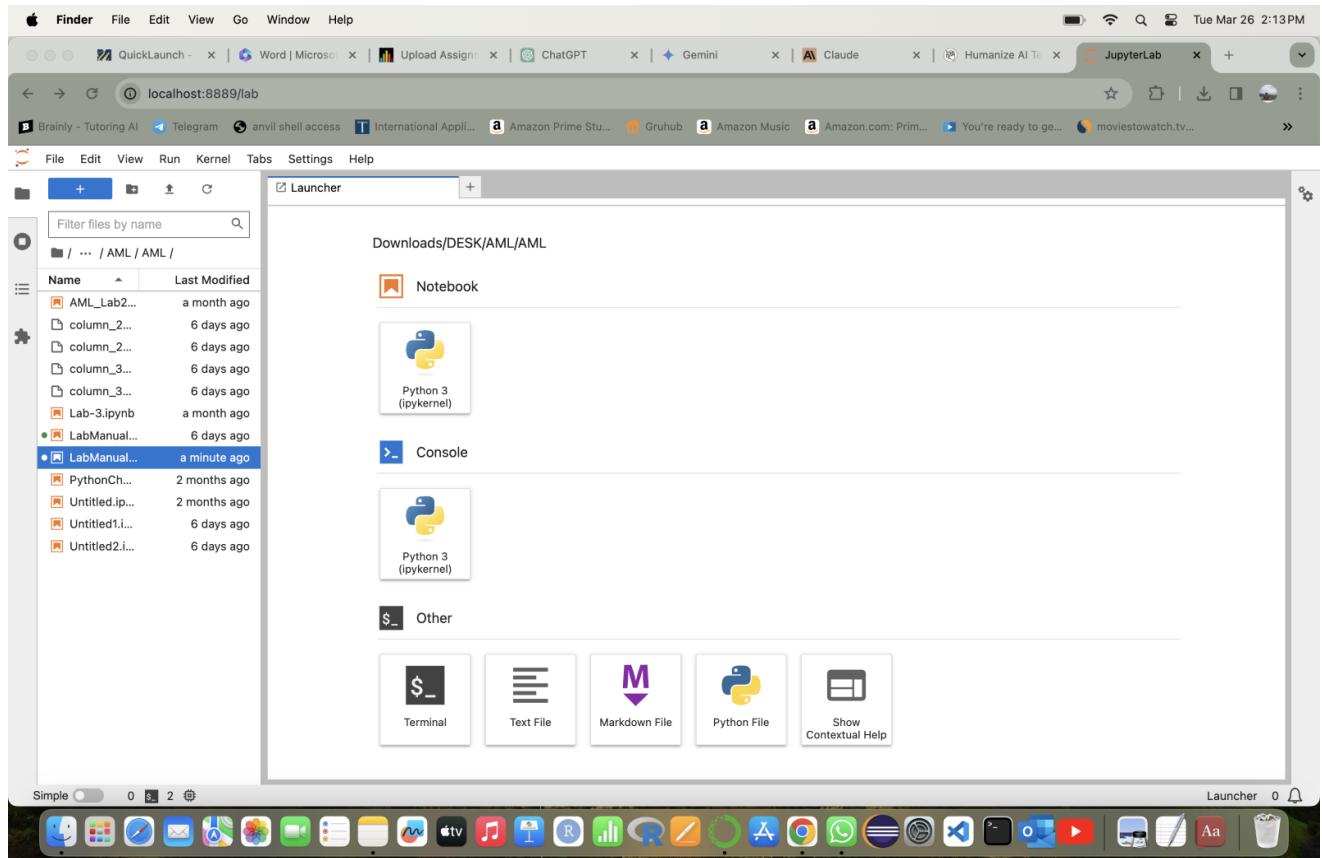


Jeevan Kumar Banoth - 02105145

Advanced machine learning

Homework – 6:

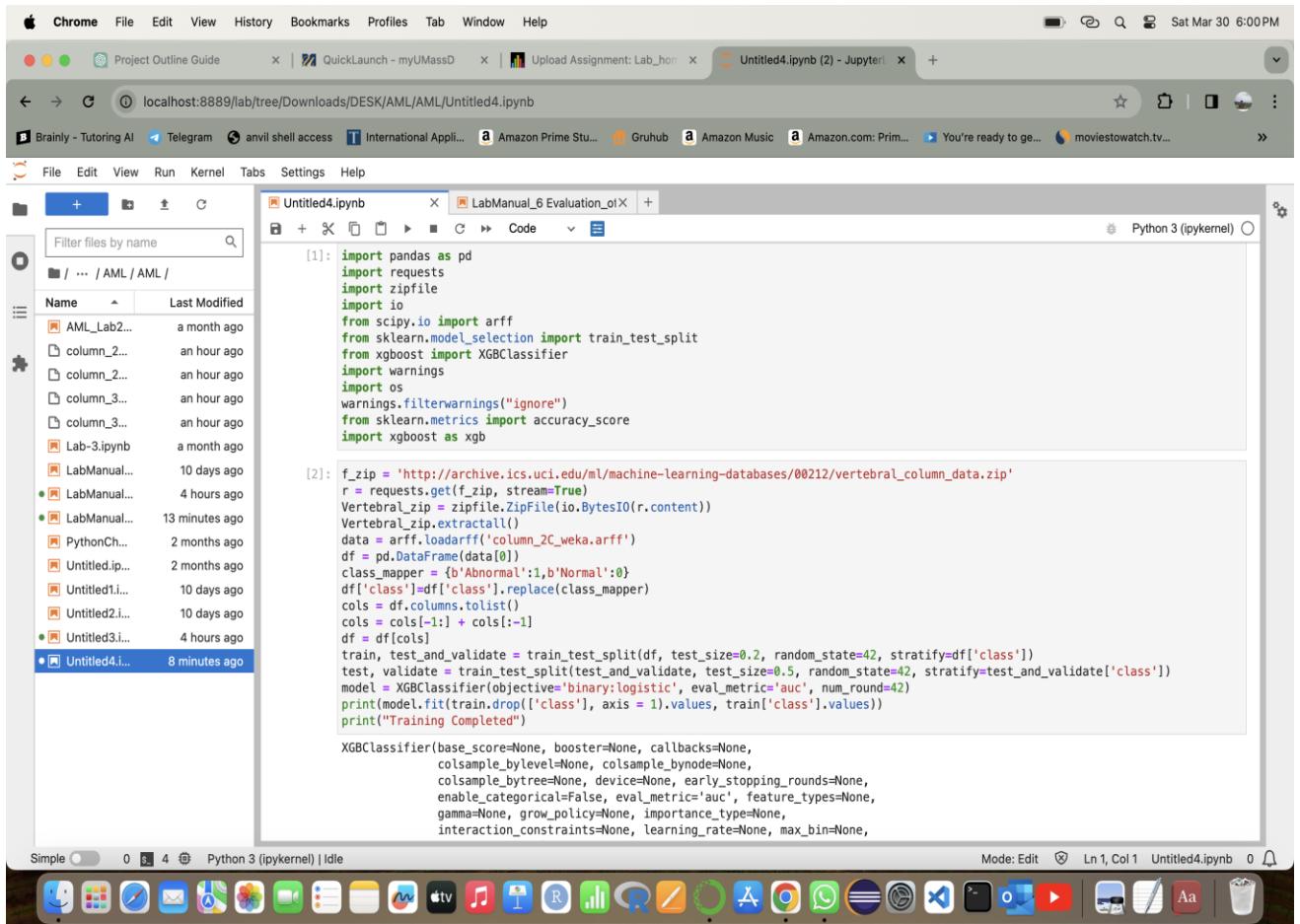
➤ Creating Jupyter Notebook with Anaconda navigator:



