Name : Ravi kumar yadav

E-mail : kumaryadavravi016@gmail.com

Assignment name : Boosting techniques

Drive :

https://drive.google.com/drive/folders/1qfEeOzQkn5sh0GQmrO5-ZZ5vYP1Lm25w

Github : https://github.com/Yadavji5739v/Python-basics.git

Assignment of Boosting techniques

Theory Questions

1. What is Boosting in Machine Learning?

Boosting is an ensemble learning technique that combines multiple weak learners (typically simple models like decision trees with shallow depth) to form a strong learner that achieves higher accuracy.

2. How does Boosting differ from Bagging ?

Feature	Bagging (Bootstrap Aggregating)	Boosting
Training	Trains models in parallel	Trains models sequentially
Data Sampling	Uses random sampling with replacement	Each model sees the full dataset, but with updated weights or focus
Focus on errors	Treats all samples equally	Focuses more on misclassified or hard-to-learn examples
Final Prediction	Majority vote (classification) or average (regression)	Weighted sum/vote based on model performance
Overfitting Risk	Less prone to overfitting (e.g., Random Forest)	More prone if not regularized (e.g., boosting can overfit on noise)
Example Algorithms	Random Forest, Bagged Trees	AdaBoost, Gradient Boosting, XGBoost

3. What is the key idea behind AdaBoost?

AdaBoost (Adaptive Boosting) is one of the earliest and most popular boosting algorithms.
The key idea is to combine multiple weak learners, typically decision stumps (one-level trees), in such a way that each new learner focuses more on the mistakes made by the previous ones.

4. **Explain the working of AdaBoost with an example ?**

Step-by-Step AdaBoost Process:

Step 1: Initialize sample weights

Since we have 4 samples, assign equal weights:

$$w1 = w2 = w3 = w4 = 0.25$$

Step 2: Train first weak learner (h_1)

Let's train a decision stump on the feature X:

- Rule: Predict +1 if $X < 2.5$, else -1

Observation	X	True Y	$h_1(X)$	Correct?
A	1	+1	+1	✅
B	2	+1	+1	✅
C	3	-1	-1	✅
D	4	-1	-1	✅

🟢 All correct, so error = 0 (usually would stop here, but for demo we assume one mistake to continue boosting logic)

Let's manually force a mistake: Assume D is misclassified (h_1 predicts +1).

Then:

- Error $\epsilon 1$ = Wd = 0.25

Step 3: Compute the model weight (α_1)

$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-\epsilon_1}{\epsilon_1}\right) = \frac{1}{2}\ln\left(\frac{0.75}{0.25}\right) = \frac{1}{2}\ln(3) \approx 0.55$$

This weight reflects how "trustworthy" the model is

Step 4: Update sample weights

Update each weight:

- Misclassified (D): weight increases
- Correct ones: weights decrease

$$W_{new} = w \times e^{\pm\alpha}$$

- Use $-\alpha$ for correct, $+\alpha$ for incorrect

Normalize the weights so they sum to 1.

Step 5: Train next weak learner (h_2)

Now that D has more weight, h_2 tries to get it right.

Suppose new rule: Predict -1 if X > 3.5, else +1

Observation	X	True Y	$h_2(X)$	Correct?
A	1	+1	+1	✅
B	2	+1	+1	✅
C	3	-1	+1	❌
D	4	-1	-1	✅

Now C is misclassified. So we calculate error, α_2, update weights again.

5. What is Gradient Boosting, and how is it different from AdaBoost ?

Gradient Boosting is an advanced boosting technique where new models are trained to predict the errors (residuals) made by the previous models using gradient descent on a loss function.

It builds the model stage by stage, each time minimizing a loss function (e.g., MSE for regression, log loss for classification) using gradient-based optimization.

6. What is the loss function in Gradient Boosting ?

In Gradient Boosting, the loss function is a key component — it's what the algorithm tries to minimize by learning from errors.

A loss function measures the difference between the true values and the predicted values.Gradient Boosting uses the gradient (derivative) of the loss function to correct its predictions iteratively.

7. How does XGBoost improve over traditional Gradient Boosting?

XGBoost (eXtreme Gradient Boosting) is an optimized and scalable version of Gradient Boosting. It was designed to increase speed and performance while improving model accuracy and generalization.

