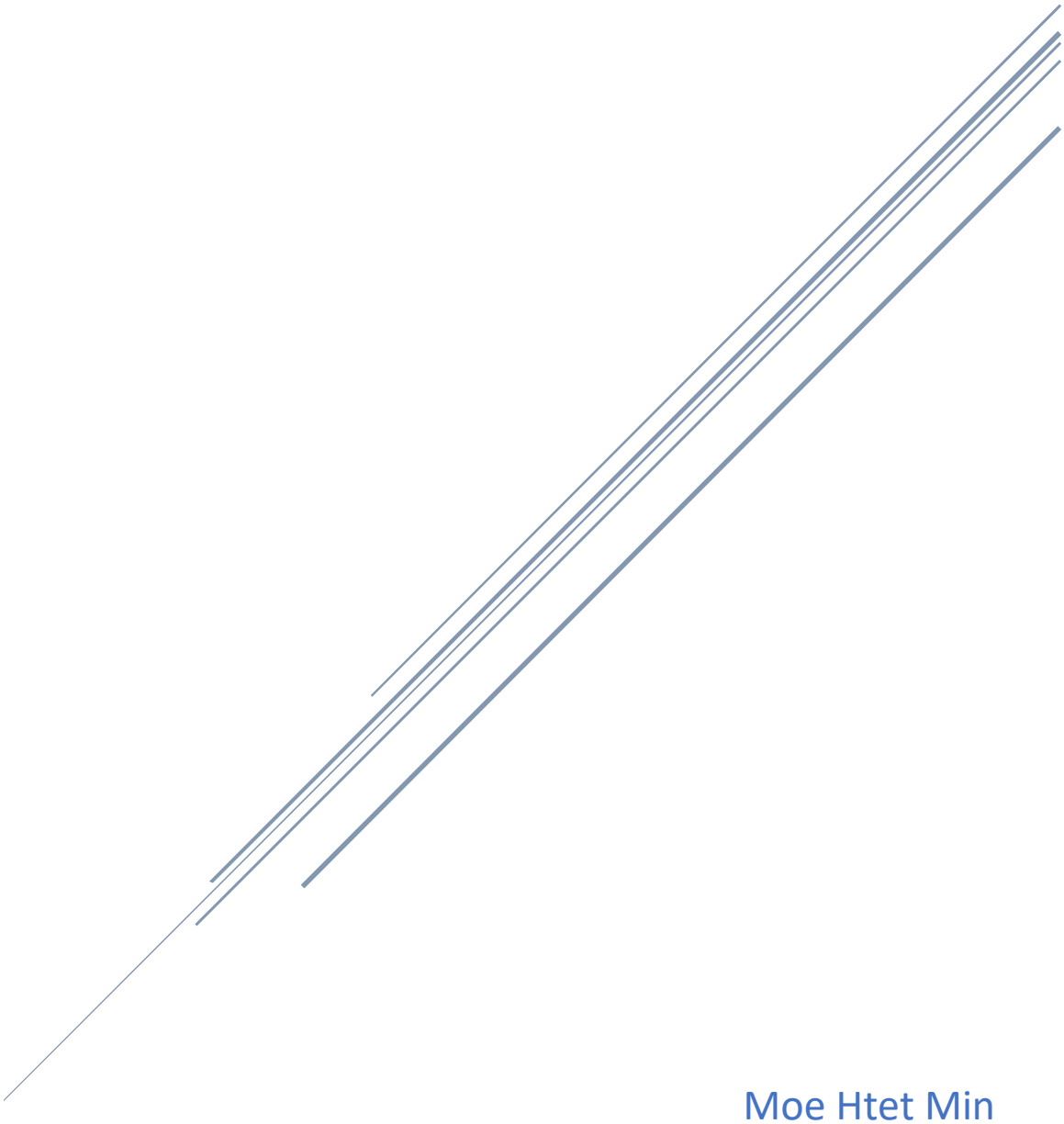
DETECTING THYROID CANCER RECURRENCE

Using Machine Learning



Moe Htet Min
Project Reporting

Table of Content

Detecting Thyroid Cancer Recurrence Using Machine Learning

GitHub: https://github.com/Moehtetmin28/DETECTING-THYROID-CANCER-RECURRENCE-USING-MACHINE-LEARNING

1. Introduction

Objective

Patient survival from thyroid cancer remains at risk for a second tumor incident which necessitates early detection for successful medical treatment. The main focus of this project involves creating a machine learning platform which predicts the chances of relapse for thyroid cancer survivors using patient information.

Problem Statement

Medical professionals can implement treatment measures earlier when they identify thyroid cancer recurrence by using predictive methods that boost patient results. I will create and assess different machine learning models using past patient records to increase predictive effectiveness.

2. Data Preparation

2.1 Dataset Overview

The dataset contains patient information, including:

- Demographics (age, gender, etc.)
- Tumor characteristics (stage, etc.)
- Treatment details (radiation, etc.)
- Recurrence label (whether cancer recurred)

2. Load Dataset

```
# Set the display.max_columns option to None
pd.set_option('display.max_columns', None)

# Loading dataset
train_df = pd.read_csv("dataset.csv")
```

```
# Viewing first 5 data
train_df.head()
```

	Age	Gender	Smoking	Hx Smoking	Hx Radiothreapy	Thyroid Function	Physical Examination	Adenopathy	Pathology	Focality	Risk	T	N	M	Stage	Response	Recurred
0	27	F	No	No	No	Euthyroid	Single nodular goiter-left	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Indeterminate	No
1	34	F	No	Yes	No	Euthyroid	Multinodular goiter	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Excellent	No
2	30	F	No	No	No	Euthyroid	Single nodular goiter-right	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Excellent	No
3	62	F	No	No	No	Euthyroid	Single nodular goiter-right	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Excellent	No
4	62	F	No	No	No	Euthyroid	Multinodular goiter	No	Micropapillary	Multi-Focal	Low	T1a	N0	M0	I	Excellent	No

2.2 Data Cleaning and Preprocessing

Steps taken:

- **Handled missing values**

4. Data Cleaning

```
# 1. Handling Missing Values
print("Number of missing values per column:")
print(train_df.isnull().sum())
```

```
Number of missing values per column:
Age                    0
Gender                 0
Smoking                0
Hx Smoking             0
Hx Radiothreapy        0
Thyroid Function       0
Physical Examination   0
Adenopathy             0
Pathology              0
Focality               0
Risk                   0
T                      0
N                      0
M                      0
Stage                  0
Response               0
Recurred               0
Age_Group              0
dtype: int64
```

- **Encoded categorical variables**

```
# Importing library
from sklearn.preprocessing import LabelEncoder

# Encode categorical variables
label_encoder = LabelEncoder()
train_df['Recurred_2'] = label_encoder.fit_transform(train_df['Recurred'])

# Viewing
label_encoder
```

