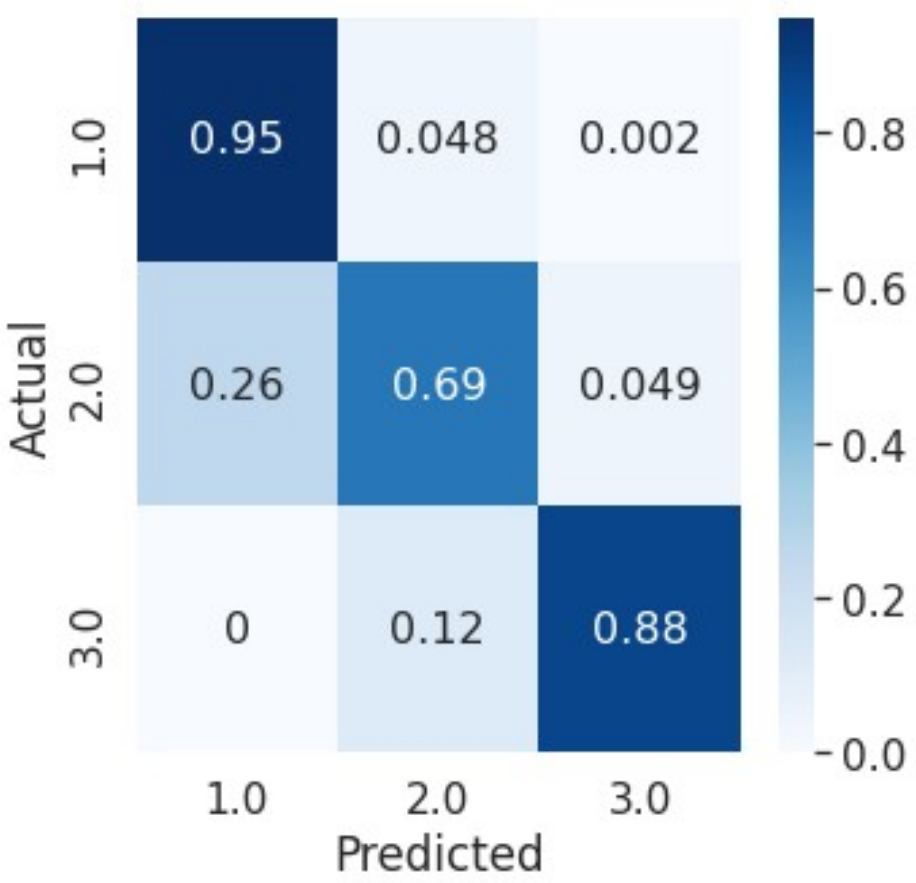
```
accuracy_score(y_test,y_predictions_svc_optimized)
```

```
0.9106583072100314
```

```
print(classification_report(y_test,y_predictions_svc_optimized))
```