- By using Anaconda navigator, first I opened Jupyter lab, and redirected it to my working folder.
- In the left side of the notebook, we can see that there are files on which we have previously worked.
- Created a notebook for this task by selecting File > New, Notebook, and then conda_python3 in the kernel dialog window.
- But, for this lab there is no need to work on the new notebook so we will start it by uploading.

- After opening the document, I created a new notebook by using python3 kernel and started doing the lab.
- So, the first part of the lab is all same as the previous one, this lab is the continuation of that lab so we must perform the previous part and continue with this one.
- Below are the Images of the code which has previous lab part.

Note: As I have already explained the code in the previous lab, I am not specially explaining it now. I am just attaching those pictures for the proof that I performed the lab.



The screenshot shows a Jupyter Notebook interface running in a Chrome browser. The notebook is titled 'Untitled4.ipynb' and contains two code cells. Cell [1] imports various Python libraries including pandas, requests, zipfile, io, arff, train_test_split, XGBClassifier, warnings, and accuracy_score. It also defines a class_mapper for handling categorical data. Cell [2] downloads a dataset from UCI's archive, extracts it, loads it into a DataFrame, and performs stratified splitting into training and testing sets. An XGBClassifier is trained on the data, and the training process is printed to the console. The browser's address bar shows the URL is 'localhost:8889/lab/tree/Downloads/DESK/AML/AML/Untitled4.ipynb'. The status bar at the bottom indicates the code is run in 'Python 3 (ipykernel)'.

```
[1]: import pandas as pd
import requests
import zipfile
import io
from scipy.io import arff
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
import warnings
import os
warnings.filterwarnings("ignore")
from sklearn.metrics import accuracy_score
import xgboost as xgb

[2]: f_zip = 'http://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebral_column_data.zip'
r = requests.get(f_zip, stream=True)
Vertebral_zip = zipfile.ZipFile(io.BytesIO(r.content))
Vertebral_zip.extractall()
data = arff.loadarff('column_2C_weka.arff')
df = pd.DataFrame(data[0])
class_mapper = {'Abnormal':1,'Normal':0}
df['class']=df['class'].replace(class_mapper)
cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]
train, test_and_validate = train_test_split(df, test_size=0.2, random_state=42, stratify=df['class'])
test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['class'])
model = XGBClassifier(objective='binary:logistic', eval_metric='auc', num_round=42)
print(model.fit(train.drop(['class'], axis = 1).values, train['class'].values))
print("Training Completed")

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='auc', feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
```

```

File Edit View Run Kernel Tabs Settings Help
Untitled4.ipynb LabManual_6 Evaluation_oI Code Python 3 (ipykernel)
[3]: test.shape
[3]: (31, 7)
[4]: row = test.iloc[0:1,:]
row.head()
[4]: pelvic_incidence  pelvic_tilt  lumbar_lordosis_angle  sacral_slope  pelvic_radius  degree_spondylolisthesis
136    88.024499  39.844669    81.774473   48.17983  116.601538      56.766083
[5]: model.predict_proba(row)
[5]: array([[0.00177544, 0.99822456]], dtype=float32)
[6]: batch_X = test.iloc[:,1:];
batch_X.head()
[6]: pelvic_incidence  pelvic_tilt  lumbar_lordosis_angle  sacral_slope  pelvic_radius  degree_spondylolisthesis
136    88.024499  39.844669    81.774473   48.17983  116.601538      56.766083
230    65.611802  23.137919    62.582179   42.473883  124.128001     -4.083298
134    52.204693  17.212673    78.094969   34.992020  136.972517      54.939134
130    50.066786  9.120340    32.168463   40.946446  99.712453      26.766697
47     41.352504  16.577364    30.706191   24.775141  113.266675     -4.497958
[7]: predicted_probabilities = model.predict_proba(batch_X)

```

```

File Edit View Run Kernel Tabs Settings Help
Untitled4.ipynb LabManual_6 Evaluation_oI Code Python 3 (ipykernel)
[7]: predicted_probabilities = model.predict_proba(batch_X)
[8]: target_predicted = pd.DataFrame(predicted_probabilities[:, 1], columns=['class'])
target_predicted.head(5)
[8]: class
0 0.998225
1 0.668622
2 0.995486
3 0.998336
4 0.961274
[9]: def binary_convert(x):
threshold = 0.5
if x > threshold:
return 1
else:
return 0
target_predicted_binary = target_predicted['class'].apply(binary_convert)
print(target_predicted_binary.head(5))
test.head(5)
[9]: class  pelvic_incidence  pelvic_tilt  lumbar_lordosis_angle  sacral_slope  pelvic_radius  degree_spondylolisthesis
0 1
1 1
2 1
3 1
4 1
Name: class, dtype: int64

```

Step 1: Creating confusion matrix:

The screenshot shows a Jupyter Notebook interface running in a Chrome browser. The notebook is titled 'Untitled4.ipynb' and contains the following code:

```

Step 1: Creating a confusion matrix

[11]: test_labels = test.iloc[:, 0]
test_labels.head(5)

[11]:
136    1
230    0
134    1
130    1
47     1
Name: class, dtype: int64

[12]: from sklearn.metrics import confusion_matrix
matrix = confusion_matrix(test_labels, target_predicted_binary)
df_confusion = pd.DataFrame(matrix, index=['Normal', 'Abnormal'], columns=['Normal', 'Abnormal'])

df_confusion

[12]:
      Normal  Abnormal
Normal      7         3
Abnormal    2        19

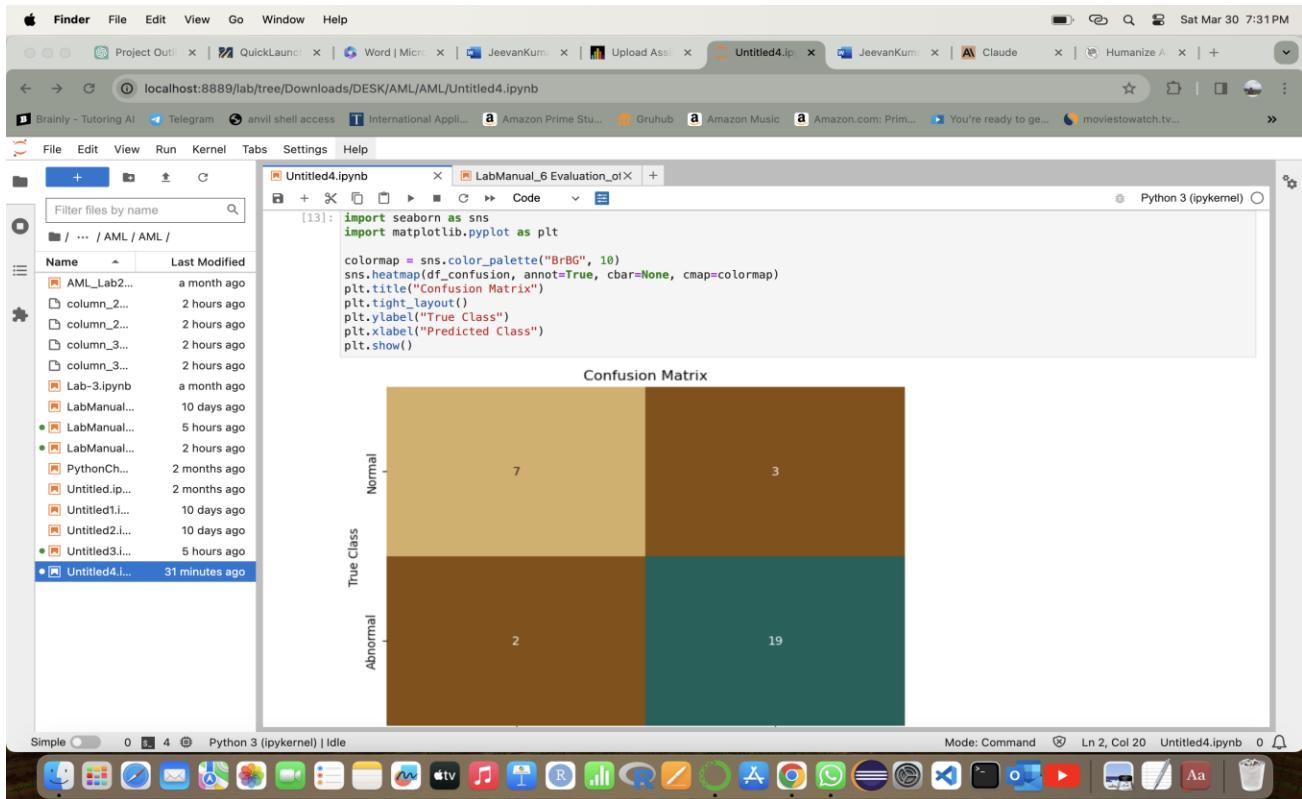
[13]: import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="whitegrid")

```

The code imports necessary libraries, retrieves test labels, prints the first five labels, calculates a confusion matrix, and prints the resulting DataFrame. The confusion matrix table is as follows:

	Normal	Abnormal
Normal	7	3
Abnormal	2	19

- **test_labels = test.iloc[:, 0]:** Extracting Actual Labels: - The first line retrieves the true labels from the test data, assuming they're in column 0 of the test DataFrame.
- **test_labels.head(5):** Printing First Five Labels: - The second line displays the initial five entries of the label series for easy reference.
- **from sklearn.metrics import confusion_matrix:** Importing Confusion Matrix Function: - The third line imports the function that evaluates the confusion matrix from the Python library.
- **Calculating Confusion Matrix:** - The final line employs the imported function to create a confusion matrix, evaluating the results against the real labels and projected binary labels.
- A NumPy array called "matrix" is used to make a Pandas DataFrame called "df_confusion."
- The rows and columns of the DataFrame have labels like "Normal" and "Abnormal," which stand for the classes in your classification task. The "df_confusion" DataFrame is then printed to the console, showing a table of the confusion matrix.



- **import seaborn as sns and import matplotlib.pyplot as plt:** To display data visually, this code uses Seaborn and Matplotlib libraries.
- **colormap = sns.color_palette("BrBG", 10):** A color palette called "colormap" is created using the "BrBG" colormap from Seaborn, with 10 different colors.
- Finally, a heatmap is generated using Seaborn's "heatmap" function, with the confusion matrix (df_confusion) as the data.
- The values within the confusion matrix are displayed on the cells (annot=True), the color bar is hidden (cbar=None), and the "colormap" palette is used for coloring (cmap=colormap).
- **plt.title("Confusion Matrix"):** Set the plot's title to "Confusion Matrix."
- **"plt.tight_layout()":** Optimize the space around plot elements to avoid clipping labels or titles.
- **"plt.ylabel("True Class")":** Label the y-axis with "True Class."
- **"plt.xlabel("Predicted Class")":** Label the x-axis with "Predicted Class."
- **plt.show()** - This line displays the plot in a new window.

```

Step: 2 Calculating performance statistics

[15]: from sklearn.metrics import roc_auc_score, roc_curve, auc
TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted_binary).ravel()
print(f"True Negative (TN) : {TN}")
print(f"False Positive (FP): {FP}")
print("False Negative (FN): {FN}")
print(f"True Positive (TP) : {TP}")
True Negative (TN) : 7
False Positive (FP): 3
False Negative (FN): 2
True Positive (TP) : 19

[16]: # Sensitivity, hit rate, recall, or true positive rate
Sensitivity = float(TP)/(TP+FN)*100
print("Sensitivity or TPR: (Sensitivity)%")
print("There is a {Sensitivity}% chance of detecting patients with an abnormality have an abnormality")
Sensitivity or TPR: 90.47619047619048%
There is a 90.47619047619048% chance of detecting patients with an abnormality have an abnormality

[17]: # Specificity or true negative rate
Specificity = float(TN)/(TN+FP)*100
print("Specificity or TNR: (Specificity)%")
print("There is a {Specificity}% chance of detecting normal patients are normal.")
Specificity or TNR: 70.0%
There is a 70.0% chance of detecting normal patients are normal.

Positive or negative predictive values