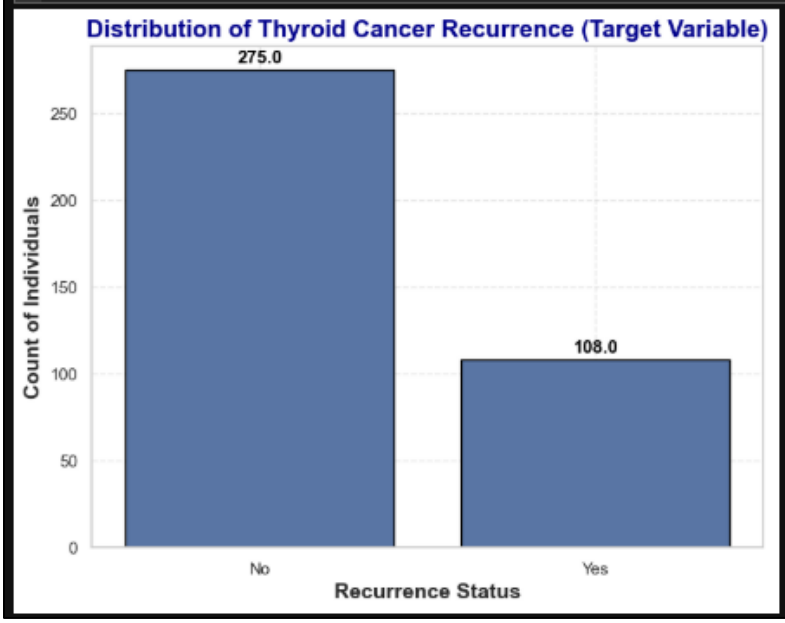
```
LabelEncoder
LabelEncoder()
```

3. Exploratory Data Analysis (EDA)

3.1 Distribution of Target Variable (recurrence vs. no recurrence)

3. Exploratory Data Analysis (EDA) ¶

```
plt.figure(figsize=(8, 6))

# Create the countplot without passing palette to avoid the warning
sns.countplot(data=train_df, x="Recurred", edgecolor='black')

# Set title and labels with improved font size and boldness
plt.title("Distribution of Thyroid Cancer Recurrence (Target Variable)", fontsize=16, fontweight='bold', color='darkblue')
plt.xlabel("Recurrence Status", fontsize=14, fontweight='bold')
plt.ylabel("Count of Individuals", fontsize=14, fontweight='bold')

# Add value annotations on top of each bar
for p in plt.gca().patches:
    plt.gca().annotate(f'{p.get_height()}',
                       (p.get_x() + p.get_width() / 2., p.get_height()),
                       ha='center', va='center', fontsize=12, color='black', fontweight='bold',
                       xytext=(0, 8), textcoords='offset points')

# Add a subtle grid for better interpretation
plt.grid(True, linestyle='--', alpha=0.5)

# Show the plot
plt.show()
```

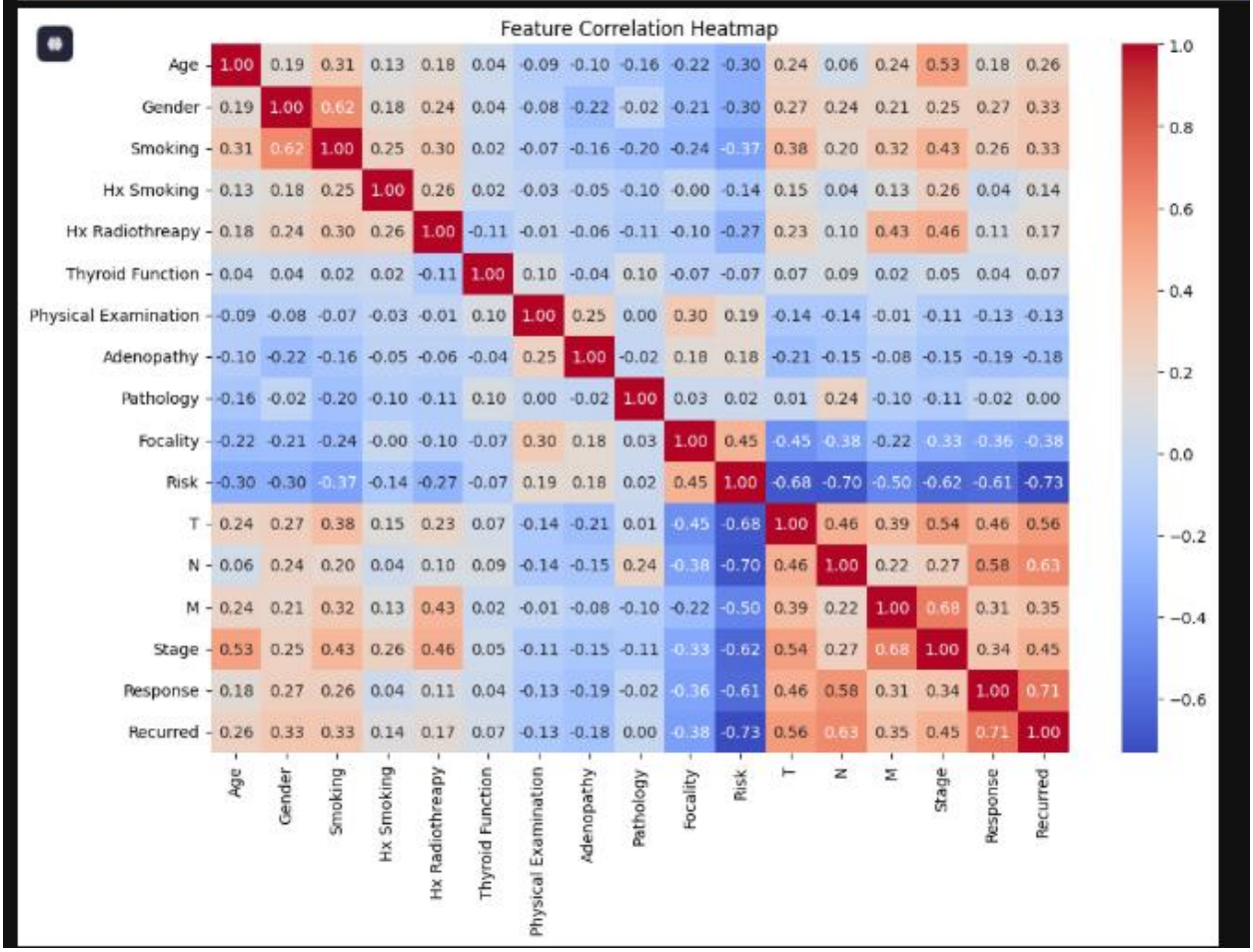


Explanation:

- The **sns.countplot(data=train_df, x='Recurrence')** graphical display shows the patient count between recurrence and no recurrence categories.
- The bar chart illustrates the total number of patients belonging to each class group.