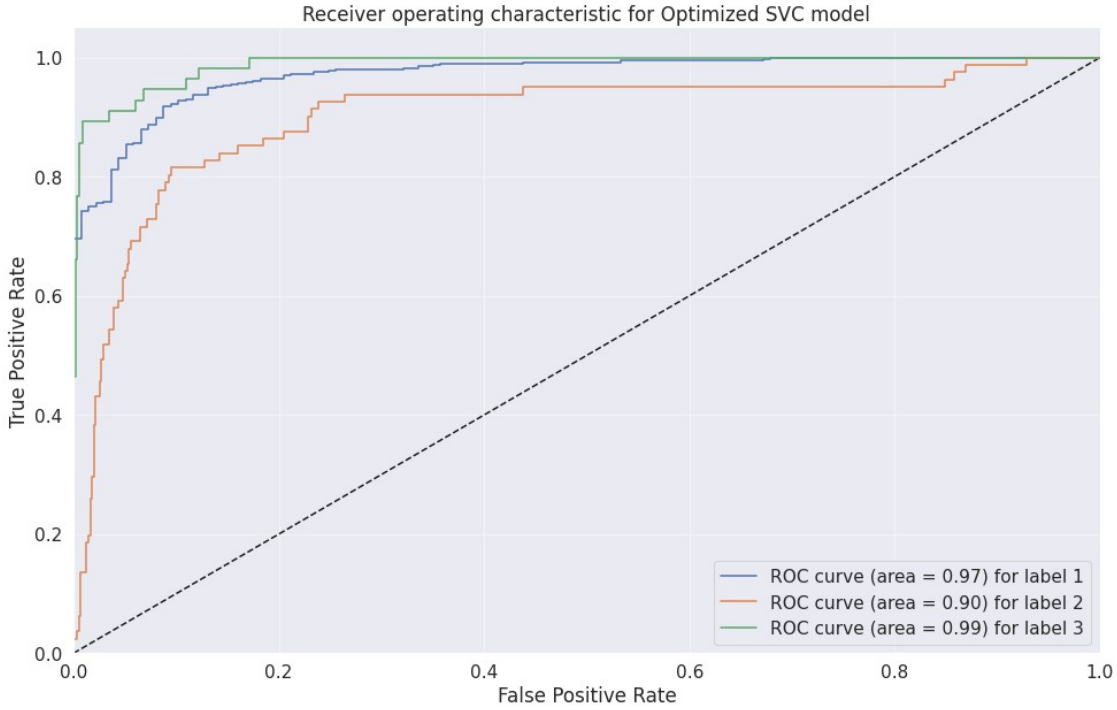
```
              precision    recall  f1-score   support

         1.0       0.96      0.95      0.95       501
         2.0       0.64      0.69      0.67        81
         3.0       0.91      0.88      0.89        56

    accuracy                           0.91       638
   macro avg       0.84      0.84      0.84       638
weighted avg       0.91      0.91      0.91       638
```

```
data = confusion_matrix(y_test, y_predictions_svc_optimized, normalize
= 'true')
df_cm = pd.DataFrame(data,
columns=np.unique(y_predictions_svc_optimized), index =
np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (5,5))
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```

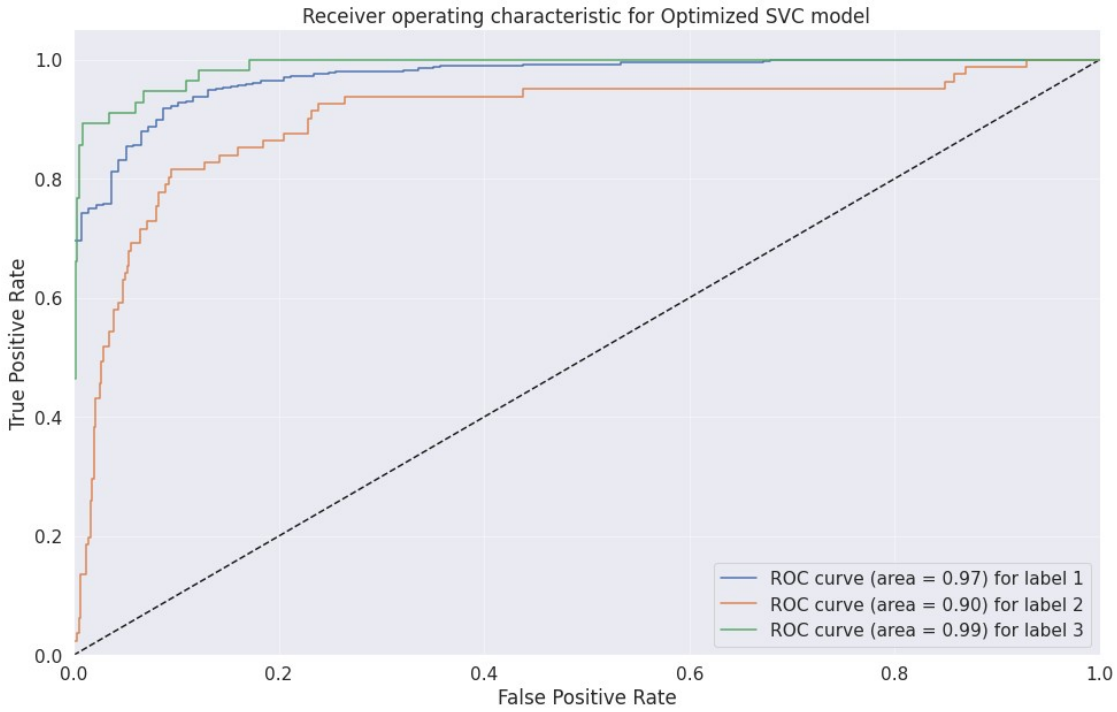



```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

def plot_multiclass_roc(clf, X_test, y_test, n_classes, figsize=(17,
6)):
    try:
        y_score = clf.decision_function(X_test)
    except:
        y_score = clf.pred_proba(X_test)
        print("Using decision function method")

    # structures
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # calculate dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:,
i])
        roc_auc[i] = auc(fpr[i], tpr[i])
```

