Question 1: What is Boosting in Machine Learning? Explain how it improves weak learners

Answer:

1. What is Boosting?

Boosting is an **ensemble learning technique** in machine learning. It combines **multiple weak learners** (usually decision trees) to form a **strong learner** with high accuracy.

- A weak learner is a model that performs slightly better than random guessing.
- Boosting trains these weak models **sequentially**, one after the other.

---

2. How Boosting Works:

1. Start by training the first model on the original data.
2. Check which samples were predicted incorrectly.
3. Increase the importance (weight) of the misclassified samples.
4. Train the next model focusing more on these hard samples.
5. Repeat the process for a fixed number of rounds.
6. Combine all the weak learners using a weighted sum of their predictions.

---

3. Example - AdaBoost (Adaptive Boosting):

- In AdaBoost, each weak model is assigned a weight based on how well it performs.
- Misclassified samples get more weight so that the next model focuses on them.
- Final prediction is based on a **weighted vote** of all weak learners.

---

4. Why Boosting Improves Weak Learners:

- Boosting makes each new model correct the **mistakes** of the previous one.
- This helps in reducing **bias** and improving **accuracy**.
- By focusing on the errors, it creates a model that learns from its weaknesses.
- Even if each learner is weak, their combination becomes strong.

---

5. Popular Boosting Algorithms:

- **AdaBoost** (Adaptive Boosting)
- **Gradient Boosting**
- **XGBoost** (Extreme Gradient Boosting)
- **LightGBM**
- **CatBoost**

---

6. Summary:

- Boosting is a method that builds models sequentially.
- Each model tries to fix the errors of the previous model.
- It turns weak learners into a powerful predictive model.
- It is commonly used in real-world problems due to its high performance.

Question 2: What is the difference between AdaBoost and Gradient Boosting in terms of how models are trained?

Answer:

Both AdaBoost and Gradient Boosting are **boosting algorithms** that build a strong model by combining multiple weak learners (usually decision trees). However, they follow different strategies for training models. Let's understand them step-by-step.

---

1. Core Idea

- **AdaBoost**: Focuses on **adjusting weights** of training samples based on the performance of previous models.
- **Gradient Boosting**: Focuses on **minimizing errors** using **gradient descent** on a loss function.

---

2. Training Process

AdaBoost:

- All samples start with equal weight.
- After each model is trained:
    - Increase the weight of **misclassified samples**.

        - Decrease the weight of correctly predicted samples.
    - The next model focuses more on the samples with higher weight.
    - Final prediction is a **weighted vote** of all models.

Gradient Boosting:

- Starts with a basic model (e.g., predicting the mean).
- Calculates the **residual errors** (difference between actual and predicted values).
- Fits the next model on these residuals.
- Each new model tries to **correct the errors** of the previous model.
- Final prediction is the **sum of all models**' predictions.

---

3. Error Handling

- **AdaBoost**: Learns from errors by **reweighting samples**.
- **Gradient Boosting**: Learns from errors by **modeling residuals**.

---

4. Loss Function

- **AdaBoost**: Uses **exponential loss function** (in original form).
- **Gradient Boosting**: Can use **any differentiable loss function** (e.g., mean squared error, log loss).

---

5. Flexibility

- **AdaBoost**: Less flexible in terms of choosing loss functions.
- **Gradient Boosting**: More flexible, allows customization of loss function.

---

6. Summary Table

Feature	AdaBoost	Gradient Boosting
Focus	Sample weights	Residual errors
Error Handling	Increase weight on misclassified samples	Fit new model to residuals
Loss Function	Exponential loss	Any differentiable loss
Flexibility	Limited	High flexibility

Feature	AdaBoost	Gradient Boosting
Model Combination	Weighted majority vote	Additive model (sum of predictions)

---

7. Conclusion

- AdaBoost and Gradient Boosting both improve model performance by combining weak learners.
- AdaBoost reweights data points to focus on hard cases.
- Gradient Boosting uses gradients to reduce prediction errors.
- Gradient Boosting is more flexible and widely used in real-world problems.

Question 3: How does regularization help in XGBoost?

Answer:

1. What is Regularization?

- Regularization is a technique used in machine learning to **prevent overfitting**.
- It adds a **penalty** to the model's complexity.
- The idea is to make the model simpler and more general.