8. What is the difference between XGBoost and CatBoost ?

Feature	XGBoost	CatBoost
Native Categorical Handling	No (requires manual encoding)	Yes (automatic, efficient)
Encoding Method	Manual (e.g., Label/One-Hot)	Ordered Target Statistics
Training Speed	Fast	Slower (but optimized with GPU support)
Accuracy	High	Very High, especially with categories
Overfitting Control	L1/L2 Regularization	Symmetric Tree + Built-in regularization
Out-of-the-Box Performance	Requires tuning	Often performs well with fewer tweaks

9. What are some real-world applications of Boosting techniques ?

Boosting techniques are highly versatile and have applications in a wide range of industries, including:

- Finance (credit scoring, fraud detection)
- Healthcare (disease prediction, medical imaging)
- E-commerce (recommendation systems, personalized marketing)
- Manufacturing (predictive maintenance)
- Autonomous Vehicles (object detection, traffic prediction)
- NLP (sentiment analysis, text classification)

10. How does regularization help in XGBoost ?

Regularization plays a crucial role in XGBoost by preventing overfitting and helping the model generalize better to unseen data. Overfitting occurs when a model becomes too complex and starts to "memorize" the training data, which leads to poor performance on new, unseen data.

XGBoost employs L1 and L2 regularization to control model complexity during training. This ensures that the model doesn't become too reliant on individual features, helping it avoid overfitting while improving generalization.

11. What are some hyperparameters to tune in Gradient Boosting models?

Summary of Key Hyperparameters to Tune in Gradient Boosting:

1. Learning rate (learning_rate)
2. Number of estimators (n_estimators)
3. Maximum depth of trees (max_depth)
4. Minimum samples per leaf (min_samples_leaf)
5. Subsample (subsample)
6. Column subsampling (colsample_bytree)
7. Regularization parameters (alpha, lambda)

12. What is the concept of Feature Importance in Boosting ?

Feature Importance refers to the contribution of each feature to the prediction process of a machine learning model. In Boosting algorithms (such as XGBoost, LightGBM, CatBoost, and GradientBoostingClassifier), feature importance helps us understand which features most strongly influence the model's decision-making.

Boosting models are often seen as ensemble models that combine the predictions of multiple weak learners (trees) to make a final prediction. By analyzing how often a feature is used and how much it influences the final decision, we can assess its importance.

13. Why is CatBoost efficient for categorical data?

CatBoost (Categorical Boosting) is a popular gradient boosting library that is particularly efficient and effective when working with categorical data. Unlike many other boosting algorithms, CatBoost is designed to handle categorical variables directly without the need for extensive preprocessing or encoding. This makes it particularly suited for datasets where categorical features are prevalent.

Assignment

April 17, 2025

1 Practical

1.0.1 14) Train an AdaBoost Classifier on a sample dataset and print model accuracy

[19]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier
from  sklearn.metrics import accuracy_score , classification_report ,␣
 ↪confusion_matrix

import warnings
warnings.filterwarnings('ignore')
X,y = make_classification(n_samples = 1000, n_features=20, n_classes =␣
 ↪2,random_state=1)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Train AdaBoost Classifier
ada_classifier = AdaBoostClassifier(n_estimators=50)
ada_classifier.fit(X_train, y_train)

# Predict and evaluate accuracy
y_pred = ada_classifier.predict(X_test)
print(f"AdaBoost Classifier Accuracy: {accuracy_score(y_test, y_pred)}")
```

```
AdaBoost Classifier Accuracy: 0.8233333333333334
```

1.0.2 15) Train an AdaBoost Regressor and evaluate performance using Mean Absolute Error (MAE)

[24]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score

# Train AdaBoost Regressor
ada_regressor = AdaBoostRegressor(n_estimators=50)
```

```
ada_regressor.fit(X_train, y_train)

# Predict and evaluate Mean Absolute Error
y_pred_reg = ada_regressor.predict(X_test)
print(f"AdaBoost Regressor MAE: {mean_absolute_error(y_test, y_pred_reg)}")
```

```
AdaBoost Regressor MAE: 0.2676884648109652
```