```

- **from sklearn.metrics import roc_auc_score, roc_curve, auc:** The line imports three functions: - `roc_auc_score`: Calculates the Area Under the Receiver Operating Characteristic (AUC-ROC) curve. - `roc_curve`: Generates the ROC curve. - `auc`: Computes the AUC-ROC metric.
- **TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted_binary).ravel():** This line unpacks the values of TN, FP, FN, and TP from the confusion matrix, which compares actual and predicted binary class labels. The matrix is flattened into a one-dimensional array using `ravel()`.
- The subsequent four lines display the values of TN, FP, FN, and TP.
- $\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} * 100$ - This line calculates the sensitivity (also known as recall or true positive rate) by dividing the number of true positives by the sum of true positives and false negatives, and then multiplying by 100 to get the percentage.
- The next two lines print the value of sensitivity and provide an interpretation of what it means.
- The code is calculating and printing important metrics related to the performance of a binary classification model. These metrics are derived from the confusion matrix, which is a table that summarizes the true and false predictions of the model.
- Sensitivity measures the proportion of positives correctly identified by the model.
- Specificity measures the proportion of negatives correctly identified by the model.

Positive or negative predictive values:

```
print("There is a {Specificity}% chance of detecting normal patients are normal.")

Specificity or TNR: 70.0%
There is a 70.0% chance of detecting normal patients are normal.

Positive or negative predictive values

[19]: # Precision or positive predictive value
Precision = float(TP)/(TP+FP)*100
print("Precision: {Precision}%")


Precision: 86.36363636363636%
You have an abnormality, and the probability that is correct is 86.36363636363636%

[20]: # Negative predictive value
NPV = float(TN)/(TN+FN)*100
print("Negative Predictive Value: {NPV}%")


Negative Predictive Value: 77.77777777777779%
You don't have an abnormality, but there is a 77.77777777777779% chance that is incorrect

False positive rate

[21]: # Fall out or false positive rate
FPR = float(FP)/(FP+TN)*100
print("False Positive Rate: {FPR}%")


False Positive Rate: 30.0%
There is a 30.0% chance that this positive result is incorrect.
```

- This code calculates more performance indicators for a classification model that distinguishes between two classes.
- It uses data from a confusion matrix, which keeps track of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).
- Precision, also called Positive Predictive Value (PPV), is one of these metrics. It shows how many of the predictions that said something was positive were right.
- The formula is: Precision = TP / (TP + FP) * 100.
- The code shows the precision value and says: "You have an abnormality, and the probability that is correct is {Precision}%".

NPV (Negative Predictive Value):

NPV measures how many predictions for "no abnormality" are correct.

Formula: NPV = (True Negatives / (True Negatives + False Negatives)) x 100

The code shows the NPV percentage and says, "You don't have an abnormality, but there's a {NPV}% chance that's wrong."

FPR (False Positive Rate):

FPR measures how many actual negative cases are accidentally labeled as positive.

Formula: $FPR = (\text{False Positives} / (\text{False Positives} + \text{True Negatives})) \times 100$

The code shows the FPR percentage and says, "There's a {FPR}% chance this positive result is wrong."

```

print(f"False Positive Rate: {FPR}%")
print(f"There is a {FPR}% chance that this positive result is incorrect.")

False Positive Rate: 30.0%
There is a 30.0% chance that this positive result is incorrect.

- False negative rate

[23]: # False negative rate
FNR = float(FN)/(TP+FN)*100
print(f"False Negative Rate: {FNR}%")
print(f"There is a {FNR}% chance that this negative result is incorrect.")

False Negative Rate: 9.523809523809524%
There is a 9.523809523809524% chance that this negative result is incorrect.

- False discovery rate

[24]: # False discovery rate
FDR = float(FP)/(TP+FP)*100
print(f"False Discovery Rate: {FDR}%")
print(f"You have an abnormality, but there is a {FDR}% chance this is incorrect.")

False Discovery Rate: 13.636363636363635%
You have an abnormality, but there is a 13.636363636363635% chance this is incorrect.

- Overall accuracy

[25]: # Overall accuracy
ACC = float(TP+TN)/(TP+FP+FN+TN)*100
print(f"Accuracy: {ACC}%")

Accuracy: 83.87096774193549%

```

- False Negative Rate quantifies the proportion of actual positives that are incorrectly classified as negatives, which is related to the model's sensitivity (or recall). A high FNR indicates the model is missing many positive instances.
- False Discovery Rate quantifies the proportion of positive predictions that are false positives. It is related to the model's precision (or positive predictive value). A high FDR suggests that many of the positive predictions made by the model are incorrect.
- **For example**, if the FNR is high, it may indicate that the model is struggling to identify positive instances accurately, and adjustments or additional training may be required to improve its sensitivity.
- Conversely, if the FDR is high, it suggests that the model is producing too many false positive predictions, and steps may be needed to improve its precision.

```

False Discovery Rate: 13.6363636363635%
You have an abnormality, but there is a 13.6363636363635% chance this is incorrect.

Overall accuracy

[25]: # Overall accuracy
ACC = float(TP+TN)/(TP+FP+FN+TN)*100
print("Accuracy: {ACC}%")
Accuracy: 83.8796774193549

[26]: print("Sensitivity or TPR: {Sensitivity}%")
print("Specificity or TNR: {Specificity}%")
print("Precision: {Precision}%")
print("Negative Predictive Value: {NPV}%")
print("False Positive Rate: {FPR}%")
print("False Negative Rate: {FNR}%")
print("False Discovery Rate: {FDR}%")
print("Accuracy: {ACC}%")

Sensitivity or TPR: 98.47619047619048%
Specificity or TNR: 78.0%
Precision: 86.36363636363636%
Negative Predictive Value: 77.7777777777779%
False Positive Rate: 30.0%
False Negative Rate: 9.523889523809524%
False Discovery Rate: 13.6363636363635%
Accuracy: 83.8796774193549

Challenge task-1: Record the previous values, then go back to step 1 and change the value used for the threshold. Values you should try are .25 and .75.

Did those threshold values make a difference?

