Theoretical

In [ ]:
```
#1. What is Boosting in Machine Learning?
'''
Boosting is an ensemble learning technique that improves prediction accuracy by com
Each model corrects the errors of the previous one by focusing more on the misclass
the final model more robust. Boosting is widely used in both classification and reg
performance on complex datasets. Some popular Boosting algorithms include AdaBoost,
'''

#2. How does Boosting differ from Bagging?
'''
Bagging and Boosting are both ensemble techniques, but they work differently. Baggi
parallel using bootstrapped datasets and averages their predictions to reduce varia
models sequentially, where each model learns from the mistakes of the previous one,
reducing overfitting, while Boosting is useful for improving accuracy on complex da
algorithm, whereas AdaBoost and Gradient Boosting are examples of Boosting.
'''

#3. What is the key idea behind AdaBoost?
'''
AdaBoost (Adaptive Boosting) is a Boosting algorithm that combines multiple weak le
assigns higher weights to misclassified samples, making subsequent models focus mor
iteration, a new weak learner is trained on the updated weights, and their predicti
AdaBoost is mainly used with decision stumps (one-level decision trees) but can be
widely used in face detection, fraud detection, and text classification.
'''

#4. Explain the working of AdaBoost with an example.
'''
AdaBoost starts by assigning equal weights to all training samples and training a w
If a sample is misclassified, its weight is increased so that the next weak classif
is repeated for several iterations, with each new classifier improving upon the err
classifiers' predictions are combined using weighted voting. For example, in a spam
iteratively refine its decision rules, focusing on emails that were previously misc
'''

#5. What is Gradient Boosting, and how is it different from AdaBoost?
'''
Gradient Boosting is an advanced Boosting technique that builds models sequentially
Instead of adjusting sample weights, it minimizes the residual errors using gradien
predict the residuals (errors) of the previous model, gradually improving accuracy.
to training samples, Gradient Boosting directly optimizes the loss function. This m
effective, especially for regression tasks and high-dimensional datasets.
'''

#6. What is the loss function in Gradient Boosting?
'''
Gradient Boosting minimizes a specified loss function to improve predictions. For r
include Mean Squared Error (MSE) and Mean Absolute Error (MAE), which measure diffe
For classification, Log Loss (Cross-Entropy) is commonly used, ensuring that the pr
loss function guides how new weak learners are added to the model. By minimizing th
Boosting effectively reduces errors and enhances model accuracy.
'''
```

```
#7. How does XGBoost improve over traditional Gradient Boosting?
'''
XGBoost (Extreme Gradient Boosting) enhances traditional Gradient Boosting by intro
parallel processing to speed up training and implements L1/L2 regularization to pre
more efficient tree-pruning strategy, handling missing values automatically and all
better feature selection. XGBoost is highly scalable, making it a preferred choice
datasets. It is widely used in applications like recommendation systems, finance, a
'''

#8. What is the difference between XGBoost and CatBoost?
'''
XGBoost and CatBoost are both Gradient Boosting-based algorithms, but they handle c
requires categorical variables to be manually encoded using one-hot encoding or lab
designed specifically for categorical data and uses Ordered Target Encoding, which
efficiency. CatBoost also incorporates a unique ordered boosting technique, which h
CatBoost is often preferred for datasets with many categorical features, such as cu
'''

#9. What are some real-world applications of Boosting techniques?
'''
Boosting techniques are widely used across various domains due to their high accura
for credit scoring and fraud detection. In healthcare, they help in disease predict
e-commerce and recommendation systems, Boosting models improve product recommendati
analysis and spam detection. Boosting is also popular in autonomous driving, where
making.
'''

#10. How does regularization help in XGBoost?
'''
Regularization in XGBoost helps in controlling overfitting by penalizing complex mo
regularization to shrink feature weights and prevent overly complex trees. This ens
Additionally, XGBoost includes parameters like `max_depth` and `min_child_weight` t
overfitting. These regularization techniques make XGBoost more stable and suitable
datasets.
'''
#11. What are some hyperparameters to tune in Gradient Boosting models?
'''
Tuning hyperparameters in Gradient Boosting is essential for optimal performance. K
which controls the contribution of each weak learner, and **number of estimators**,
The **maximum depth** of trees prevents overfitting, while **subsample** controls t
model. Other parameters like **min_child_weight** and **colsample_bytree** help bal
Proper tuning ensures the best trade-off between bias and variance.
'''

#12. What is the concept of Feature Importance in Boosting?
'''
Feature Importance in Boosting helps identify the most influential features in a da
scores to features based on their contribution to improving predictions. These scor
variables have the most impact. Feature Importance is useful for feature selection,
to be removed for better model efficiency. It is commonly used in domains like heal
interpret model decisions.
'''

#13. Why is CatBoost efficient for categorical data?
'''
```

```
CatBoost is designed to handle categorical variables efficiently without requiring
Target Encoding, which prevents data leakage by encoding categories based on past d
technique reduces overfitting by creating more generalized models. Unlike XGBoost a
CatBoost automatically processes categorical features, making it ideal for datasets
customer segmentation and sentiment analysis.
```