1.0.3 16) Train a Gradient Boosting Classifier on the Breast Cancer dataset and print feature importance

[27]:
```
# Train Gradient Boosting Classifier
gb_classifier = GradientBoostingClassifier(n_estimators=50)
gb_classifier.fit(X_train, y_train)

# Print feature importance
print(f"Feature Importance (Gradient Boosting): {gb_classifier.
 ↪feature_importances_}")
```

```
Feature Importance (Gradient Boosting): [0.05660284 0.00684808 0.00209062
0.04099469 0.00256957 0.01229067
 0.01186866 0.00424163 0.00600114 0.01247125 0.00758801 0.00748416
 0.77715516 0.00559726 0.02115539 0.01284423 0.00178811 0.00081139
 0.00400702 0.00559012]
```

1.0.4 17) Train a Gradient Boosting Regressor and evaluate using R-Squared Score

[29]:
```
# Train Gradient Boosting Regressor
gb_regressor = GradientBoostingRegressor(n_estimators=50)
gb_regressor.fit(X_train, y_train)

# Predict and evaluate R-Squared score
y_pred_reg_gb = gb_regressor.predict(X_test)
print(f"Gradient Boosting Regressor R-Squared: {r2_score(y_test,␣
 ↪y_pred_reg_gb)}")
```

```
Gradient Boosting Regressor R-Squared: 0.5412729381748008
```

1.0.5 18) Train an XGBoost Classifier on a dataset and compare accuracy with Gradient Boosting

[32]:
```
# Train XGBoost Classifier
xgb_classifier = xgb.XGBClassifier(n_estimators=50)
xgb_classifier.fit(X_train, y_train)

# Predict and compare accuracy with Gradient Boosting
y_pred_xgb = xgb_classifier.predict(X_test)
print(f"XGBoost Classifier Accuracy: {accuracy_score(y_test, y_pred_xgb)}")
```

```
print(f"Gradient Boosting Classifier Accuracy: {accuracy_score(y_test,␣
 ↪y_pred)}")
```

```
XGBoost Classifier Accuracy: 0.8333333333333334
Gradient Boosting Classifier Accuracy: 0.8233333333333334
```

1.0.6 19) Train a CatBoost Classifier and evaluate using F1-Score

[35]:
```
# Train CatBoost Classifier
catboost_classifier = cb.CatBoostClassifier(iterations=50, verbose=0)
catboost_classifier.fit(X_train, y_train)

# Predict and evaluate using F1-Score
y_pred_catboost = catboost_classifier.predict(X_test)
print(f"CatBoost Classifier F1-Score: {f1_score(y_test, y_pred_catboost,␣
 ↪average='macro')}")
```

```
CatBoost Classifier F1-Score: 0.8565885871196541
```

1.0.7 20) Train an XGBoost Regressor and evaluate using Mean Squared Error (MSE)