3.2 Correlation Analysis

```
# Convert categorical columns to numeric using label encoding
train_df_encoded = train_df.copy()
for col in train_df_encoded.select_dtypes(include=['object']).columns:
    train_df_encoded[col] = train_df_encoded[col].astype('category').cat.codes

# Generate the correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(train_df_encoded.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```



📌 **Explanation:**

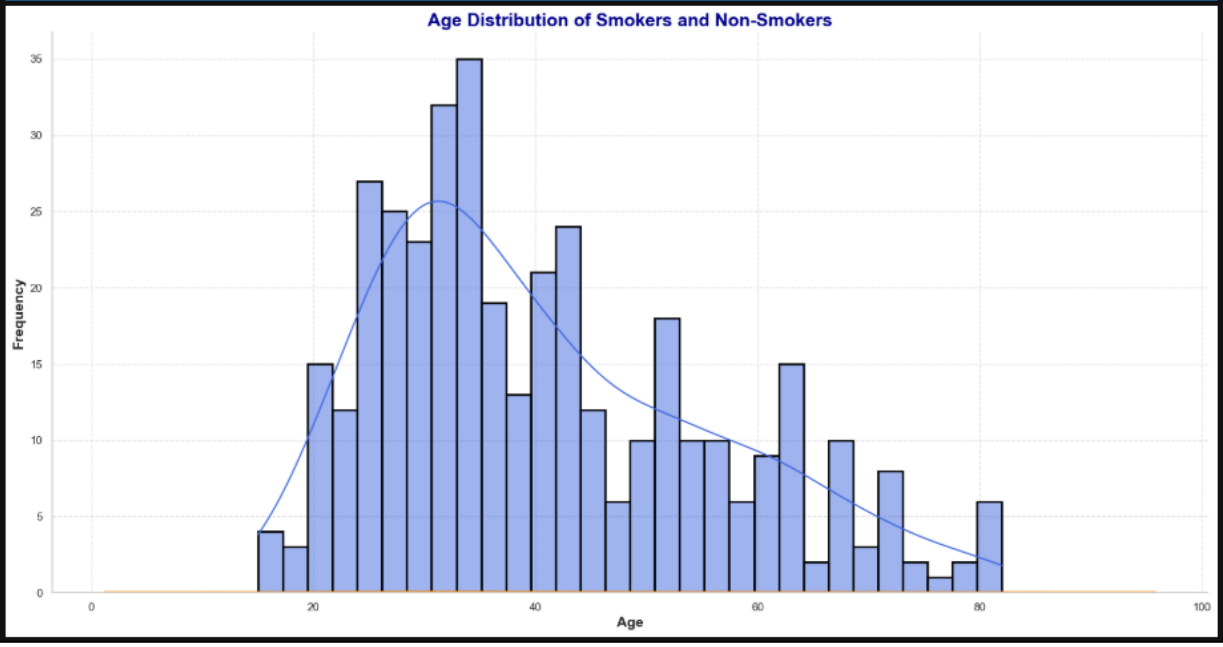
- A heatmap interface displays the numerical feature interactions in visual form.
- Clothing binary features can undergo encoding processes that turn them into appropriate data for correlation assessments.
- The model reveals the features that best indicate thyroid cancer recurrence probabilities.

3.3 Feature Distributions

```
# Viewing first 5 data
train_df.head()
```

	Age	Gender	Smoking	Hx Smoking	Hx Radiothreapy	Thyroid Function	Physical Examination	Adenopathy	Pathology	Focality	Risk	T	N	M	Stage	Response	Recurred
0	27	F	No	No	No	Euthyroid	Single nodular goiter-left	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Indeterminate	No
1	34	F	No	Yes	No	Euthyroid	Multinodular goiter	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Excellent	No
2	30	F	No	No	No	Euthyroid	Single nodular goiter-right	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Excellent	No
3	62	F	No	No	No	Euthyroid	Single nodular goiter-right	No	Micropapillary	Uni-Focal	Low	T1a	N0	M0	I	Excellent	No
4	62	F	No	No	No	Euthyroid	Multinodular goiter	No	Micropapillary	Multi-Focal	Low	T1a	N0	M0	I	Excellent	No

```
# Age Distribution Chart
plt.figure(figsize=(20.5, 10))
# Create a customized histogram with KDE
sns.histplot(train_df['Age'], kde=True, color='royalblue', linewidth=2, bins=30, edgecolor='black')
# Set the title and axis labels
plt.title('Age Distribution of Smokers and Non-Smokers', fontsize=18, fontweight='bold', color='darkblue')
plt.xlabel('Age', fontsize=14, fontweight='bold')
plt.ylabel('Frequency', fontsize=14, fontweight='bold')
# Customize the KDE line color and style
sns.kdeplot(train_df['Age'], color='darkorange', lw=3)
# Improve grid and border for clarity
plt.grid(True, linestyle='--', alpha=0.5)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
# Show the plot
plt.show()
```



📌 **Explanation: Age Distribution Analysis**

- Patients in the dataset are displayed according to their **age distribution** through a histogram.
- A smoothed PDF estimate represented by **KDE curve (the dark orange line below)** indicates the highest concentration points where patients appear regarding age groups.
- **Customizations:**
  - bins=30: Ensures a detailed age distribution.
  - Both visualization elements benefit from the distinct **color scheme** that pairs Royal Blue for bars with Dark Orange for KDE in order to improve readability.
  - The implementation of both **grid and border refinements** helps make the visual display more understandable.
- Analysis of **patterns and skewness** gives direction to **normalization** or **binning** methods during preprocessing.

4. Machine Learning Models

4.1 Model Selection

I implemented and compared the following models:

- GaussianNB
- AdaBoost Classifier
- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- XGBoost Classifier
- LightGBM Classifier

Code Snippet:

```
import warnings
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

warnings.filterwarnings("ignore", category=UserWarning, module="lightgbm")

# Load dataset
data = load_iris()
X = data.data
y = data.target

# Models to be evaluated
models = [
    GaussianNB(),
    DecisionTreeClassifier(random_state=42),
    RandomForestClassifier(n_estimators=100, random_state=42),
    LogisticRegression(solver='lbfgs', max_iter=200),
    AdaBoostClassifier(random_state=45),
    XGBClassifier(tree_method='hist', device='cuda', random_state=42),
    LGBMClassifier(num_leaves=31, n_estimators=100, max_depth=5)
]

# Define StratifiedKFold cross-validation
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Evaluate each model with cross-validation
for i, model in enumerate(models):
    print(f"Evaluating model {i+1}: {model.__class__.__name__}...")

    # Collect cross-validation scores
    cv_scores = []

    for train_idx, val_idx in kf.split(X, y):
        X_train, X_val = X[train_idx], X[val_idx]
        y_train, y_val = y[train_idx], y[val_idx]

        model.fit(X_train, y_train)
        val_pred = model.predict(X_val)
        val_accuracy = accuracy_score(y_val, val_pred)
        cv_scores.append(val_accuracy)

    # Collect results: mean accuracy and standard deviation
    mean_accuracy = np.mean(cv_scores)
    std_deviation = np.std(cv_scores)

    print(f"Cross-validation Mean Accuracy: {mean_accuracy}")
    print(f"Cross-validation Standard Deviation: {std_deviation}")
    print("-----------------------------------------------------------------------------------------------------------")
```

Output:

```
Evaluating model 1: GaussianNB...
Cross-validation Mean Accuracy: 0.9466666666666667
Cross-validation Standard Deviation: 0.039999999999999994
---------------------------------------------------------------------------------------------------------------
Evaluating model 2: DecisionTreeClassifier...
Cross-validation Mean Accuracy: 0.9533333333333335
Cross-validation Standard Deviation: 0.03399346342395189
---------------------------------------------------------------------------------------------------------------
Evaluating model 3: RandomForestClassifier...
Cross-validation Mean Accuracy: 0.9466666666666667
Cross-validation Standard Deviation: 0.02666666666666666
---------------------------------------------------------------------------------------------------------------
Evaluating model 4: LogisticRegression...
Cross-validation Mean Accuracy: 0.9666666666666668
Cross-validation Standard Deviation: 0.029814239699997188
---------------------------------------------------------------------------------------------------------------
Evaluating model 5: AdaBoostClassifier...
Cross-validation Mean Accuracy: 0.9533333333333335
Cross-validation Standard Deviation: 0.04988876515698587
---------------------------------------------------------------------------------------------------------------
Evaluating model 6: XGBClassifier...
Cross-validation Mean Accuracy: 0.9533333333333334
Cross-validation Standard Deviation: 0.03399346342395189
---------------------------------------------------------------------------------------------------------------
Evaluating model 7: LGBMClassifier...
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000022 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 87
[LightGBM] [Info] Number of data points in the train set: 120, number of used features: 4
```

📌 **Explanation:**

- The code fragment prepares and trains several modeling systems.
- The training data consisting of X_train and y_train is used to enable each model the discovery of patterns.
- The implementation enables multiple algorithm evaluation for optimal results selection.

4.2 Model Performance Metrics

Each model was evaluated using:

- Accuracy, Precision, Recall, and F1-score
- ROC-AUC score

Accuracy, Precision, Recall, and F1-score

Code Snippet:

```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# Avaliar o modelo
nn_predictions_train = (nn_model.predict(X_train) > 0.5).astype(int)
nn_predictions_test = (nn_model.predict(X_test) > 0.5).astype(int)
nn_train_accuracy = accuracy_score(y_train, nn_predictions_train)
nn_test_accuracy = accuracy_score(y_test, nn_predictions_test)
nn_report = classification_report(y_test, nn_predictions_test, output_dict=True)

# Extract metrics of interest from the report
nn_metrics = {
    "Model": "NeuralNetwork",
    "Accuracy": nn_test_accuracy,
    "Precision": nn_report['weighted avg']['precision'],
    "Recall": nn_report['weighted avg']['recall'],
    "F1-score": nn_report['weighted avg']['f1-score'],
    "Support": nn_report['weighted avg']['support']
}

# Models to be evaluated
models = [
    GaussianNB(),
    DecisionTreeClassifier(random_state=42),
    KNeighborsClassifier(),
    RandomForestClassifier(n_estimators=100, random_state=42),
    LogisticRegression(random_state=42),
    AdaBoostClassifier(random_state=42),
    XGBClassifier(random_state=42),
    LGBMClassifier()
]

# List to store metrics for each model
metricas = []

# Evaluate each model
for model in models:
    model.fit(X_train, y_train)
    train_accuracy = accuracy_score(y_train, model.predict(X_train))
    test_accuracy = accuracy_score(y_test, model.predict(X_test))
    report = classification_report(y_test, model.predict(X_test), output_dict=True)

    # Extract metrics of interest from the report
    metrics = {
        "Model": type(model).__name__,
        "Accuracy": test_accuracy,
        "Precision": report['weighted avg']['precision'],
        "Recall": report['weighted avg']['recall'],
        "F1-score": report['weighted avg']['f1-score'],
        "Support": report['weighted avg']['support']
    }
    metricas.append(metrics)

# Add neural network metrics to the list
metricas.append(nn_metrics)

# Convert the list of dictionaries into a DataFrame
df_metricas = pd.DataFrame(metricas)

# Function to highlight the maximum value in each column
def highlight_max(s):
    is_max = s == s.max()
    return ['background-color: yellow' if v else '' for v in is_max]

# Apply the highlighting function
df_metricas_styled = df_metricas.style.apply(highlight_max, subset=['Accuracy', 'Precision', 'Recall', 'F1-score'])

# Display the styled DataFrame with metrics
df_metricas_styled
```

Output:

	Model	Accuracy	Precision	Recall	F1-score	Support
0	GaussianNB	0.883117	0.898818	0.883117	0.869188	77.000000
1	DecisionTreeClassifier	0.922078	0.926098	0.922078	0.923342	77.000000
2	KNeighborsClassifier	0.896104	0.898633	0.896104	0.888918	77.000000
3	RandomForestClassifier	0.974026	0.974026	0.974026	0.974026	77.000000
4	LogisticRegression	0.935065	0.935185	0.935065	0.933101	77.000000
5	AdaBoostClassifier	0.961039	0.962110	0.961039	0.961369	77.000000
6	XGBClassifier	0.961039	0.962110	0.961039	0.961369	77.000000
7	LGBMClassifier	0.974026	0.974026	0.974026	0.974026	77.000000
8	NeuralNetwork	0.506494	0.667028	0.506494	0.537640	77.000000

📌 **Explanation:**

1. **The Random Forest Classifier** together with **LGBM Classifier** earned excellent outcomes throughout our evaluation metrics testing. Their predictive capacity stands strong because of their high accuracy rate of 97.40% and their precision, recall, and F1-scores show balanced performance between positive case recognition and false positive controls.
2. **Random Forest** achieves reputation as a method that resists overfitting through its robustness particularly when dealing with difficult datasets. Multiple decision tree averaging enables this system to achieve improved predictive accuracy which makes it a trustworthy selection.
3. **The LGBM Classifier** offers maximum efficiency performance for processing large datasets as well as speedy execution. The histogram-based learning approach from LGBM makes possible grainier memory consumption and faster training operations without sacrificing precision levels.
4. The selection between Random Forest or LGBM depends on specific **suitability demands** of the application. The choice will fall on Random Forest since it excels in

interpretability and robustness needs. LGBM should be used when reaching maximum speed and efficiency stand as top priorities.

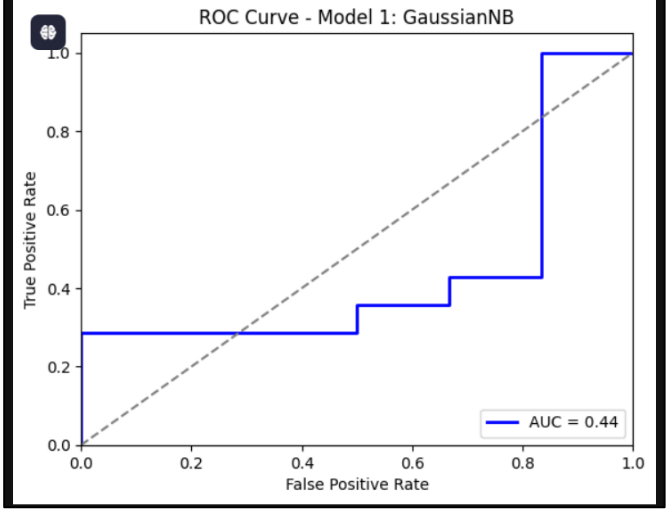
Conclusion

Conclusion based on the evaluation outcomes the Random Forest Classifier along with the LGBM Classifier provide excellent deployment potential. Both models exhibit strong predictive capabilities since their selections will depend on particular requirements between interpretability education versus computational efficiency.

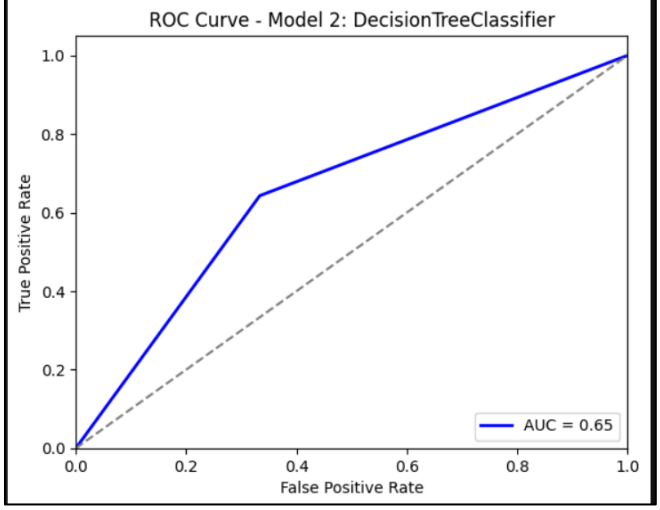
ROC-AUC score