```
    # roc for each class
    fig, ax = plt.subplots(figsize=figsize)
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title('Receiver operating characteristic for Optimized SVC
model')
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for
label %i' % (roc_auc[i], i+1))
    ax.legend(loc="best")
    ax.grid(alpha=.4)
    sns.despine()
    plt.show()
    fig = ax.get_figure()
    return fig, ax

svc_model_optimized_roc_auc_curve =
plot_multiclass_roc(svc_model_optimized, X_test, y_test, n_classes=3,
figsize=(16, 10))
```




```
svc_model_optimized_roc_auc_curve[0]
```



Naive Bayes Model

Training Naive Bayes model

```
# Since the features of our data are continous, the right choice is
GaussianNB
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score

gnb_model = GaussianNB()
scores = cross_val_score(gnb_model, X_train, y_train, n_jobs=4, cv=4,
scoring = "accuracy")
print(scores)
print(np.mean(scores))
```

```
[0.75       0.69354839 0.74462366 0.8172043 ]
0.7513440860215055
```

```
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB


nb = {'gaussian': GaussianNB(),
      'bernoulli': BernoulliNB(),
      'multinomial': MultinomialNB()}

scores = {}
for key, model in nb.items():
```

```
    s = cross_val_score(model, X_train, y_train, cv=4, n_jobs=4,
scoring='accuracy')
    scores[key] = np.mean(s)
scores
```

```
{'bernoulli': 0.8346774193548387,
 'gaussian': 0.7513440860215055,
 'multinomial': 0.801747311827957}
```

This is interesting because bernoulli model usually works with binary data and the multinomial model is usuall the best with continous data.

Now, let's discretize the dataset. There are many ways to do this, but we'll use `pd.DataFrame.rank(pct=True)`.

- Create `X_discrete` from `X` using .rank(pct=True)
- Look at the values. They are still not discrete. Modify `X_discrete` so that it is indeed discrete. (Hint: try to get the first 2 digits using `.applymap`)
- Split `X_discrete` and `y` into training and test datasets
- Fit a MultinomialNB to the training split.
- Get predictions on the test set.
- Plot the confusion matrix for predictions.

```
X_train_discrete = X_train.rank(pct=True)
X_test_discrete = X_test.rank(pct=True)
```

```
X_train_discrete.sample(5)
```

```
      baseline value  accelerations  ...  histogram_variance
histogram_tendency
2088        0.484543       0.205981  ...            0.046707
0.350134
1364        0.435148       0.520497  ...            0.776882
0.810148
1730        0.532594       0.906922  ...            0.928763
0.810148
254         0.361223       0.205981  ...            0.515121
0.040323
1253        0.068212       0.868952  ...            0.515121
0.350134

[5 rows x 21 columns]
```

```
X_train_discrete = X_train_discrete.applymap(lambda r: int(r*100))
X_test_discrete = X_test_discrete.applymap(lambda r: int(r*100))
X_train_discrete.sample(5)
```

```
      baseline value  accelerations  ...  histogram_variance
histogram_tendency
2039              28             20  ...                  76
35
```

```
556               71             60  ...                  76
81
1603              66             72  ...                  62
35
907               90             72  ...                  48
81
397               93             20  ...                  14
81

[5 rows x 21 columns]
```

```
mnb_model = MultinomialNB()
mnb_model.fit(X_train_discrete, y_train)
```

```
MultinomialNB()
```

```
y_pred_mnb = pd.Series(mnb_model.predict(X_test))
y_pred_mnb
```

```
0      1.0
1      1.0
2      2.0
3      1.0
4      1.0
      ...
633    2.0
634    1.0
635    1.0
636    1.0
637    1.0
Length: 638, dtype: float64
```