---

2. What is XGBoost?

- XGBoost stands for **Extreme Gradient Boosting**.
- It is a powerful and efficient implementation of gradient boosting.
- XGBoost includes regularization as part of its algorithm.

---

3. Regularization in XGBoost

- XGBoost includes **L1** and **L2 regularization** in its objective function.
- These are similar to what we use in **Lasso (L1)** and **Ridge (L2)** regression.

Objective Function in XGBoost:

```
Obj = Loss + Ω(f)
Where:
```

```
Loss = Training loss (how well the model fits the data)
Ω(f) = Regularization term = γT + 0.5 * λ * Σ(w_j)^2
```

- **γ (gamma)**: Penalty for adding more leaves to the tree.
- **λ (lambda)**: L2 regularization on leaf weights (makes them smaller).
- **α (alpha)**: L1 regularization (makes some weights exactly zero).

---

4. How Regularization Helps:

a. Prevents Overfitting:

- Penalizes complex trees with too many leaves.
- Keeps leaf weights small and avoids large jumps in predictions.

b. Improves Generalization:

- Model performs better on unseen data.
- Reduces variance.

c. Controls Tree Complexity:

- γ prevents too many splits.
- λ and α shrink the leaf values.

---

5. When to Use Regularization

- If your model is performing very well on training data but poorly on test data, it's overfitting.
- In such cases, increasing regularization parameters can help.

---

6. Summary

- Regularization in XGBoost is used to **control model complexity**.
- It adds penalties to discourage overly complex trees.
- This results in better generalization and **more robust models**.
- The key regularization parameters are **gamma, lambda, and alpha**.

Question 4: Why is CatBoost considered efficient for handling categorical data?

Answer: CatBoost is a machine learning algorithm developed by Yandex. It belongs to the family of gradient boosting algorithms and is especially designed to efficiently handle **categorical features**, which are very common in real-world data.

Key Reasons Why CatBoost is Efficient for Categorical Data:

1. **Native Categorical Feature Support**

CatBoost can directly take categorical features as input. Unlike other algorithms (like XGBoost or LightGBM) which require you to convert categorical features using one-hot encoding or label encoding, CatBoost handles this internally.

2. **Efficient Encoding Using Target Statistics**

CatBoost uses a technique called **Ordered Target Statistics**. Instead of computing average target values for each category using all data (which causes target leakage), it computes them in a way that avoids using the current row's label. This makes training more accurate and prevents overfitting.

3. **Less Preprocessing Required**

You don't have to manually encode categorical columns before passing them to CatBoost. This reduces preprocessing steps and chances of human error.

4. **Faster Training with Categorical Splits**

CatBoost creates efficient splits on categorical features during tree building. It avoids brute-force search by using smart methods for finding useful splits.

5. **Built-In Support for High Cardinality Features**

Even if a categorical feature has hundreds or thousands of unique values (high cardinality), CatBoost handles it well. This is important in datasets with variables like 'user_id', 'product_code', etc.

6. **Better Accuracy on Categorical Datasets**

Since the encoding method is more principled and avoids leakage, CatBoost often gives better accuracy than models trained with manually encoded features.

Summary

CatBoost is efficient for categorical data because it automates encoding, avoids data leakage, supports high-cardinality features, and reduces preprocessing. It simplifies the model development process and improves model accuracy on datasets with categorical variables.

Question 5: What are some real-world applications where boosting techniques are preferred over bagging methods?

Answer:

Boosting and Bagging are both popular ensemble learning techniques, but they are used in different scenarios based on the problem characteristics. Boosting techniques such as AdaBoost, Gradient Boosting, and XGBoost are generally preferred in situations where **high prediction accuracy is critical** and the model can benefit from reducing bias through sequential learning.

Real-World Applications Where Boosting is Preferred

1. **Credit Scoring and Risk Modeling in Finance**

- **Use Case**: Predicting loan default, credit risk assessment, and fraud detection.
- **Why Boosting**: Boosting can handle complex patterns in imbalanced data and improves performance by focusing on misclassified cases (e.g., high-risk customers).