Practical

In [1]:
```
#Train an AdaBoost Classifier on a sample dataset and print model accuracy
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier


X, y = make_classification(n_samples=1000, n_features=20, random_state=42)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


base_estimator = DecisionTreeClassifier(max_depth=1)
adaboost_clf = AdaBoostClassifier(base_estimator, n_estimators=50, random_state=42)


adaboost_clf.fit(X_train, y_train)


y_pred = adaboost_clf.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 0.87
```

In [3]:
```
#Train an AdaBoost Regressor and evaluate performance using Mean Absolute Error (MA
from sklearn.datasets import make_regression
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor


X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=42)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


base_estimator = DecisionTreeRegressor(max_depth=4)
adaboost_reg = AdaBoostRegressor(base_estimator, n_estimators=50, random_state=42)


adaboost_reg.fit(X_train, y_train)
```

```
y_pred = adaboost_reg.predict(X_test)


mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Absolute Error (MAE): {mae:.2f}")
```

```
Mean Absolute Error (MAE): 73.09
```

In [5]:
```
#Train a Gradient Boosting Classifier on the Breast Cancer dataset and print featur
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import load_breast_cancer
import pandas as pd


data = load_breast_cancer()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


gb_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_sta
gb_clf.fit(X_train, y_train)


feature_importance = gb_clf.feature_importances_

feature_importance_df = pd.DataFrame(
{"Feature": data.feature_names, "Importance": feature_importance}
).sort_values(by="Importance", ascending=False)


print(feature_importance_df)
```

```
                    Feature  Importance
7      mean concave points    0.450418
27    worst concave points    0.240209
20            worst radius    0.075424
22         worst perimeter    0.051441
21           worst texture    0.039881
23              worst area    0.038200
1             mean texture    0.027821
26         worst concavity    0.017576
16         concavity error    0.012933
13              area error    0.010848
10            radius error    0.005238
24        worst smoothness    0.004811
19 fractal dimension error    0.004313
5         mean compactness    0.003838
11           texture error    0.003488
15       compactness error    0.002644
17    concave points error    0.002072
4          mean smoothness    0.002039
28          worst symmetry    0.001500
6           mean concavity    0.001115
25       worst compactness    0.000891
18          symmetry error    0.000709
14        smoothness error    0.000669
3                mean area    0.000555
8            mean symmetry    0.000466
12         perimeter error    0.000356
9   mean fractal dimension    0.000317
2           mean perimeter    0.000201
0              mean radius    0.000013
29 worst fractal dimension    0.000013
```

In [6]:

```
#4 Train a Gradient Boosting Regressor and evaluate using R-Squared Score
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score


X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=42)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


gb_reg = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_stat
gb_reg.fit(X_train, y_train)


y_pred = gb_reg.predict(X_test)


r2 = r2_score(y_test, y_pred)
print(f"R-Squared Score: {r2:.2f}")
```

```
R-Squared Score: 0.92
```

In [ ]:

```
!pip install xgboost
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Collecting xgboost
  Downloading xgboost-3.0.0-py3-none-manylinux_2_28_x86_64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in /opt/conda/envs/anaconda-2024.02-py310/lib/p
ython3.10/site-packages (from xgboost) (1.26.4)
Collecting nvidia-nccl-cu12 (from xgboost)
  Downloading nvidia_nccl_cu12-2.26.2-py3-none-manylinux2014_x86_64.manylinux_2_17_x
86_64.whl.metadata (2.0 kB)
Requirement already satisfied: scipy in /opt/conda/envs/anaconda-2024.02-py310/lib/p
ython3.10/site-packages (from xgboost) (1.12.0)
Downloading xgboost-3.0.0-py3-none-manylinux_2_28_x86_64.whl (253.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 253.9/253.9 MB 28.5 MB/s eta 0:00:000:0
0:01[36m0:00:01
Downloading nvidia_nccl_cu12-2.26.2-py3-none-manylinux2014_x86_64.manylinux_2_17_x86
_64.whl (201.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 201.3/201.3 MB 80.5 MB/s eta 0:00:000:0
0:01[36m0:00:01
Installing collected packages: nvidia-nccl-cu12, xgboost
```