```
y_test.value_counts()
```

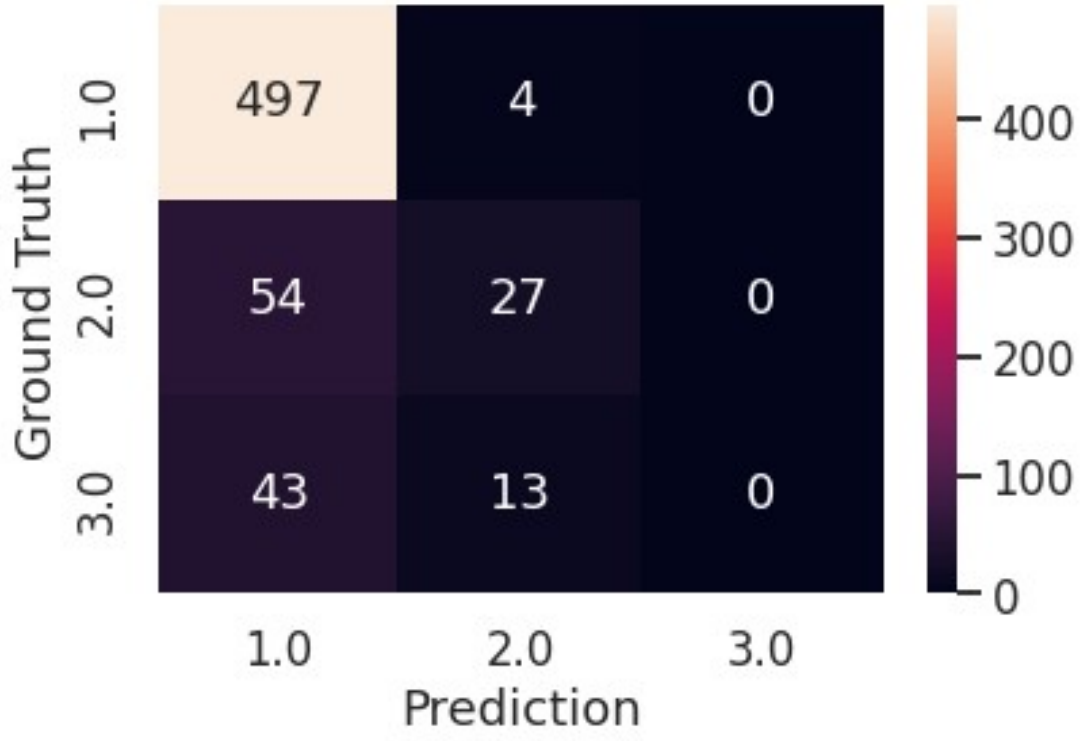
```
1.0    501
2.0     81
3.0     56
Name: fetal_health, dtype: int64
```

```
y_pred_mnb.value_counts()
```

```
1.0    594
2.0     44
dtype: int64
```

```
from sklearn.metrics import confusion_matrix

labels = sorted(y_test.unique())

cm = pd.DataFrame(confusion_matrix(y_test, y_pred_mnb, labels=labels),
                  columns=labels, index=labels)
sns.set_context('talk')
```