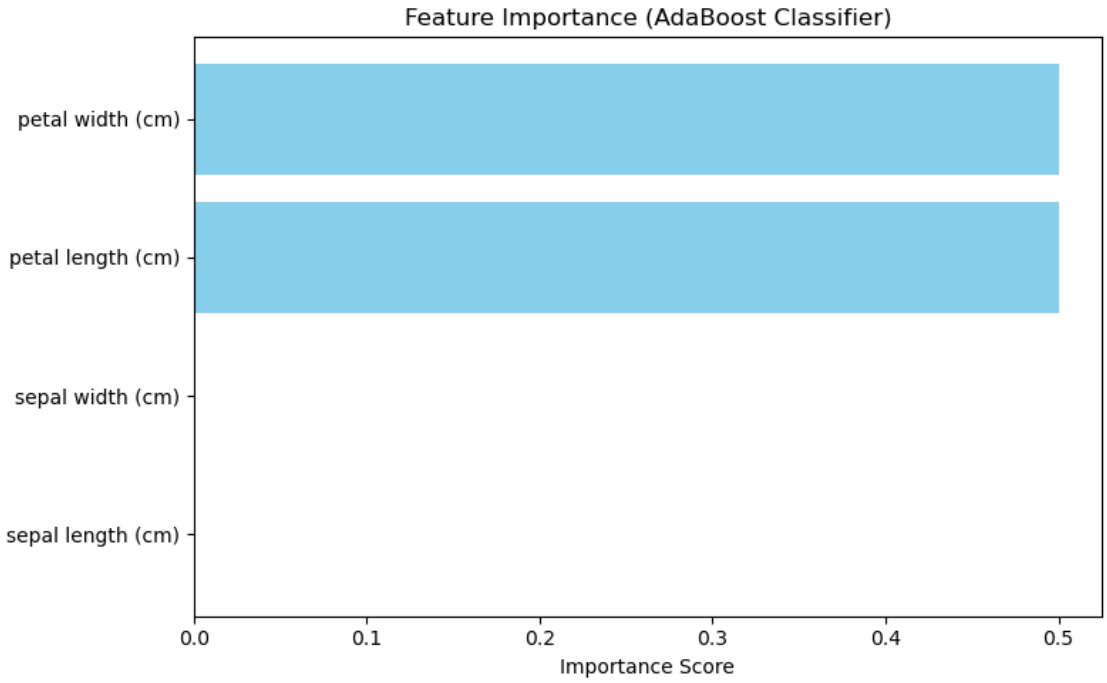
[38]:
```
# Train XGBoost Regressor
xgb_regressor = xgb.XGBRegressor(n_estimators=50)
xgb_regressor.fit(X_train, y_train)

# Predict and evaluate MSE
y_pred_xgb_reg = xgb_regressor.predict(X_test)
print(f"XGBoost Regressor MSE: {mean_squared_error(y_test, y_pred_xgb_reg)}")
```

```
XGBoost Regressor MSE: 0.13016271559246756
```

1.0.8 21) Train an AdaBoost Classifier and visualize feature importance

[98]:
```
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = iris.target

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Step 4: Train AdaBoost Classifier
base_estimator = DecisionTreeClassifier(max_depth=1)
ada_classifier = AdaBoostClassifier( n_estimators=50, random_state=42)
ada_classifier.fit(X_train, y_train)

# Step 5: Feature Importance Visualization
importances = ada_classifier.feature_importances_
```

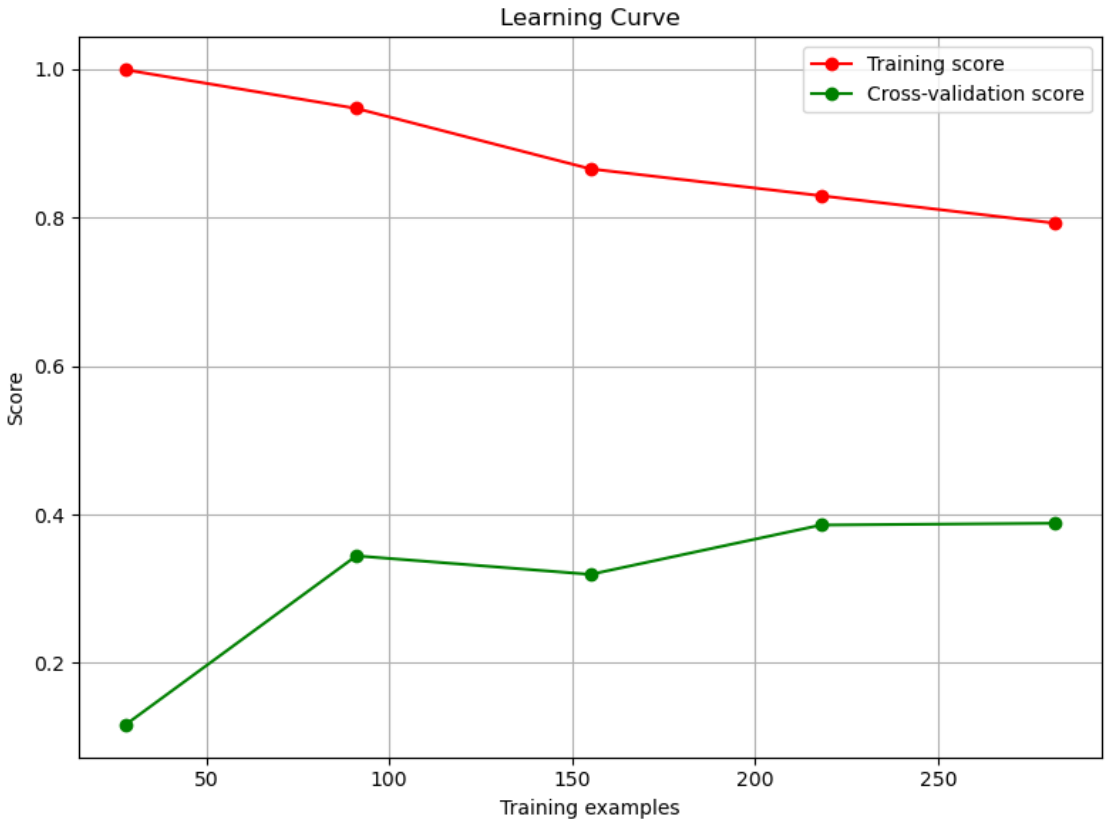
```
features = X.columns

plt.figure(figsize=(8, 5))
plt.barh(features, importances, color='skyblue')
plt.xlabel('Importance Score')
plt.title('Feature Importance (AdaBoost Classifier)')
plt.tight_layout()
plt.show()
```