```

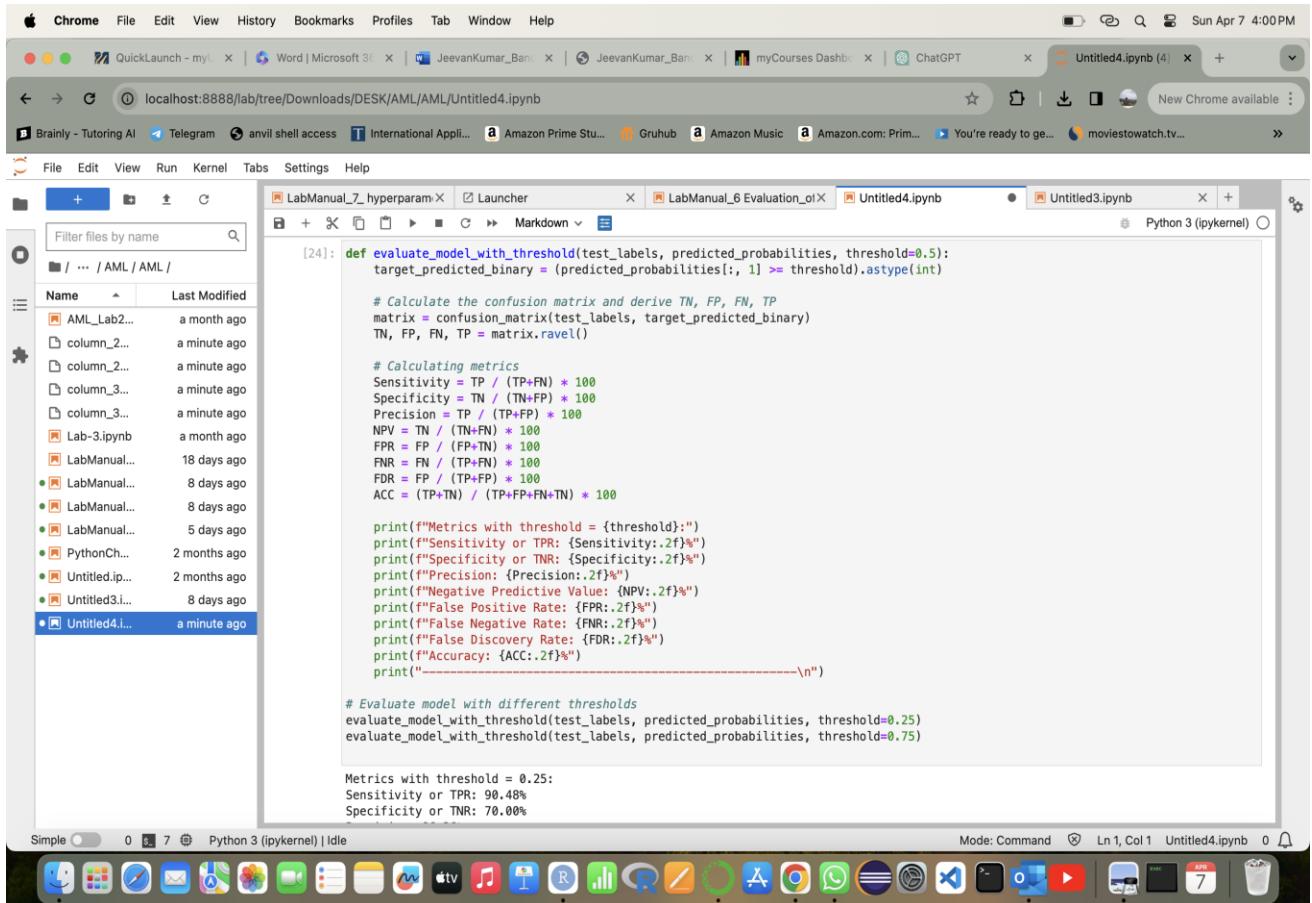
Overall Accuracy:

- The accuracy measure represents the proportion of correct predictions (both positive and negative) made by the model.
- It is calculated as: $ACC = (TP + TN) / (TP + FP + FN + TN) * 100$
- The code prints the accuracy value as a percentage.
- These measures comprehensively assess a binary classification model's effectiveness, enabling you to identify its strengths and limitations. While accuracy provides a broad view of the model's performance, it may not fully capture its behavior, especially in imbalanced datasets.
- To fully evaluate the model, it is essential to consider other metrics alongside accuracy. These metrics provide insights into the model's ability to correctly classify positive and negative samples and the trustworthiness of its predictions.
- For example, a high sensitivity coupled with a low specificity may indicate that the model is good at detecting positive instances but also tends to misclassify many negative instances as positive (high false positive rate).
- By analyzing this collection of metrics, you can identify potential areas for improvement, such as adjusting the model's decision threshold, gathering more training data, or exploring different modeling techniques or feature engineering approaches.

Challenge task 1: Record the previous values, then go back to step 1 and change the value used for the threshold. The values you should try are .25 and .75.

Did those threshold values make a difference?

Ans:



The screenshot shows a Mac desktop with a file browser window open on the left and a Jupyter Notebook cell on the right.

File Browser (Left):

- Name: AML_Lab2... Last Modified: a month ago
- column_2... a minute ago
- column_2... a minute ago
- column_3... a minute ago
- column_3... a minute ago
- Lab-3.ipynb a month ago
- LabManual... 18 days ago
- LabManual... 8 days ago
- LabManual... 8 days ago
- LabManual... 5 days ago
- PythonCh... 2 months ago
- Untitled.ip... 2 months ago
- Untitled3.ip... 8 days ago
- Untitled4.ip... a minute ago

Jupyter Notebook Cell (Right):

```
[24]: def evaluate_model_with_threshold(test_labels, predicted_probabilities, threshold=0.5):
    target_predicted_binary = (predicted_probabilities[:, 1] >= threshold).astype(int)

    # Calculate the confusion matrix and derive TN, FP, FN, TP
    matrix = confusion_matrix(test_labels, target_predicted_binary)
    TN, FP, FN, TP = matrix.ravel()

    # Calculating metrics
    Sensitivity = TP / (TP+FN) * 100
    Specificity = TN / (TN+FP) * 100
    Precision = TP / (TP+FP) * 100
    NPV = TN / (TN+FN) * 100
    FPR = FP / (FP+TN) * 100
    FNR = FN / (TP+FN) * 100
    FDR = FP / (TP+FP) * 100
    ACC = (TP+TN) / (TP+FP+FN+TN) * 100

    print(f"Metrics with threshold = {threshold}:")
    print(f"Sensitivity or TPR: {Sensitivity:.2f}%")
    print(f"Specificity or TNR: {Specificity:.2f}%")
    print(f"Precision: {Precision:.2f}%")
    print(f"Negative Predictive Value: {NPV:.2f}%")
    print(f"False Positive Rate: {FPR:.2f}%")
    print(f"False Negative Rate: {FNR:.2f}%")
    print(f"False Discovery Rate: {FDR:.2f}%")
    print(f"Accuracy: {ACC:.2f}%")
    print("\n")

# Evaluate model with different thresholds
evaluate_model_with_threshold(test_labels, predicted_probabilities, threshold=0.25)
evaluate_model_with_threshold(test_labels, predicted_probabilities, threshold=0.75)
```