```
ax = sns.heatmap(cm, annot=True,fmt ="d", xticklabels=True,
yticklabels=True)
ax.set_xticklabels(labels)
ax.set_yticklabels(labels)
ax.set_ylabel('Ground Truth')
ax.set_xlabel('Prediction');
```



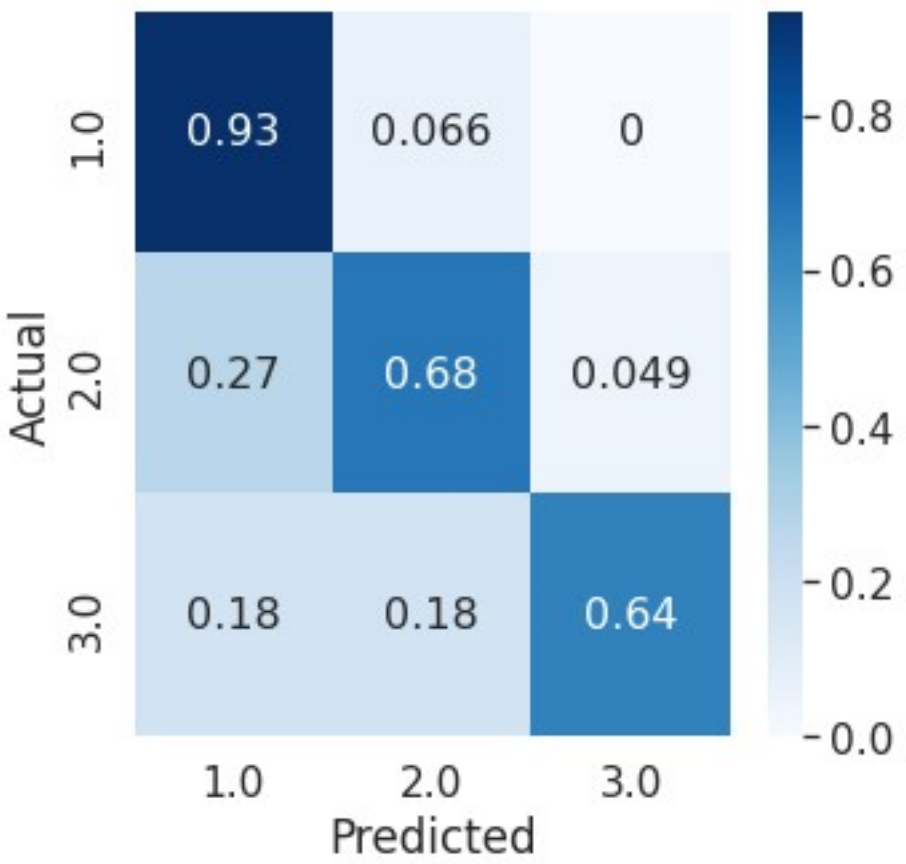
Using MNB with discretized values resulted in a `really bad confusion matrix`. The model has a `100% False Negative result in predicting class 3 (pathological)`. This model **absolutely** cannot be used in production since `babies that DO have problems` will not be detected and the algorithm will output that the babies are `falsely healthy`. It performs badly in predicting class 2 (suspect), but performs well in predicting class 1 (healthy), which may be due to the imbalanced dataset. This model does not work well with imbalanced datasets.

Conclusion:

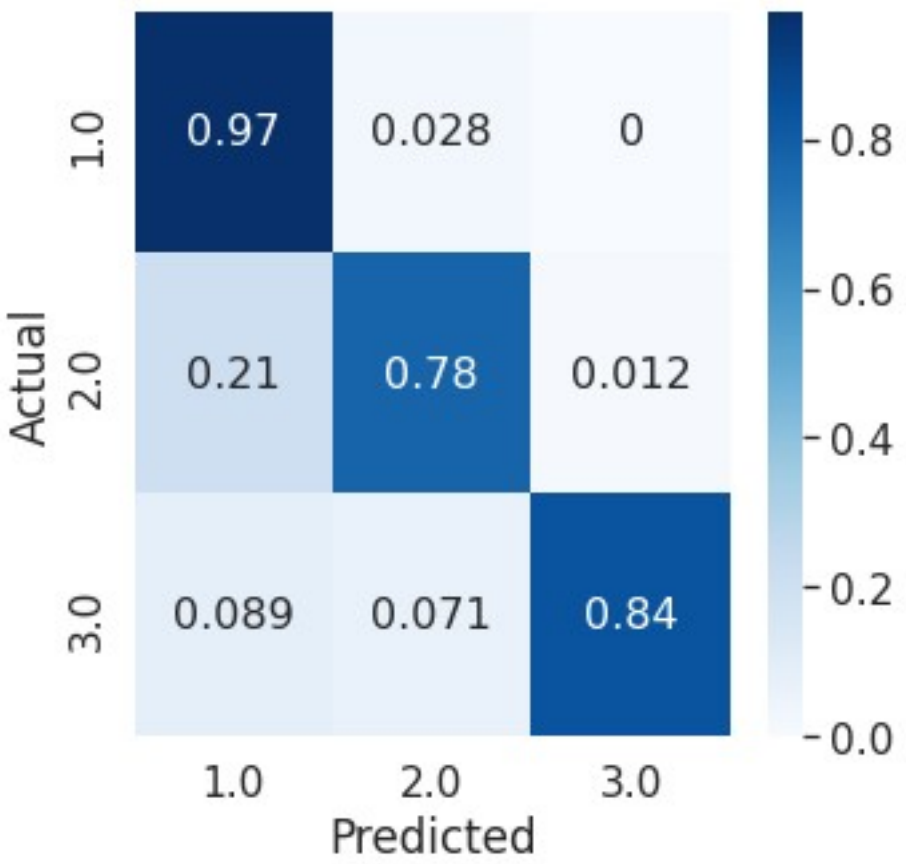
I think one of the best ways we can determine which classification model is the best for this particular problem is by looking at their their confusion matrix:

Below are the models' respective confusion matrix. Note that these figures represent the optimized version of each model:

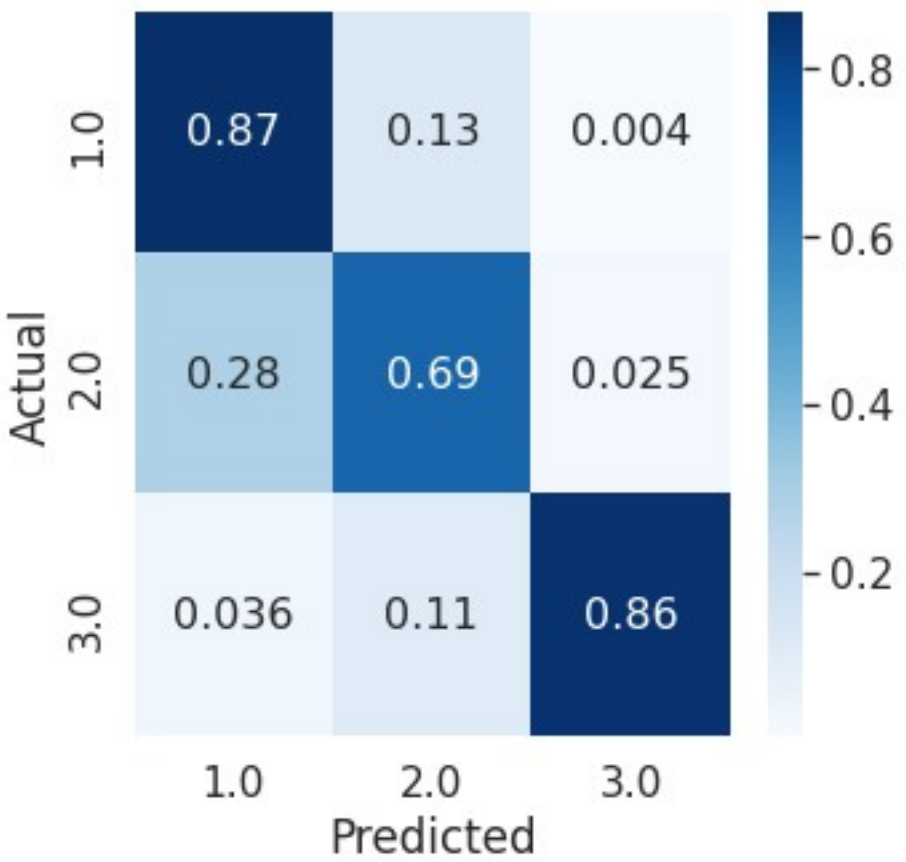
Logistic Regression



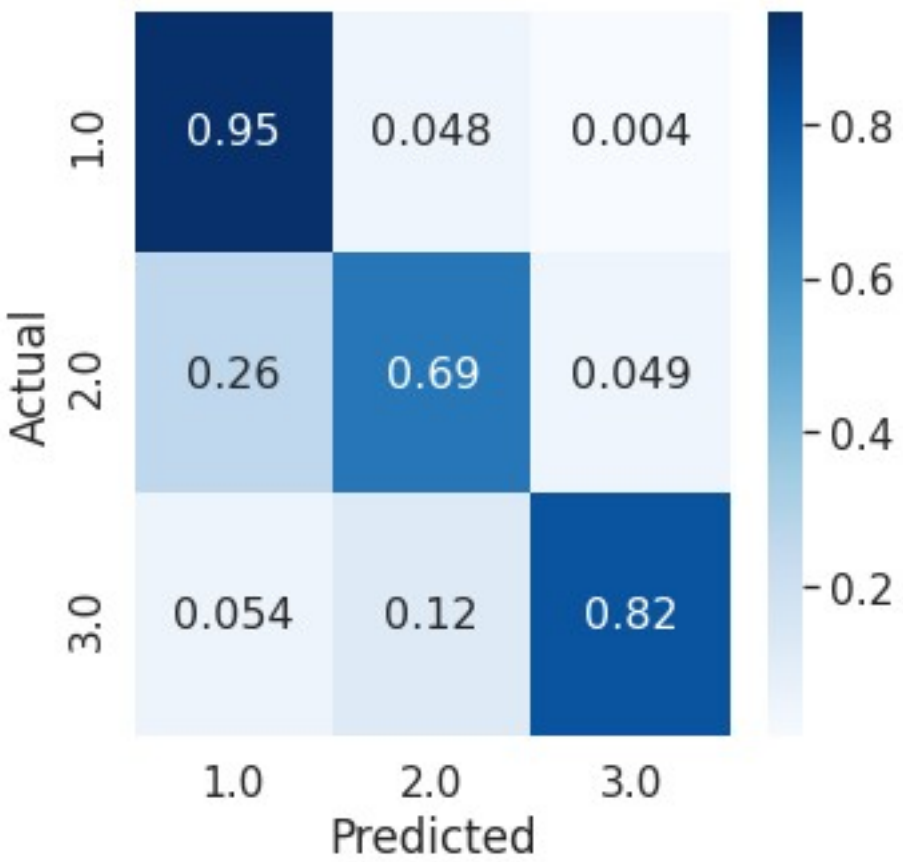
Random Forest



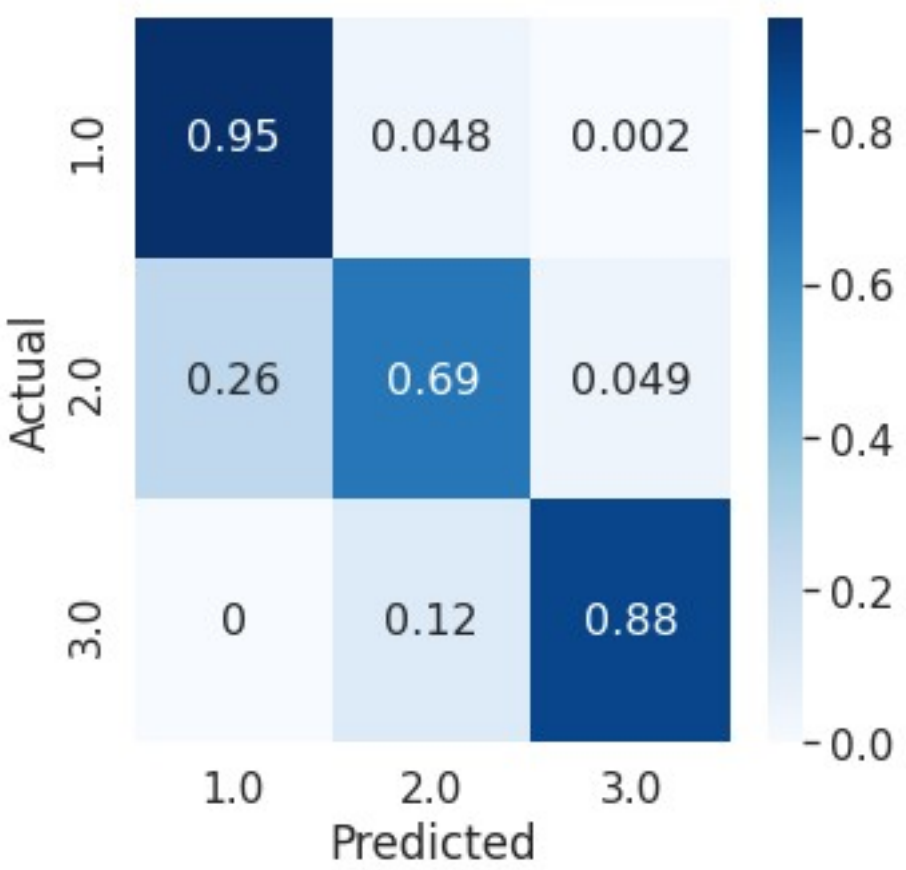
Decision Tree


0.87
0.13
0.004
0.28
0.69
0.025
0.036
0.11
0.86
1.0
2.0
3.0
Actual