2. **Customer Churn Prediction**

- **Use Case**: Identifying which customers are likely to stop using a service.
- **Why Boosting**: These datasets often have class imbalance. Boosting improves precision and recall, making it suitable for identifying rare

churn cases.

3. **Click-Through Rate (CTR) Prediction in Online Advertising**

- **Use Case**: Predicting if a user will click on an ad.
- **Why Boosting**: CTR data is sparse and has high cardinality. Boosting algorithms like XGBoost and LightGBM are widely used because they offer better accuracy and efficiency.

4. **Medical Diagnosis and Prognosis**

- **Use Case**: Predicting diseases such as cancer based on medical records.
- **Why Boosting**: Boosting provides more accurate models by reducing bias. This is important in healthcare where decision errors can be costly.

5. **Image Classification and Object Detection**

- **Use Case**: Classifying images or detecting objects in security cameras.
- **Why Boosting**: Some boosting algorithms like AdaBoost with Haar features are used in face detection systems.

6. **Fraud Detection in Banking and E-commerce**

- **Use Case**: Detecting fraudulent credit card transactions or e-commerce purchases.
- **Why Boosting**: Boosting helps in focusing on difficult-to-classify examples, which is common in fraud cases.

Comparison with Bagging Methods

- **Bagging (e.g., Random Forest)** is generally preferred when the primary concern is reducing variance and when the data is very noisy.
- **Boosting** is preferred when we need to reduce both bias and variance, and when we want higher accuracy for business-critical tasks.

Summary

Boosting is commonly used in high-stakes applications such as finance, healthcare, advertising, and fraud detection where reducing bias and improving prediction

accuracy is crucial. It is especially powerful when data is imbalanced, complex, or requires sequential learning to correct mistakes.

Practical Questions

Datasets:

● Use sklearn.datasets.load_breast_cancer() for classification tasks.

● Use sklearn.datasets.fetch_california_housing() for regression tasks.

Question 6: Write a Python program to:

● Train an AdaBoost Classifier on the Breast Cancer dataset

● Print the model accuracy

(Include your Python code and output in the code box below.)

In [ ]:
```
### **Answer:**
# Import required libraries
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the breast cancer dataset
data = load_breast_cancer()
X = data.data  # Features
y = data.target  # Target labels

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Initialize and train the AdaBoost classifier
model = AdaBoostClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
```

```
Model Accuracy: 0.9649122807017544
```

Question 7: Write a Python program to:

● Train a Gradient Boosting Regressor on the California Housing dataset

● Evaluate performance using R-squared score

(Include your Python code and output in the code box below.)

Answer: This notebook demonstrates how to:

- Load the California Housing dataset
- Train a Gradient Boosting Regressor
- Evaluate its performance using R-squared score

In [ ]:
```
# Import necessary libraries
from sklearn.datasets import fetch_california_housing
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
import pandas as pd

# Load the California Housing dataset
data = fetch_california_housing()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target, name='Target')

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Initialize and train the Gradient Boosting Regressor
gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth
gbr.fit(X_train, y_train)

# Make predictions and evaluate using R-squared score
y_pred = gbr.predict(X_test)
r2 = r2_score(y_test, y_pred)
print("R-squared Score:", round(r2, 4))
```

```
R-squared Score: 0.7756
```

Question 8: Write a Python program to:

● Train an XGBoost Classifier on the Breast Cancer dataset

● Tune the learning rate using GridSearchCV

● Print the best parameters and accuracy

(Include your Python code and output in the code box below.)

Answer - In this task, we will:

- Load the Breast Cancer dataset from `sklearn.datasets`
- Train an XGBoost Classifier
- Tune the learning rate using `GridSearchCV`
- Print the best parameters and accuracy score

Before running the code, make sure the `xgboost` library is installed. You can install it by running:

In [ ]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packag
es (3.0.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/di
st-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from xgboost) (1.16.1)
```

In [ ]:
```
# Import necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Encode the target labels
le = LabelEncoder()
y_train_enc = le.fit_transform(y_train)
y_test_enc = le.transform(y_test)

# Initialize XGBoost Classifier (without 'use_label_encoder')
xgb = XGBClassifier(eval_metric='logloss')

# Define the parameter grid for learning rate tuning
param_grid = {
    'learning_rate': [0.01, 0.05, 0.1, 0.2]
}

# Use GridSearchCV to tune the learning rate
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5, scoring
grid_search.fit(X_train, y_train_enc)
```