Output of the code cell:

```
Metrics with threshold = 0.25:
Sensitivity or TPR: 90.48%
Specificity or TNR: 70.00%
```

```

print("False Positive Rate: {FPR:.2f}%")
print("False Negative Rate: {FNR:.2f}%")
print("False Discovery Rate: {FDR:.2f}%")
print("Accuracy: {ACC:.2f}%")
print("\n")

# Evaluate model with different thresholds
evaluate_model_with_threshold(test_labels, predicted_probabilities, threshold=0.25)
evaluate_model_with_threshold(test_labels, predicted_probabilities, threshold=0.75)

Metrics with threshold = 0.25:
Sensitivity or TPR: 90.48%
Specificity or TNR: 70.00%
Precision: 86.36%
Negative Predictive Value: 77.78%
False Positive Rate: 30.00%
False Negative Rate: 9.52%
False Discovery Rate: 13.64%
Accuracy: 83.87%


Metrics with threshold = 0.75:
Sensitivity or TPR: 80.95%
Specificity or TNR: 80.00%
Precision: 89.47%
Negative Predictive Value: 66.67%
False Positive Rate: 20.00%
False Negative Rate: 19.05%
False Discovery Rate: 10.53%
Accuracy: 80.65%

```

Step: 3 Calculating the AUC-ROC Curve

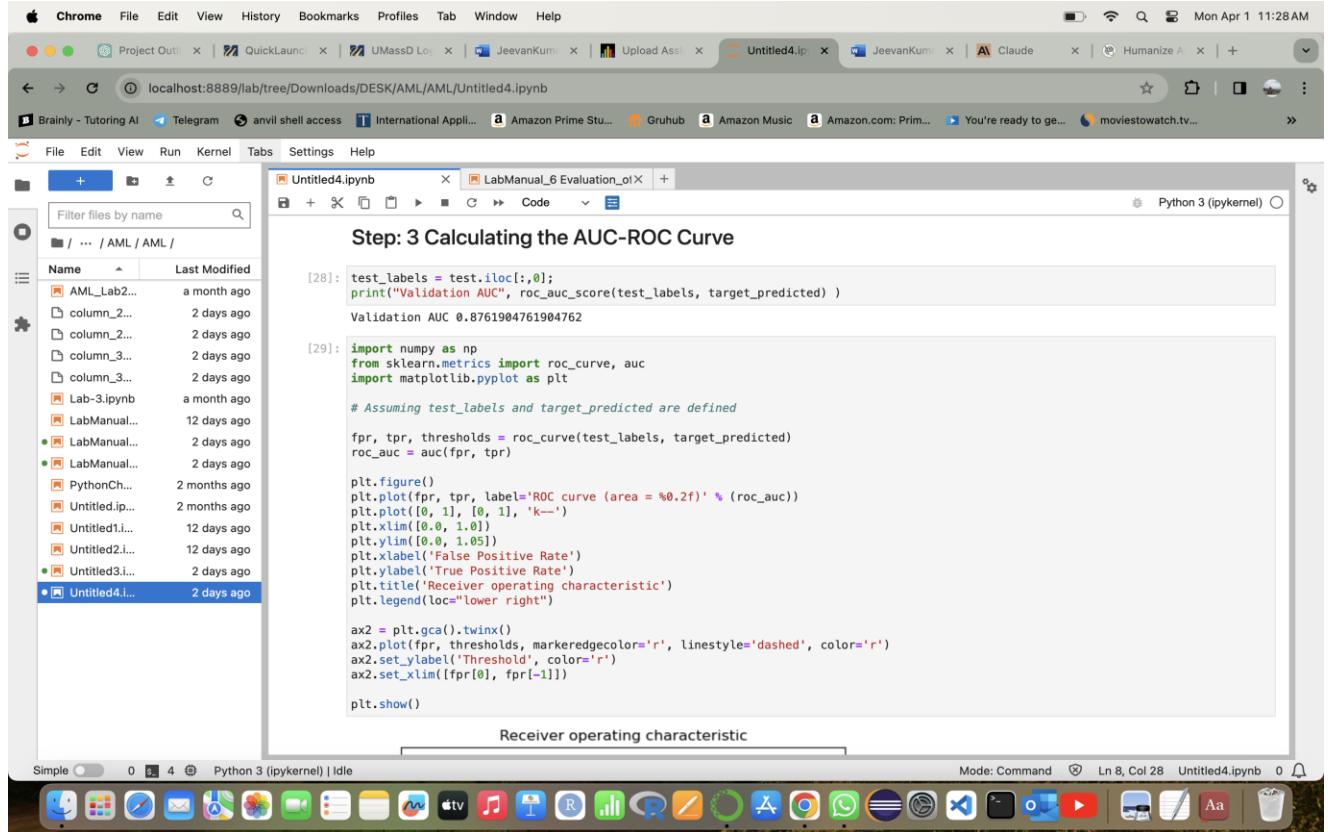
If the threshold is changed to 0.25 (lower threshold):

- The True Positive Rate (TPR) or Sensitivity would likely increase, as more instances would be classified as positive (abnormal).
- The False Positive Rate (FPR) would also increase, as more negative instances (normal) would be incorrectly classified as positive (abnormal).
- The Precision or Positive Predictive Value (PPV) would likely decrease, as the number of False Positives increases.
- The overall Accuracy might decrease or increase, depending on the distribution of classes and the impact of increased False Positives and True Positives.

If the threshold is changed to 0.75 (higher threshold):

- The True Positive Rate (TPR) or Sensitivity would likely decrease, as fewer instances would be classified as positive (abnormal).
- The False Positive Rate (FPR) would decrease, as fewer negative instances (normal) would be incorrectly classified as positive (abnormal).
- The Precision or Positive Predictive Value (PPV) would likely increase, as the number of False Positives decreases.
- The overall Accuracy might decrease or increase, depending on the distribution of classes and the impact of decreased False Positives and True Positives.

In general, adjusting the threshold value allows you to tradeoff between different performance metrics. A lower threshold favors higher recall (or TPR) at the cost of lower precision, while a higher threshold favors higher precision at the cost of lower recall.



```

Chrome File Edit View History Bookmarks Profiles Tab Window Help
localhost:8889/lab/tree/Downloads/DESK/AML/AML/Untitled4.ipynb
Brainly - Tutoring AI Telegram anvil shell access International Appli... Amazon Prime Stu... Gruhub Amazon Music Amazon.com: Prim... You're ready to ge... moviestowatch.tv...
File Edit View Run Kernel Tabs Settings Help
Untitled4.ipynb LabManual_6 Evaluation_oI
Step: 3 Calculating the AUC-ROC Curve
[28]: test_labels = test.iloc[:,0];
print("Validation AUC", roc_auc_score(test_labels, target_predicted) )
Validation AUC 0.8761904761904762

[29]: import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming test_labels and target_predicted are defined

fpr, tpr, thresholds = roc_curve(test_labels, target_predicted)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")

ax2 = plt.gca().twinx()
ax2.plot(fpr, thresholds, markeredgecolor='r', linestyle='dashed', color='r')
ax2.set_ylabel('Threshold', color='r')
ax2.set_xlim([fpr[0], fpr[-1]])

plt.show()

```

This code is used to visualize the Receiver Operating Characteristic (ROC) curve and calculate the Area Under the ROC Curve (AUC-ROC) for a binary classification model.