1.0
2.0
3.0
Predicted
0.8
0.6
0.4
0.2

KNN


0.95
0.048
0.004
0.26
0.69
0.049
0.054
0.12
0.82
1.0
2.0
3.0
Actual
1.0
2.0
3.0
Predicted
0.8
0.6
0.4
0.2

svc



Based on these confusion matrix, I think on the average the random forest model would be favorable in terms of having greater true positives across classes 1 (normal) and 2 (suspect), and only 2nd best in finding true positives for class 3 (pathological).

I also think that the Random Forest classifier performs well in imbalanced data sets like we have here. This means we can rely on it when it is put in production.

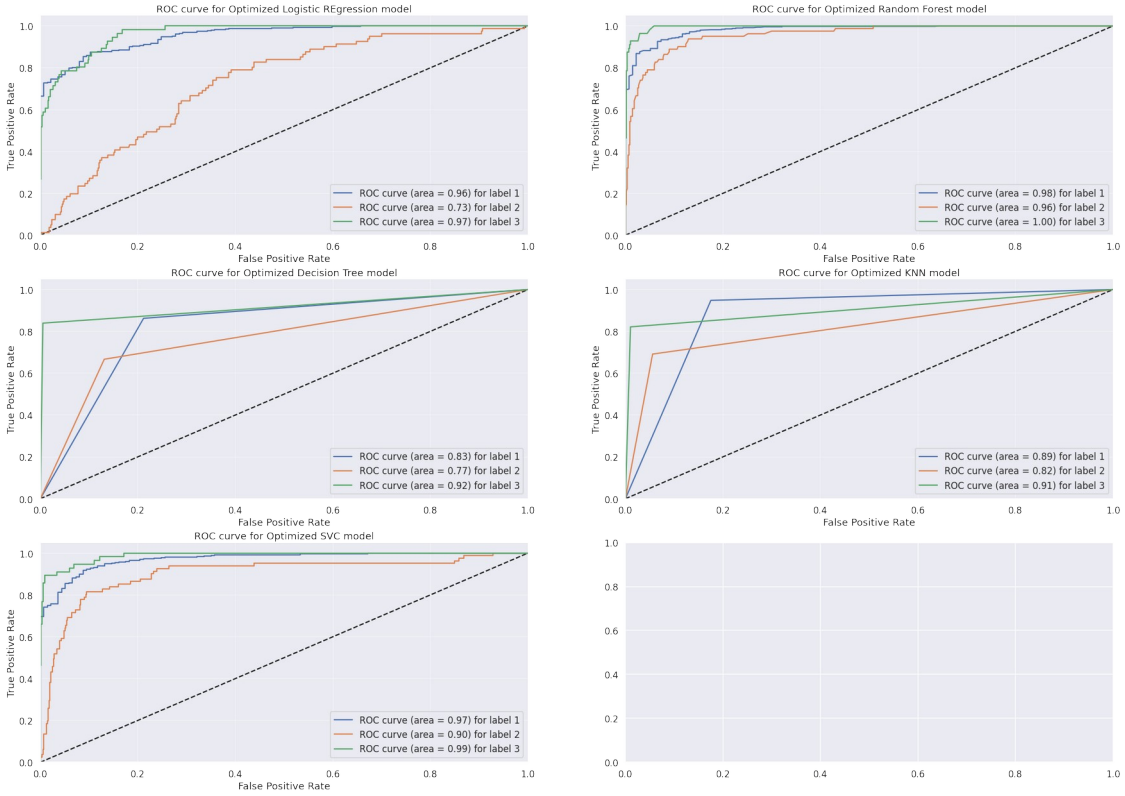
Let us check the ROC_AUC metrics for all the optimized models