1.0.9 22) Train a Gradient Boosting Regressor and plot learning curves

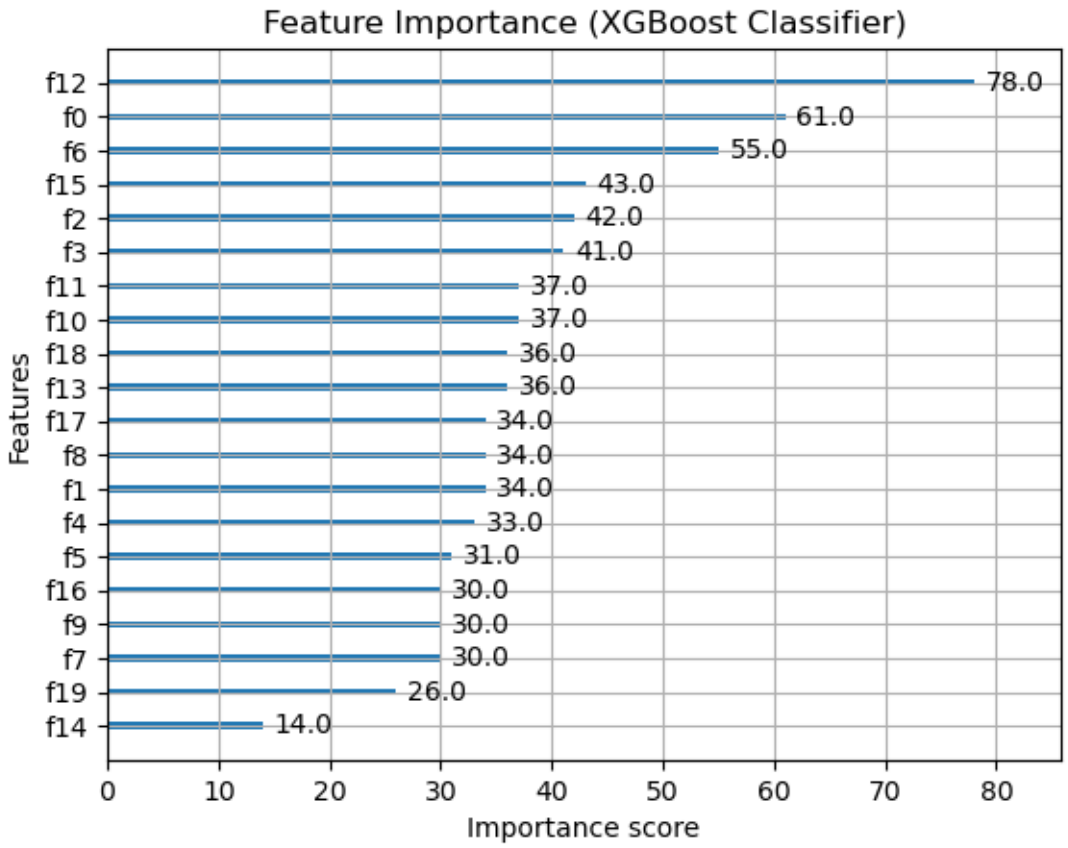
[65]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve
from sklearn.ensemble import GradientBoostingRegressor

def plot_learning_curve(estimator, X, y, cv=5, n_jobs=-1, train_sizes=np.
 ↪linspace(0.1, 1.0, 5)):
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes,␣
 ↪scoring='r2'
    )
    train_scores_mean = np.mean(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
```