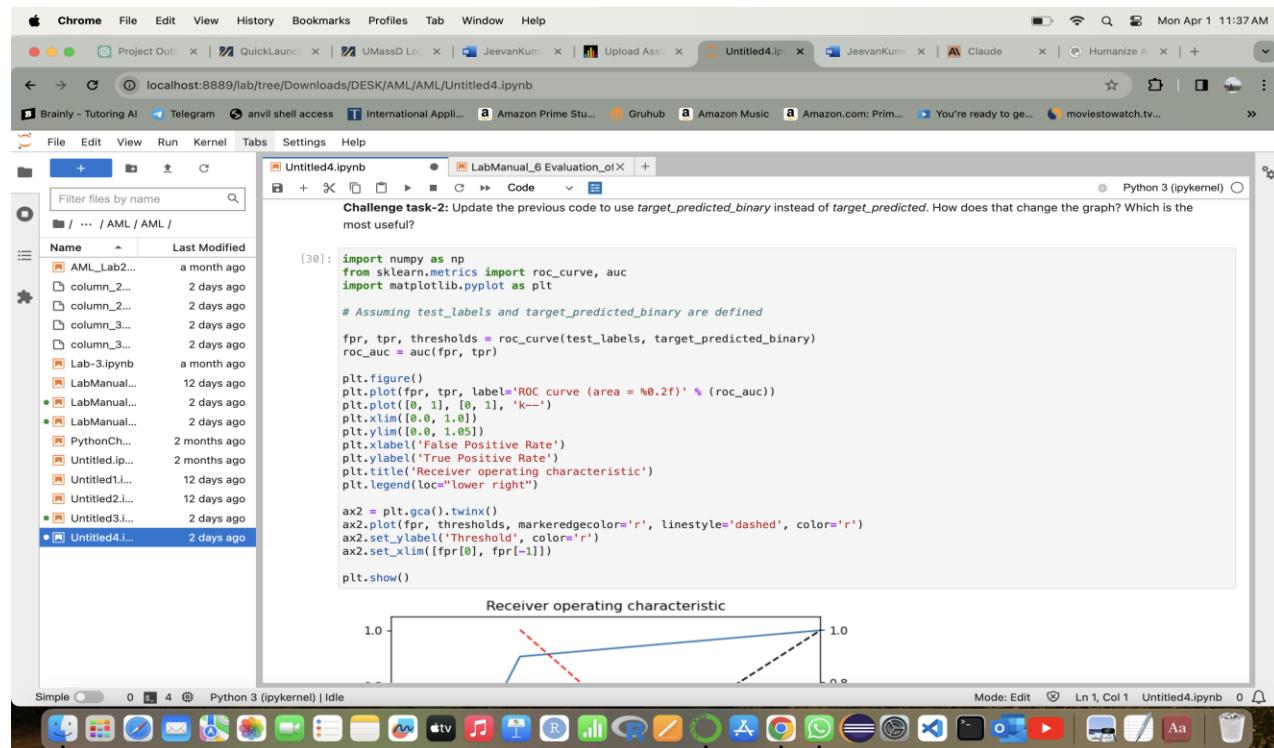
- `print ("Validation AUC", roc_auc_score(test_labels, target_predicted))` - This line calculates and prints the AUC-ROC score using the `roc_auc_score` function from `sklearn.metrics`. The AUC-ROC score is a single value that summarizes the model's performance across all possible classification thresholds.
- `fpr, tpr, thresholds = roc_curve(test_labels, target_predicted)` - This line computes the false positive rate (`fpr`), true positive rate (`tpr`), and classification thresholds (`thresholds`) for the model using the `roc_curve` function from `sklearn.metrics`.
- `roc_auc = auc(fpr, tpr)` - This line calculates the AUC-ROC value using the `auc` function from `sklearn.metrics` and the previously computed `fpr` and `tpr` values.
- The following lines create a new figure and plot the ROC curve using `matplotlib.pyplot`. The ROC curve is plotted by plotting the `tpr` (y-axis) against the `fpr` (x-axis). The AUC-ROC value is also displayed in the plot legend.

- plt.plot([0, 1], [0, 1], 'k--') - This line plots a diagonal line from (0, 0) to (1, 1), which represents the performance of a random classifier.
- The code sets the x and y limits of the plot, adds labels to the x and y axes, and sets a title for the plot.
- ax2 = plt.gca().twinx() - This line creates a secondary y-axis on the same plot.
- ax2.plot(fpr, thresholds, markeredgecolor='r', linestyle='dashed', color='r') - This line plots the classification thresholds on the secondary y-axis using a red dashed line.
- The code sets the y-axis label for the secondary y-axis as "Threshold" and adjusts the x-axis limits for the secondary y-axis.
- plt.show() - This line displays the plot.

Challenge task 2: Update the previous code to use `target_predicted_binary` instead of `target_predicted`. How does that change the graph? Which is the most useful?

Ans:

Below is the updated code as instructed in the challenging task. And you can see the output of it as well.