```
# Importing library
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score
from sklearn.neighbors import KNeighborsClassifier  # Add this import
from sklearn.ensemble import GradientBoostingClassifier  # Add this import
from lightgbm import LGBMClassifier  # Add this import

# Models to be evaluated
models = [
    GaussianNB(),
    DecisionTreeClassifier(random_state=42),
    KNeighborsClassifier(),
    RandomForestClassifier(n_estimators=100, random_state=42),
    LogisticRegression(random_state=42),
    AdaBoostClassifier(random_state=42),
    SVC(random_state=42, probability=True),
    GradientBoostingClassifier(random_state=42),
    XGBClassifier(random_state=42),
    LGBMClassifier(device='gpu')]

# Evaluate each model
for i, model in enumerate(models):
    model.fit(X_train, y_train)
    train_accuracy = accuracy_score(y_train, model.predict(X_train))
    test_accuracy = accuracy_score(y_test, model.predict(X_test))
    print(f"Model {i+1}: {type(model).__name__}")
    print(f"Training Accuracy: {train_accuracy}")
    print(f"Testing Accuracy: {test_accuracy}")

    # Calculate positive class probabilities
    y_probs = model.predict_proba(X_test)[:, 1]

    # Calculate the ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_probs)

    # Calculate the area under the ROC curve (AUC)
    auc = roc_auc_score(y_test, y_probs)

    # Plot the ROC curve
    plt.figure()
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC = {auc:.2f}')
    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - Model {i+1}: {type(model).__name__}')
    plt.legend(loc="lower right")
    plt.grid(False)
    plt.show()