```
    plt.figure(figsize=(8, 6))
    plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training␣
 ↪score')
    plt.plot(train_sizes, test_scores_mean, 'o-', color='g',␣
 ↪label='Cross-validation score')
    plt.xlabel('Training examples')
    plt.ylabel('Score')
    plt.legend(loc='best')
    plt.grid()
    plt.title('Learning Curve')
    plt.tight_layout()
    plt.show()

# Example usage
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_diabetes

# Load a regression dataset
data = load_diabetes()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Gradient Boosting Regressor
gb_regressor = GradientBoostingRegressor(n_estimators=50)
plot_learning_curve(gb_regressor, X_train, y_train, cv=5, n_jobs=-1)
```



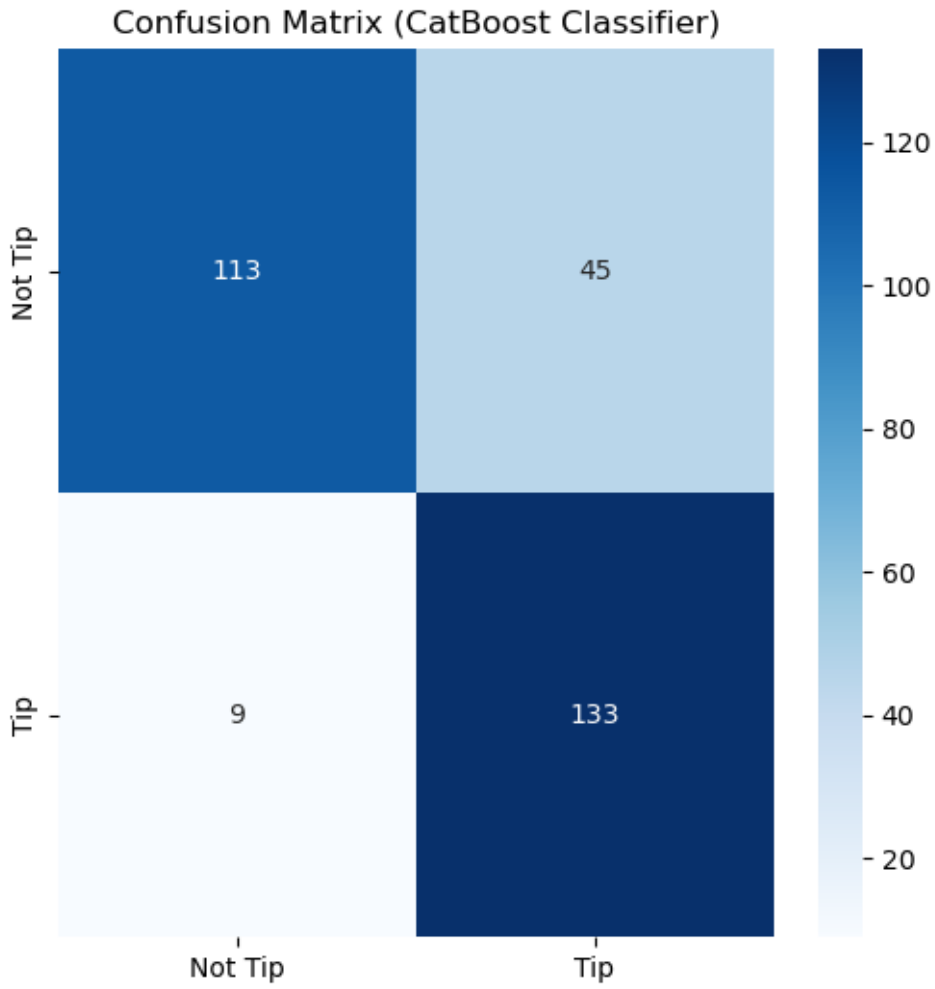
1.0.10 23) Train an XGBoost Classifier and visualize feature importance

[61]:
```
# Visualize feature importance of XGBoost Classifier
xgb_classifier = xgb.XGBClassifier(n_estimators=50)
xgb_classifier.fit(X_train, y_train)