The screenshot shows a Jupyter Notebook interface running in a browser window. The notebook has a single cell containing Python code to generate an ROC curve. The code imports numpy, sklearn.metrics, and matplotlib.pyplot, then uses roc_curve to get fpr, tpr, and thresholds. It plots the ROC curve (fpr vs tpr) and a dashed red line from (0,0) to (1,1). The secondary y-axis is labeled 'Threshold' and ranges from 0.0 to 1.05. The plot is titled 'Receiver operating characteristic'. The output cell shows the generated plot.

```

import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Assuming test_labels and target_predicted_binary are defined

fpr, tpr, thresholds = roc_curve(test_labels, target_predicted_binary)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")

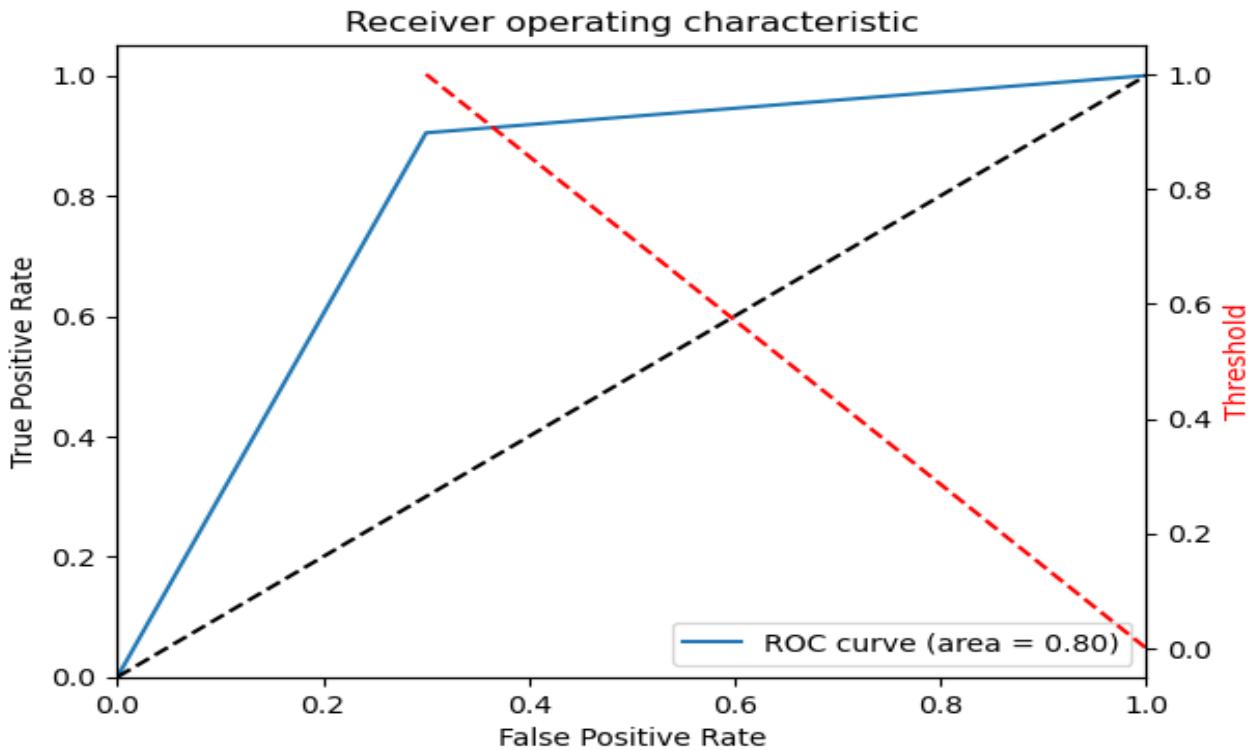
ax2 = plt.gca().twinx()
ax2.plot(fpr, thresholds, markeredgecolor='r', linestyle='dashed', color='r')
ax2.set_ylabel('Threshold', color='r')
ax2.set_xlim([fpr[0], fpr[-1]])

plt.show()

```

Receiver operating characteristic





The main difference between using `target_predicted` and `target_predicted_binary` is that `target_predicted` likely contains the predicted probabilities or scores for each instance, while `target_predicted_binary` contains the binary predictions (0 or 1) based on a specific threshold.

When using `target_predicted_binary`, the ROC curve will be a step function, with each step representing a change in the classification threshold. This is because there are a limited number of unique thresholds that can be used to convert the predicted probabilities or scores into binary predictions.

On the other hand, when using `target_predicted` (the predicted probabilities or scores), the ROC curve will be a smooth curve, as there are many possible thresholds that can be used to convert the probabilities or scores into binary predictions.

- The ROC curve generated using `target_predicted` is more useful for analyzing the model's performance across all possible thresholds and for comparing the performance of different models or algorithms. It provides a more detailed view of the trade-off between true positive rate and false positive rate at different thresholds.
- The ROC curve generated using `target_predicted_binary` is more useful for understanding the performance of the model at the specific threshold used to generate the binary predictions. It can help identify the operating point of the model (the specific true positive rate and false positive rate) at that threshold.

Finally, the choice of which ROC curve to use depends on the specific goals and requirements of the analysis, as well as the availability of the predicted probabilities or scores (target_predicted) or the binary predictions (target_predicted_binary).

Conclusion:

- This code thoroughly assesses and graphically displays the performance of a binary classification model.
- It calculates numerous performance metrics, such as sensitivity, specificity, and overall accuracy, using confusion matrix values.
- The code also emphasizes the Receiver Operating Characteristic (ROC) curve and the Area Under the ROC Curve (AUC-ROC) metric.
- It computes the AUC-ROC score using sklearn.metrics' roc_auc_score function, which provides a summary measure of the model's performance across all possible classification thresholds.
- The code creates a graph known as an ROC curve. This graph helps visualize how well the model performs in classifying data.
- It shows the true positive rate (TPR) and false positive rate (FPR) at different classification thresholds.
- Two versions of the curve are generated: one based on predicted probabilities, and one based on binary predictions.
- The first curve is smooth, while the second is a step function. The visualization includes a second y-axis that shows the classification threshold for each point on the curve.
- This information is helpful for choosing an appropriate threshold, which balances the TPR and FPR based on your specific needs.
- The code then focuses on the Receiver Operating Characteristic (ROC) curve and the Area Under the ROC Curve (AUC-ROC).
- It demonstrates the generation of two ROC curves: one using the predicted probabilities and another using the binary predictions. The former provides a smooth curve, while the latter is a step function.
- This approach enables a thorough understanding of the model's strengths and weaknesses, facilitating informed decision-making and potential model optimization strategies based on the specific requirements and priorities of the problem, such as prioritizing sensitivity over specificity or vice versa, or optimizing precision or negative predictive value based on the consequences of false positives or false negatives.
- A wide range of performance metrics are calculated, including sensitivity (recall or true positive rate), specificity (true negative rate), precision (positive predictive value), negative predictive value, false positive rate, false negative rate, false discovery rate, and overall accuracy.

- These metrics quantify different aspects of the model's performance and help identify potential areas for improvement or areas where the model excels.
- Both curves visualize the trade-off between true positive rate and false positive rate at different classification thresholds.
- Overall, the code offers a comprehensive toolkit for evaluating and visualizing the performance of a binary classification model.
- It enables a thorough understanding of the model's strengths and weaknesses, facilitating informed decision-making and potential model optimization strategies.
- By combining the confusion matrix, performance metrics, and ROC curve analysis, the code offers a comprehensive toolkit for evaluating and visualizing the performance of a binary classification model.
- This approach enables a thorough understanding of the model's behavior, facilitating informed decision-making and potential model optimization strategies.

-----X-----