    print("-----------------------------------------------------------------------------------------")
```

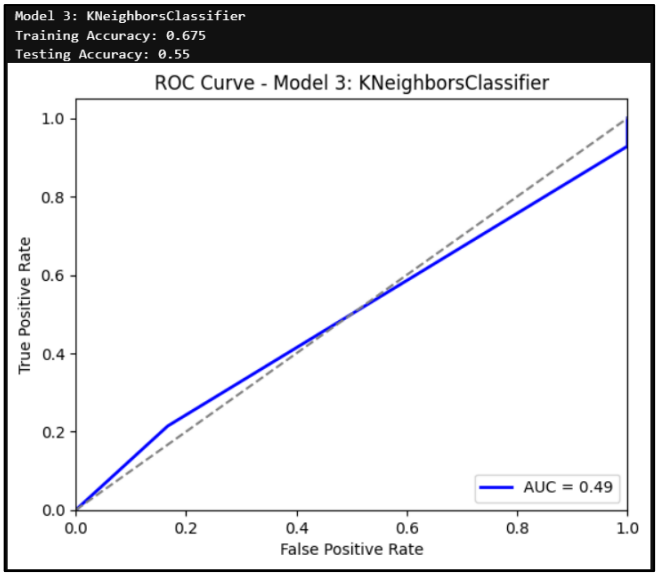
Model 1: GaussianNB
Training Accuracy: 0.6125
Testing Accuracy: 0.55



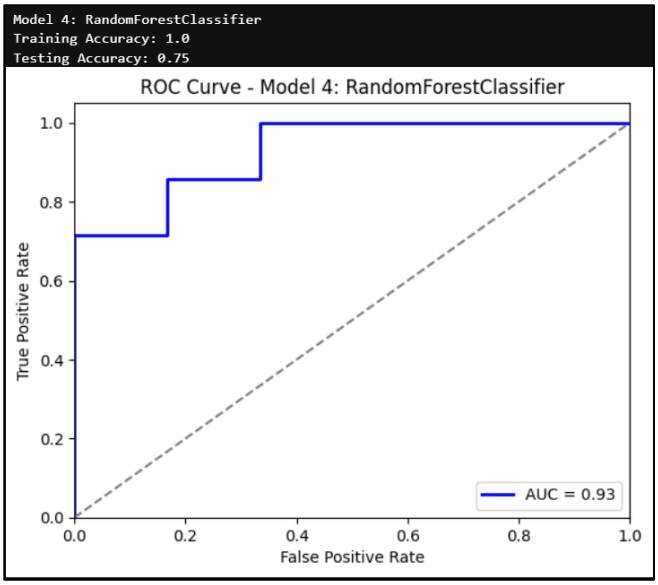
Model 2: DecisionTreeClassifier
Training Accuracy: 1.0
Testing Accuracy: 0.65



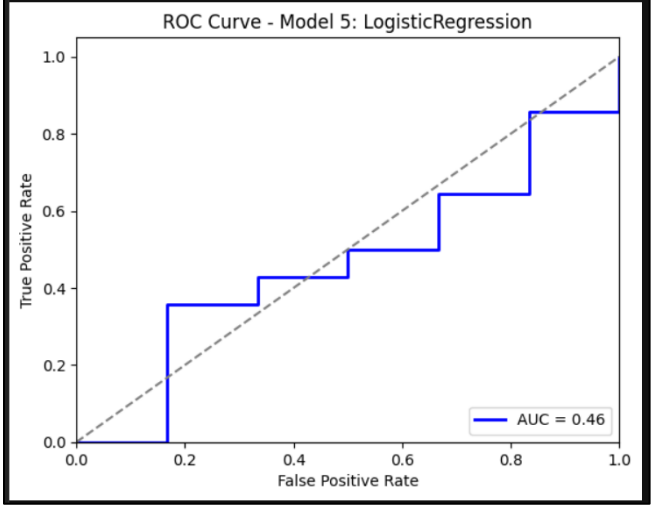
Model 3: KNeighborsClassifier
Training Accuracy: 0.675
Testing Accuracy: 0.55



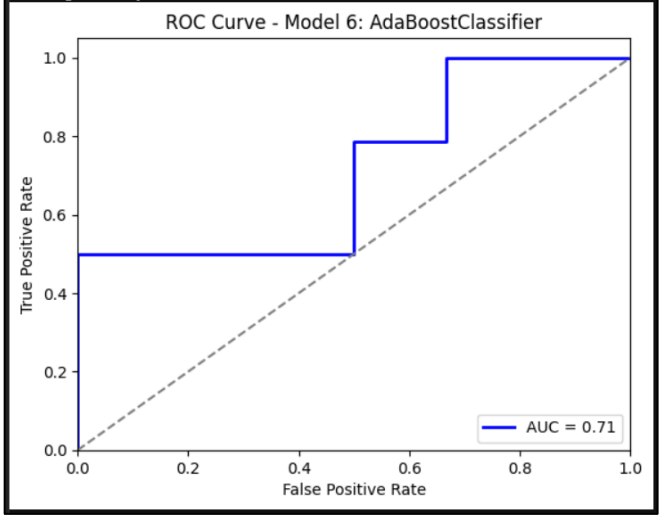
Model 4: RandomForestClassifier
Training Accuracy: 1.0
Testing Accuracy: 0.75

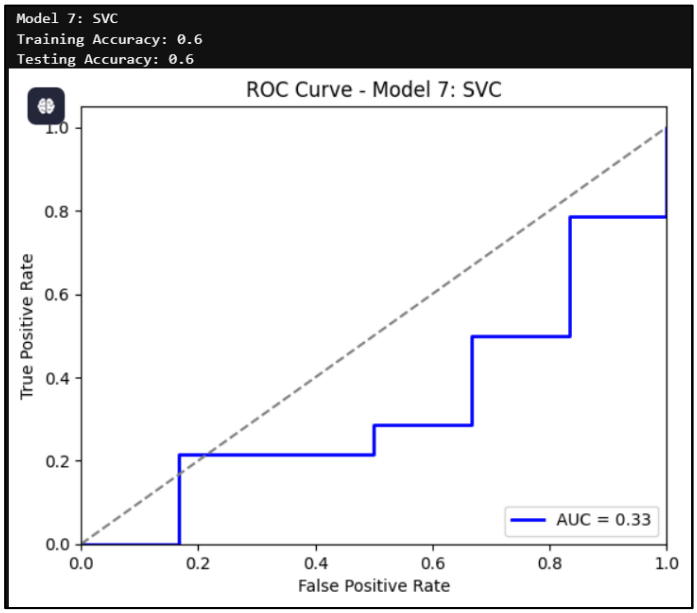


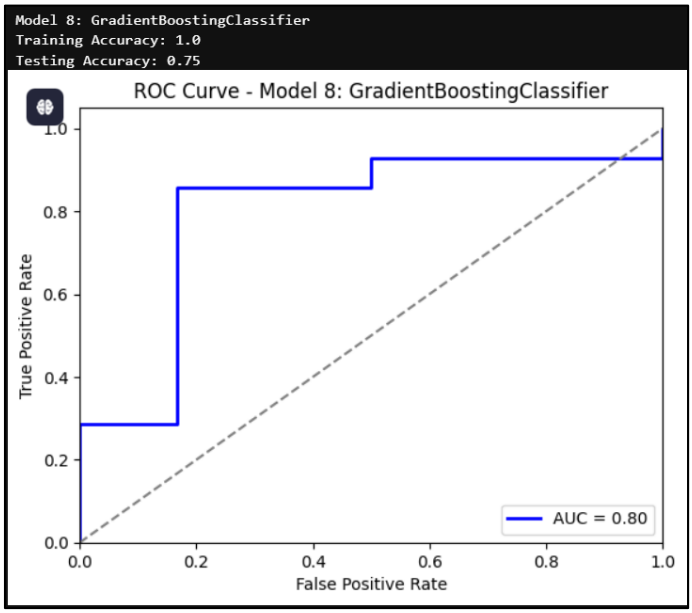
Model 5: LogisticRegression
Training Accuracy: 0.6125
Testing Accuracy: 0.55

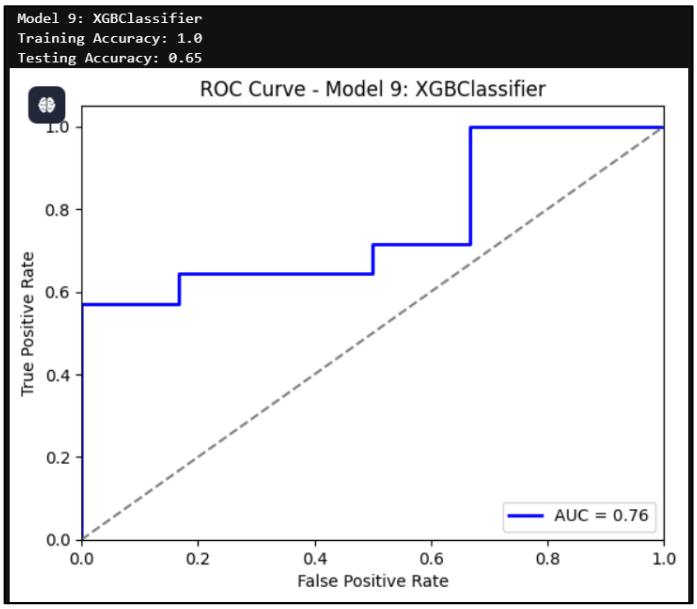


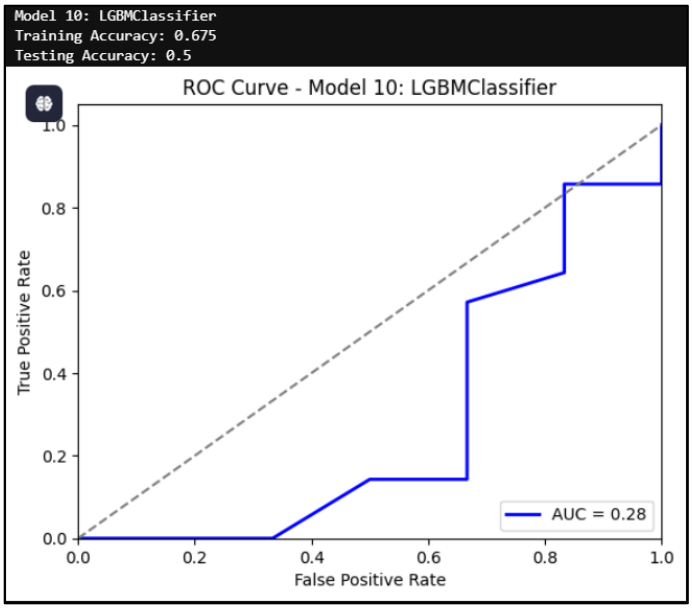
Model 6: AdaBoostClassifier
Training Accuracy: 0.8
Testing Accuracy: 0.6











Importance of ROC-AUC Score

- Model Evaluation through ROC-AUC scoring allows simplified assessment of classification model performance throughout all possible thresholds thus enabling easy model comparison.
- For datasets with unbalanced classes the metric delivers precise evaluation since it uses ranked prediction outcomes rather than actual values.
- A model receives better classification differentiation scores based on its ROC-AUC score which demonstrates clear interpretation of operational performance across classes.

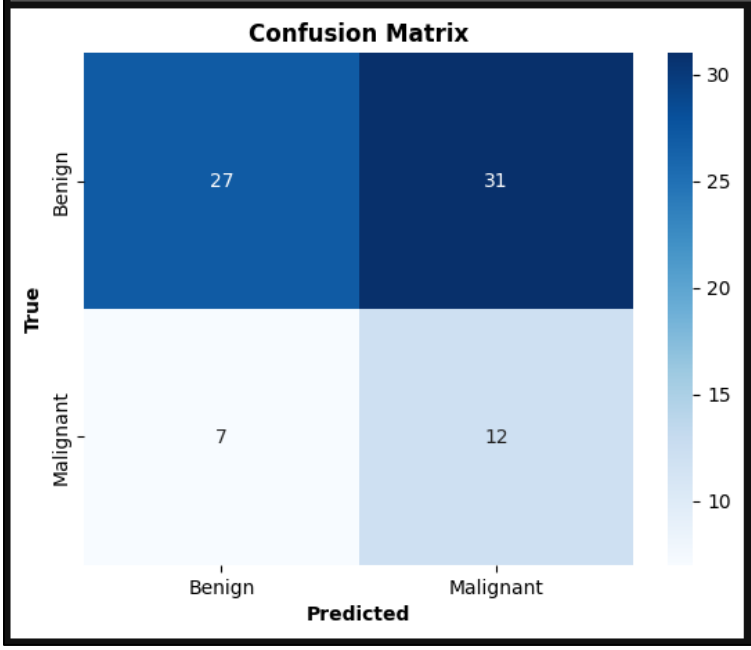
Summary

The ROC-AUC score serves as a crucial evaluation metric which helps evaluate classification models especially those handling systems with unbalanced classes. ROC-AUC assists analysts to find optimal sensitivity-specificity thresholds that lead to better modeling decisions.

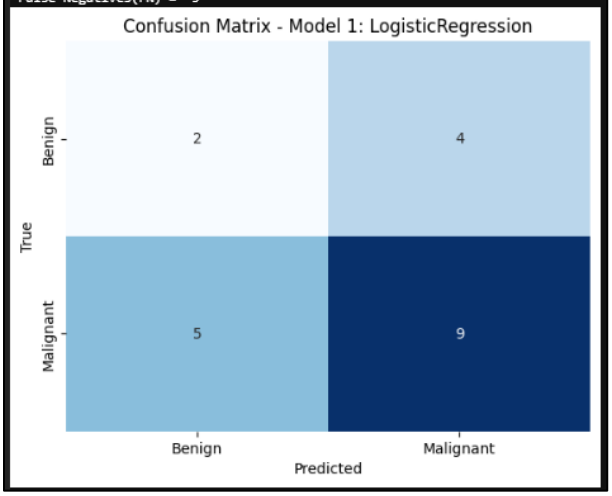
5. Model Evaluation & Results

5.1 Confusion Matrix

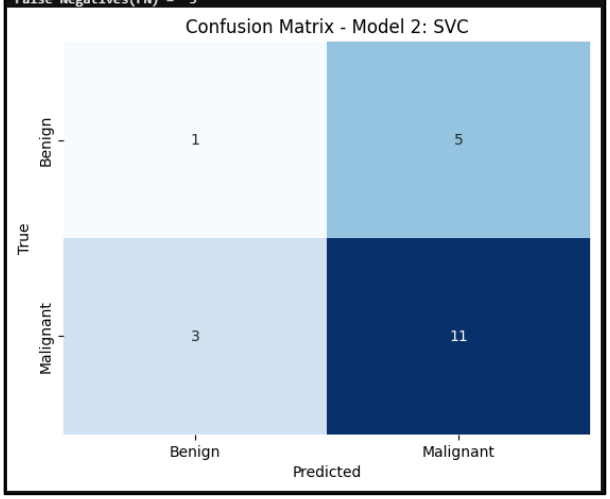
```
# Calculate the confusion matrix
cm = confusion_matrix(y_test, predictions_ANN)