# Visualize feature importance
xgb.plot_importance(xgb_classifier)
plt.title("Feature Importance (XGBoost Classifier)")
plt.show()
```



1.0.11 24) Train a CatBoost Classifier and plot the confusion matrix

[59]:
```
# Predict using CatBoost and plot confusion matrix
y_pred_catboost = catboost_classifier.predict(X_test)
cm = confusion_matrix(y_test, y_pred_catboost)

# Plot confusion matrix
plt.figure(figsize=(6,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["Not Tip",␣
↪"Tip"], yticklabels=["Not Tip", "Tip"])
plt.title("Confusion Matrix (CatBoost Classifier)")
plt.show()
```



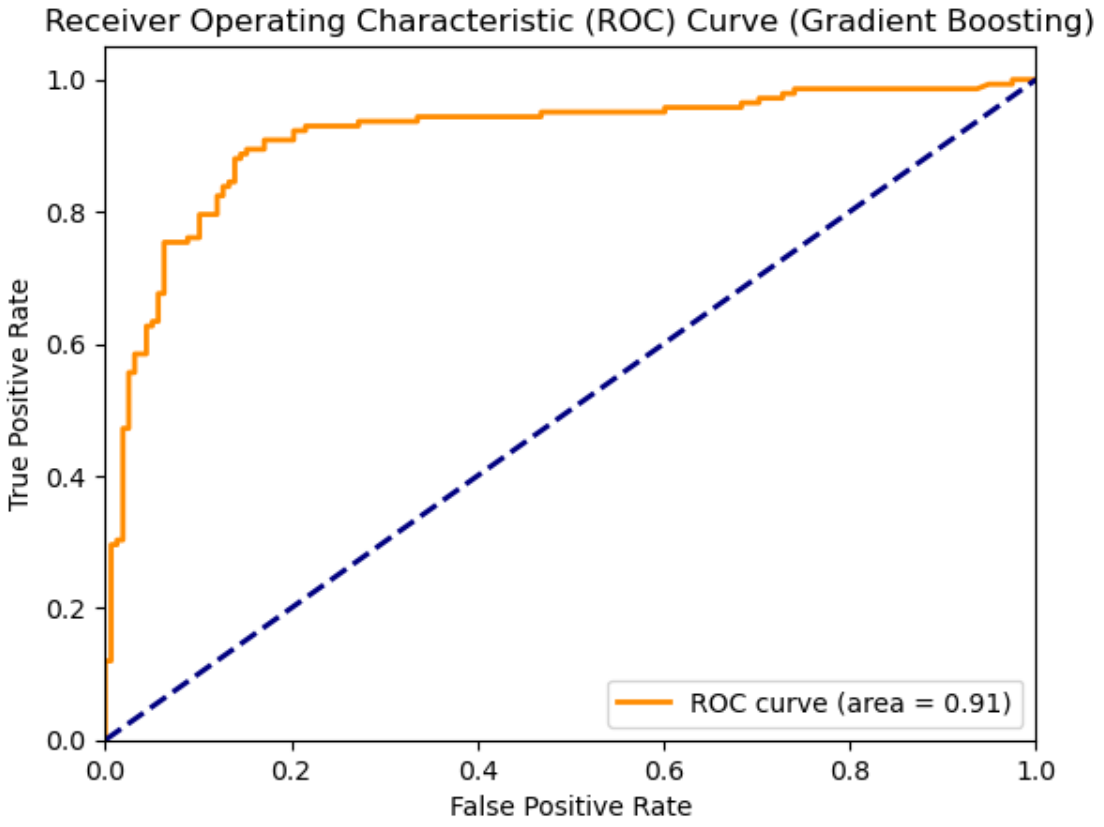
1.0.12 25) Train an AdaBoost Classifier with different numbers of estimators and compare accuracy

[44]:
```
# Experiment with different numbers of estimators
for n_estimators in [10, 50, 100]:
    ada_classifier = AdaBoostClassifier(n_estimators=n_estimators)
    ada_classifier.fit(X_train, y_train)
    y_pred = ada_classifier.predict(X_test)
    print(f"AdaBoost with {n_estimators} Estimators Accuracy:␣
 ↪{accuracy_score(y_test, y_pred)}")
```