```
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

def plot_multiclass_roc(clf, X_test, y_test, n_classes, ax, title):
    try:
        y_score = clf.decision_function(X_test)
        print("Using decision_function method")
    except:
        y_score = clf.predict_proba(X_test)
        print("Using predict_proba method")

    # structures
```

```
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    # calculate dummies once
    y_test_dummies = pd.get_dummies(y_test, drop_first=False).values
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_dummies[:, i], y_score[:,
i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # roc for each class
    ax.plot([0, 1], [0, 1], 'k--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    model = title[0]
    title.pop(0)
    ax.set_title('ROC curve for Optimized ' + model + ' model')
    for i in range(n_classes):
        ax.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f) for
label %i' % (roc_auc[i], i+1))
    ax.legend(loc="best")
    ax.grid(alpha=.4)
    sns.despine()

fig, ax = plt.subplots(nrows = 3,
                       ncols = 2,
                       figsize= (35, 25))
#plot to each different index. Call the ax created in a given (row,
column)
title= ["Logistic REgression", "Random Forest", "Decision Tree","KNN",
"SVC", "Naive Bayes"]
plot_multiclass_roc(best_clf, X_test, y_test, n_classes=3, ax=ax[0,0],
title=title)
plot_multiclass_roc(RandomForestModel_optimized, X_test, y_test,
n_classes=3, ax=ax[0,1], title=title)
plot_multiclass_roc(decisiontree_model_optimized, X_test, y_test,
n_classes=3, ax=ax[1,0],title=title)
plot_multiclass_roc(best_clf_knn, X_test, y_test, n_classes=3,
ax=ax[1,1], title=title)
plot_multiclass_roc(svc_model_optimized, X_test, y_test, n_classes=3,
ax=ax[2,0], title=title)
plt.show()
```

```
Using decision_function method
Using predict_proba method
Using predict_proba method
```

```
Using predict_proba method
Using decision_function method
```



The optimized Random Forest model also has the best ROC curve for all the 3 classes which means that it is ideal since the ROC curve is a good indicator of the performance of the classifier. This means the rate at which the model is able to predict True Positives (Sensitivity) more accurately than False Positives (1 - Specificity).

True Postive Rate (Sensitivity) = True Positive Rate (TPR) = TP / (TP + FN)

- tells you what proportion of the positive cases are correctly identified as positive (True Positives) by the model by comparing True Positives to the Total number of Ground Truth Positives.

False Positive Rate (1 - Specificity) = 1 - True Negative Rate (TNR) = FP / (FP + TN)

- tells you what proportion of the negative cases are incorrectly identified as positive (False Positives) by the model by comapring the False Positves to the actual number of negatives in the test set.

The SVC model comes in second place when we are using the ROC_AUC curve and score as the metric

However, it is adviced that for imbalanced datasets that we replace the False Postive Rate (1-specificity) with `Precision or (P) = TP/(TP + FP)` that compares the positves we

correctly classified vs the positves that were correctly classified in addtion to the ones we labeled falsely positive.

I did not make any further hypertuning or classification metrics for Naive Bayes since I can infer that it will be the least useful model.