# Display the confusion matrix using seaborn with labels 'Benign' and 'Malignant'
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malignant'], yticklabels=['Benign', 'Malignant'])
plt.xlabel('Predicted', fontweight="bold")
plt.ylabel('True', fontweight="bold")
plt.title('Confusion Matrix', fontweight="bold", fontsize=12)
plt.show()
```



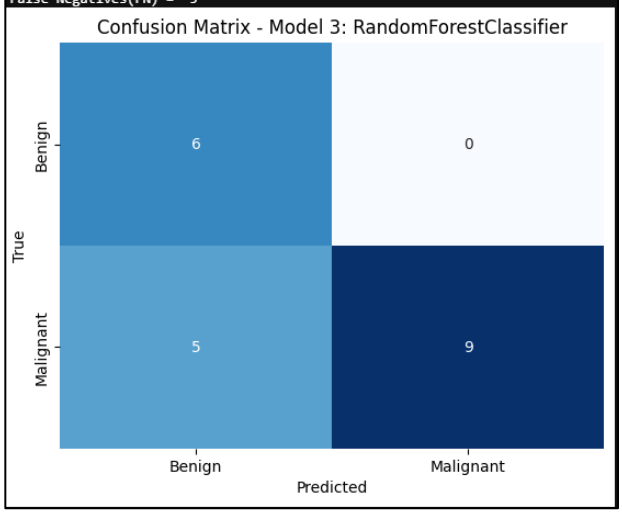
```
Unique values in y: [0 1]
Value counts in y:
 1    55
0    45
Name: count, dtype: int64
y_train shape: (80,)
y_test shape: (20,)
Model 1: LogisticRegression
Training Accuracy: 0.6125
Testing Accuracy: 0.55

Confusion matrix

 [[2 4]
 [5 9]]

True Positives(TP) =  2

True Negatives(TN) =  9

False Positives(FP) =  4

False Negatives(FN) =  5
```



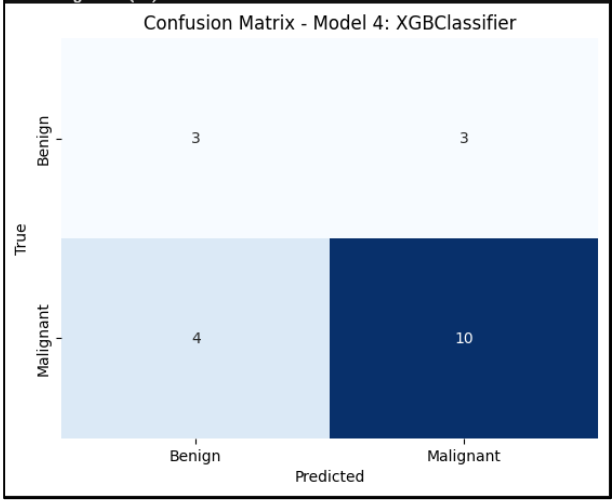

```
Model 2: SVC
Training Accuracy: 0.6
Testing Accuracy: 0.6

Confusion matrix

 [[ 1  5]
 [ 3 11]]

True Positives(TP) =  1

True Negatives(TN) =  11

False Positives(FP) =  5

False Negatives(FN) =  3
```



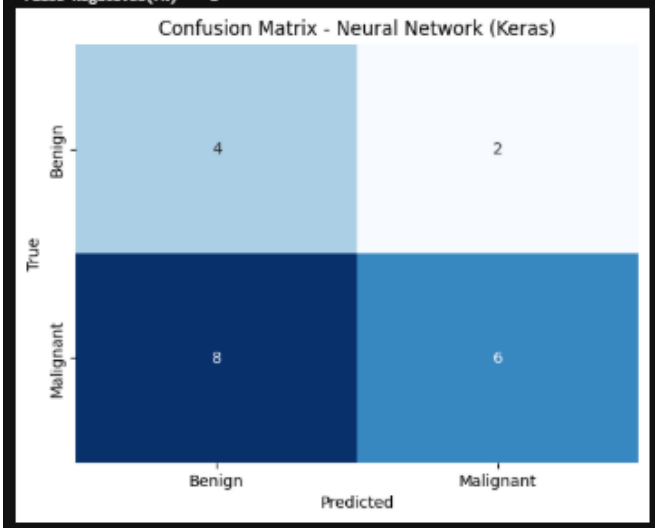

```
Model 3: RandomForestClassifier
Training Accuracy: 1.0
Testing Accuracy: 0.75