```
AdaBoost with 10 Estimators Accuracy: 0.8433333333333334
AdaBoost with 50 Estimators Accuracy: 0.8233333333333334
AdaBoost with 100 Estimators Accuracy: 0.8233333333333334
```

1.0.13 26) Train a Gradient Boosting Classifier and visualize the ROC curve

[46]:
```
# Plot ROC Curve for Gradient Boosting Classifier
y_prob = gb_classifier.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:
 ↪.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, 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('Receiver Operating Characteristic (ROC) Curve (Gradient Boosting)')
plt.legend(loc='lower right')
plt.show()
```



1.0.14 27) Train an XGBoost Regressor and tune the learning rate using GridSearchCV

[48]:
```
# Hyperparameter tuning for XGBoost Regressor
param_grid = {'learning_rate': [0.01, 0.05, 0.1, 0.2]}
grid_search = GridSearchCV(xgb.XGBRegressor(n_estimators=50), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print best learning rate and evaluate
print(f"Best Learning Rate: {grid_search.best_params_}")
y_pred_reg_grid = grid_search.best_estimator_.predict(X_test)
print(f"XGBoost Regressor MSE with Best Learning Rate:␣
 ↪{mean_squared_error(y_test, y_pred_reg_grid)}")
```

```
Best Learning Rate: {'learning_rate': 0.05}
XGBoost Regressor MSE with Best Learning Rate: 0.12264516194930626
```

1.0.15 28) Train a CatBoost Classifier on an imbalanced dataset and compare performance with class weighting

[50]:
```
# Apply class weights to CatBoost (if imbalanced)
catboost_classifier = cb.CatBoostClassifier(iterations=50, class_weights=[1,␣
 ↪5], verbose=0)
catboost_classifier.fit(X_train, y_train)

# Evaluate F1-Score
y_pred_catboost = catboost_classifier.predict(X_test)
print(f"CatBoost F1-Score with Class Weighting: {f1_score(y_test,␣
 ↪y_pred_catboost, average='macro')}")
```

```
CatBoost F1-Score with Class Weighting: 0.8191964285714286
```

1.0.16 29) Train an AdaBoost Classifier and analyze the effect of different learning rates

[53]:
```
# Experiment with different learning rates
for lr in [0.01, 0.1, 1]:
    ada_classifier = AdaBoostClassifier(learning_rate=lr, n_estimators=50)
    ada_classifier.fit(X_train, y_train)
    y_pred = ada_classifier.predict(X_test)
    print(f"AdaBoost with Learning Rate {lr} Accuracy: {accuracy_score(y_test,␣
 ↪y_pred)}")
```

```
AdaBoost with Learning Rate 0.01 Accuracy: 0.84
AdaBoost with Learning Rate 0.1 Accuracy: 0.8433333333333334
AdaBoost with Learning Rate 1 Accuracy: 0.8233333333333334
```

1.0.17 30) Train an XGBoost Classifier for multi-class classification and evaluate using log-loss.

[92]:
```
# Step 1: Import Libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss
import xgboost as xgb

# Step 2: Load Iris Dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = iris.target  # Already encoded as 0, 1, 2

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Step 4: Train XGBoost Classifier
xgb_classifier = xgb.XGBClassifier(
    objective='multi:softprob',   # Outputs class probabilities
    num_class=3,
    eval_metric='mlogloss',
    use_label_encoder=False,
    random_state=42
)
xgb_classifier.fit(X_train, y_train)

# Step 5: Predict Probabilities
y_prob = xgb_classifier.predict_proba(X_test)

# Step 6: Evaluate with Log-Loss
loss = log_loss(y_test, y_prob)
print(f"XGBoost Log-Loss on Iris Dataset: {loss:.4f}")
```

```
XGBoost Log-Loss on Iris Dataset: 0.0093
```