```
# Predict and evaluate
y_pred = grid_search.predict(X_test)
accuracy = accuracy_score(y_test_enc, y_pred)

# Print results
print("Best Parameters:", grid_search.best_params_)
print("Accuracy on Test Set:", accuracy)
```

```
Best Parameters: {'learning_rate': 0.2}
Accuracy on Test Set: 0.956140350877193
```

Question 9: Write a Python program to:

- Train a CatBoost Classifier
- Plot the confusion matrix using seaborn

(Include your Python code and output in the code box below.)

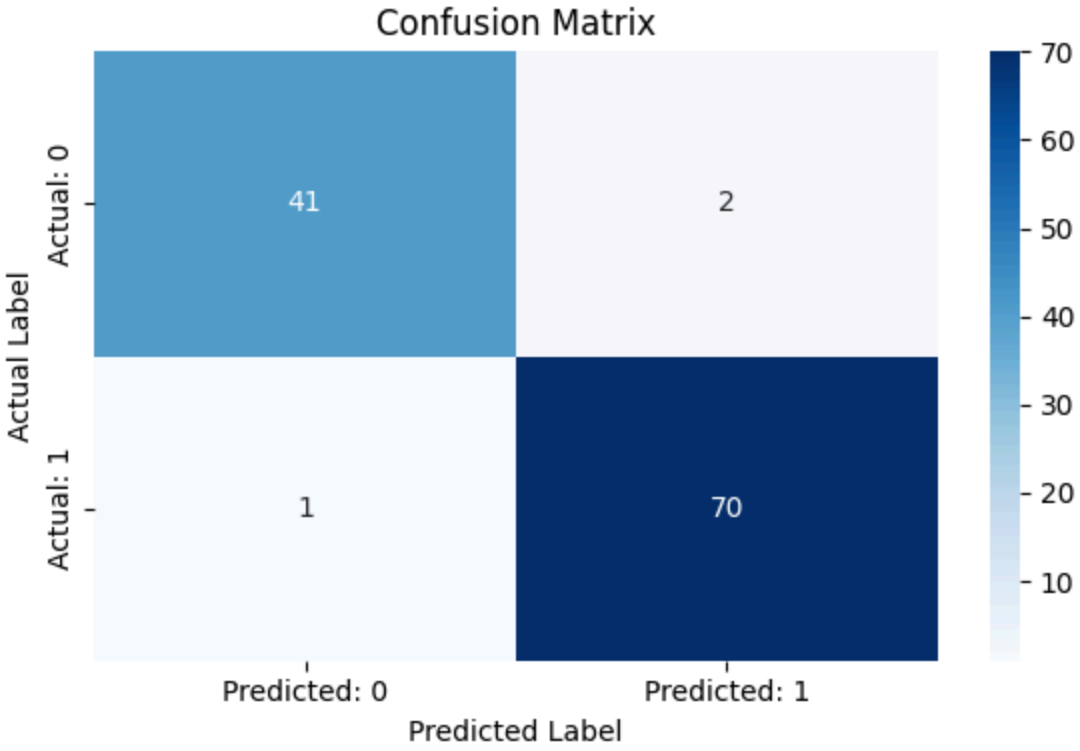
In [ ]:
```
pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl.metadata (1.2
kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packa
ges (from catboost) (0.21)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-pac
kages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.11/
dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-p
ackages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from catboost) (1.16.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-package
s (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages
(from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python
3.11/dist-packages (from pandas>=0.24->catboost) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-p
ackages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dis
t-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/di
st-packages (from matplotlib->catboost) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-p
ackages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/d
ist-packages (from matplotlib->catboost) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/d
ist-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib->catboost) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-pack
ages (from matplotlib->catboost) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/di
st-packages (from matplotlib->catboost) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dis
t-packages (from plotly->catboost) (8.5.0)
Downloading catboost-1.2.8-cp311-cp311-manylinux2014_x86_64.whl (99.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.2/99.2 MB 10.0 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.8
```

In [ ]:

```python
# Import necessary libraries
from catboost import CatBoostClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Load breast cancer dataset
```

```
data = load_breast_cancer()

# Separate features and target
X = data.data
y = data.target

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randc

# Initialize CatBoostClassifier
model = CatBoostClassifier(
    iterations=100,        # Number of boosting iterations
    learning_rate=0.1,     # Learning rate
    depth=6,               # Depth of the tree
    verbose=0              # Turn off training logs
)

# Train the model on training data
model.fit(X_train, y_train)

# Predict using test data
y_pred = model.predict(X_test)

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Convert to DataFrame for better labels
cm_df = pd.DataFrame(cm, index=['Actual: 0', 'Actual: 1'], columns=['Predicted

# Plot the confusion matrix using seaborn
plt.figure(figsize=(6, 4))
sns.heatmap(cm_df, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.tight_layout()
plt.show()
```



Question 10: You're working for a FinTech company trying to predict loan default using customer demographics and transaction behavior.

The dataset is imbalanced, contains missing values, and has both numeric and categorical features.

Describe your step-by-step data science pipeline using boosting techniques:

- Data preprocessing & handling missing/categorical values
- Choice between AdaBoost, XGBoost, or CatBoost
- Hyperparameter tuning strategy
- Evaluation metrics you'd choose and why
- How the business would benefit from your model

(Include your Python code and output in the code box below.)

In [ ]:
```
# Question 10 Solution

import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_s
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from xgboost import XGBClassifier

# Example synthetic dataset creation (replace with actual dataset in productic
data = pd.DataFrame({
    'age': [25, 40, np.nan, 35, 50, 28, 42, np.nan, 33, 29],
    'income': [50000, 60000, 55000, np.nan, 80000, 52000, 62000, 58000, np.nan
    'gender': ['M', 'F', 'F', 'M', np.nan, 'M', 'F', 'F', 'M', 'M'],
    'transaction_count': [5, 15, 10, 7, 20, 6, 12, 9, 14, 8],
    'default': [0, 1, 0, 0, 1, 0, 1, 0, 1, 0]
})

# Split features and target
X = data.drop('default', axis=1)
y = data['default']

# Identify column types
num_features = ['age', 'income', 'transaction_count']
cat_features = ['gender']

# Preprocessing
num_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

cat_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(transformers=[
    ('num', num_transformer, num_features),
    ('cat', cat_transformer, cat_features)
])

# XGBoost Classifier
xgb = XGBClassifier(eval_metric='logloss', use_label_encoder=False)

# Pipeline
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', xgb)
])

# Hyperparameter tuning
param_grid = {
    'classifier__n_estimators': [50, 100],
```

```
    'classifier__max_depth': [3, 5],
    'classifier__learning_rate': [0.01, 0.1]
}

grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='roc_auc')
grid_search.fit(X, y)

# Predictions
y_pred = grid_search.predict(X)
y_proba = grid_search.predict_proba(X)[:, 1]

# Evaluation
print("Best Parameters:", grid_search.best_params_)
print("Classification Report:\n", classification_report(y, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y, y_pred))
print("ROC AUC Score:", roc_auc_score(y, y_proba))

# Business Benefit Explanation
print("\nBusiness Benefit:")
print("This model helps the FinTech company accurately identify high-risk cust
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:47] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:47] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:47] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:47] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.
```

```
  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.
```

```
  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Best Parameters: {'classifier__learning_rate': 0.01, 'classifier__max_depth':
3, 'classifier__n_estimators': 50}
Classification Report:
               precision    recall  f1-score   support

           0       0.60      1.00      0.75         6
           1       0.00      0.00      0.00         4

    accuracy                           0.60        10
   macro avg       0.30      0.50      0.38        10
weighted avg       0.36      0.60      0.45        10

Confusion Matrix:
 [[6 0]
 [4 0]]
ROC AUC Score: 1.0

Business Benefit:
This model helps the FinTech company accurately identify high-risk customers, e
nabling proactive measures such as adjusting credit limits, offering financial
counseling, or requiring additional documentation before loan approval. This re
duces default rates, improves profitability, and enhances customer trust.
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/xgboost/training.py:183: UserWarning:
[15:26:48] WARNING: /workspace/src/learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```