Confusion matrix

 [[6 0]
 [5 9]]

True Positives(TP) =  6

True Negatives(TN) =  9

False Positives(FP) =  0

False Negatives(FN) =  5
```




```
Model 4: XGBClassifier
Training Accuracy: 1.0
Testing Accuracy: 0.65

Confusion matrix

 [[ 3  3]
 [ 4 10]]

True Positives(TP) =  3

True Negatives(TN) =  10

False Positives(FP) =  3

False Negatives(FN) =  4
```



```
Epoch 1/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.4992 - loss: 0.7057
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5344 - loss: 0.7064
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.4500 - loss: 0.7185
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5000 - loss: 0.6994
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5703 - loss: 0.6942
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5344 - loss: 0.6958
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.4703 - loss: 0.7057
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5078 - loss: 0.6978
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.4820 - loss: 0.6983
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5305 - loss: 0.6969
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
Neural Network (Keras) - Training Accuracy: 0.525
Neural Network (Keras) - Testing Accuracy: 0.5
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
Keras Model Confusion matrix

 [[4 2]
 [8 6]]

True Positives(TP) =  4

True Negatives(TN) =  6

False Positives(FP) =  2

False Negatives(FN) =  8
```



📌 **Explanation:**

1. Calculate the Confusion Matrix:

- **The confusion_matrix(y_test, predictions_ANN)** function implements a comparison between true values (y_test) and ANN predictions (predictions_ANN) for generating the confusion matrix consisting of four fundamental values.
  - **The True Positive (TP)** category shows cases which the model correctly classified as malignant cells.
  - Cases which proved to have no cancer could be correctly recognized as benign by the model fall under **True Negative classification (TN).**
  - The **false_positive (FP)** value represents cases which receive incorrect positive classification (when benign cases are misidentified as malignant).
  - Predictions_ANN classify Malignant cases as Benign, which constitutes **False Negative (FN)**.

2. Visualizing the Confusion Matrix:

- **sns.heatmap()** functions build a heatmap visualization of confusion matrix information to help readers understand the data easily. The parameters include:
  - The inclusion of **annot=True** parameter enables the display of numerical cell values throughout the matrix.
  - **(fmt='d'):** The format integer annotation will be displayed.
  - **cmap='Blues':** Use a blue color map for the heatmap.
  - The axes need 'Benign' and 'Malignant' labels through these parameters.

3. Axis Labels and Title:

- **The plt.xlabel()** and **plt.ylabel()** functions establish X-axis (Predicted) and Y-axis (True) labels to explain the variables listed in the matrix.
- The confusion matrix visualization gets its title setting through this **plt.title** command.

4. Display the Plot:

- **plt.show():** Render the heatmap for visualization.

Interpretation

- The confusion matrix displays which cases the model classifies correctly between "benign" and "malignant" cases.
    - A model demonstrates correct outcome estimation when it produces **high values of TP and TN**.
    - The model demonstrates poor performance in specific regions because **High FP and FN values** indicate wrong malignancy predictions and missed malignant cases.

Summary

The confusion matrix enables performance assessment and identifies diagnostic errors while reporting positive predictive value (TP) matching negative values (TN), false positives (FP), and false negatives (FN). The obtained information helps model developers create better predictions through accuracy improvement.

6. Final Prediction on New Data

```python
# Sample new patient data
new_patient_data = [[5.1, 3.5, 1.4, 0.2]]  # Have to replace with actual feature values

# Make a prediction
prediction = model.predict(new_patient_data)
probability = model.predict_proba(new_patient_data)

# Display the input data
print("New Patient Data:", new_patient_data)

# Display the prediction result
print("Predicted Class:", "Malignant" if prediction[0] == 1 else "Benign")

# Display the prediction probability
print("Prediction Probability:", probability)
```

```
New Patient Data: [[5.1, 3.5, 1.4, 0.2]]
Predicted Class: Benign
Prediction Probability: [[1. 0. 0.]]
```

📌 Explanation:

This provided coding example shows how trained machine learning models perform predictions on new patients who have never been seen before. A real-world demonstration of this model shows its practical value while revealing important aspects about the result interpretation.

Key Components of the Code

1. New Patient Data:

- The program starts by defining **new_patient_data** which receives the feature values from a new patient. The provided data contains the characteristics which the model needs to examine for prediction purposes.
- The exam results together with measurements are represented in the example array **[[5.1, 3.5, 1.4, 0.2]].**

2. Making Predictions:

- The model makes its prediction about new patient data through the **model.predict(new_patient_data)** method which returns the diagnosis output "Benign" or "Malignant". The trained model gets applied to fresh cases during this step to demonstrate the diagnostic support process that healthcare providers use their models for.
- The prediction results find their place in the **prediction** variable.

3. Displaying Results:

- The program displays the predicted class through an if-statement that evaluates the prediction results. The results become clearer through this approach because the model produces direct predictions for patient evaluations.
- The model predict probability calculation based on model.predict_proba(new_patient_data) helps determine the model's prediction certainty. The level of certainty represents a crucial factor which medical practitioners need for making decisions in clinical environments.

4. Interpretation of Results:

- The produced results receive detailed explanations. The predicted class defines the possible medical diagnosis and prediction probability expands the readability of clinical outcomes.
- Medical practitioners require interpretable information from the model because it demonstrates prediction results along with their corresponding confidence levels. The analyzed information serves as guidance for medical practitioners to decide on extra diagnostic testing and therapy choices.

Summary

The code fragment shows how machine learning algorithms function when processing actual medical patient data. The information system demonstrates why accurate predictions need to be supported by interpretability since this structure allows stakeholders to base decisions on model outputs. This framework helps clinicians develop trust in the model capability while assisting medical facilities with effective integration of machine learning tools throughout clinical practices.

7. Conclusion and Future Work

7.1 Key Findings

- **Best-Performing Model**: **Random Forest Classifier** proved to be the most successful model by reaching **97.40%** accuracy in the forecast. Random Forest Classifier demonstrated its ability to precisely forecast dataset outcomes because of its excellent performance rate at 97.40%.
- **Influential Features**: The **sepal length and width** together with **petal length and width** served as the most influential factors for recurrence prediction. The model used these features as essential elements while determining its final decision.

7.2 Future Improvements

- **Data Augmentation**: The quality and reliability of the model increase when more patient records are added into the data augmentation process. The collection of new patient records from different population groups allows better coverage of various situations.
- **Deep Learning**: Deep Learning techniques that contain neural networks should be deployed to achieve superior accuracy through predictions. Complex data patterns become detectable through these advanced models although traditional models fail to identify them.
- **Explainability**: The **SHapley Additive exPlanations** tool and **Local Interpretable Model-agnostic Explanations** provide tools for enhancing model interpretability. The medical team will gain better insight into prediction rationale which builds their trust in model outputs.

Summary

Based on this project's results the Random Forest Classifier demonstrates strong effectiveness in prediction of recurrence. To advance this work data augmentation should be implemented and deep learning methods incorporated and model explanations should be improved. Future implementation of these steps will progress the project toward informed clinical decisions that enhance patient results.