In [10]:

```python
#Train an XGBoost Classifier on a dataset and compare accuracy with Gradient Boosti
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


data = load_breast_cancer()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


gb_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_sta
gb_clf.fit(X_train, y_train)


xgb_clf = XGBClassifier(n_estimators=100, learning_rate=0.1, eval_metric="logloss",
xgb_clf.fit(X_train, y_train)


y_pred_gb = gb_clf.predict(X_test)
y_pred_xgb = xgb_clf.predict(X_test)


accuracy_gb = accuracy_score(y_test, y_pred_gb)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)


print(f"Gradient Boosting Accuracy: {accuracy_gb:.4f}")
print(f"XGBoost Accuracy: {accuracy_xgb:.4f}")
```

```
Gradient Boosting Accuracy: 0.9561
XGBoost Accuracy: 0.9561
```

In [11]:
```
!pip install catboost
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Collecting catboost
  Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /home/104c0660-1be7-404b-8861-82ee06d5989
e/.local/lib/python3.10/site-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /opt/conda/envs/anaconda-2024.02-py310/
lib/python3.10/site-packages (from catboost) (3.8.0)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /opt/conda/envs/anaconda-2024.0
2-py310/lib/python3.10/site-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in /opt/conda/envs/anaconda-2024.02-py31
0/lib/python3.10/site-packages (from catboost) (2.1.4)
Requirement already satisfied: scipy in /opt/conda/envs/anaconda-2024.02-py310/lib/p
ython3.10/site-packages (from catboost) (1.12.0)
Requirement already satisfied: plotly in /opt/conda/envs/anaconda-2024.02-py310/lib/
python3.10/site-packages (from catboost) (5.19.0)
Requirement already satisfied: six in /opt/conda/envs/anaconda-2024.02-py310/lib/pyt
hon3.10/site-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/envs/anaconda-20
24.02-py310/lib/python3.10/site-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/envs/anaconda-2024.02-py31
0/lib/python3.10/site-packages (from pandas>=0.24->catboost) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/envs/anaconda-2024.02-py
310/lib/python3.10/site-packages (from pandas>=0.24->catboost) (2023.3)
Requirement already satisfied: contourpy>=1.0.1 in /opt/conda/envs/anaconda-2024.02-
py310/lib/python3.10/site-packages (from matplotlib->catboost) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/envs/anaconda-2024.02-py31
0/lib/python3.10/site-packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in /opt/conda/envs/anaconda-2024.02
-py310/lib/python3.10/site-packages (from matplotlib->catboost) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /opt/conda/envs/anaconda-2024.02
-py310/lib/python3.10/site-packages (from matplotlib->catboost) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /opt/conda/envs/anaconda-2024.02-p
y310/lib/python3.10/site-packages (from matplotlib->catboost) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/envs/anaconda-2024.02-py3
10/lib/python3.10/site-packages (from matplotlib->catboost) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in /opt/conda/envs/anaconda-2024.02-
py310/lib/python3.10/site-packages (from matplotlib->catboost) (3.0.9)
Requirement already satisfied: tenacity>=6.2.0 in /opt/conda/envs/anaconda-2024.02-p
y310/lib/python3.10/site-packages (from plotly->catboost) (8.2.2)
Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 58.9 MB/s eta 0:00:00 0:00:
01[36m0:00:01
Installing collected packages: catboost
Successfully installed catboost-1.2.7
```

In [12]:
```
#Train a CatBoost Classifier and evaluate using F1-Score
from catboost import CatBoostClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.metrics import f1_score


data = load_breast_cancer()
X, y = data.data, data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


cat_clf = CatBoostClassifier(n_estimators=100, learning_rate=0.1, verbose=0, random
cat_clf.fit(X_train, y_train)


y_pred_cat = cat_clf.predict(X_test)


f1 = f1_score(y_test, y_pred_cat)


print(f"F1-Score: {f1:.4f}")
```

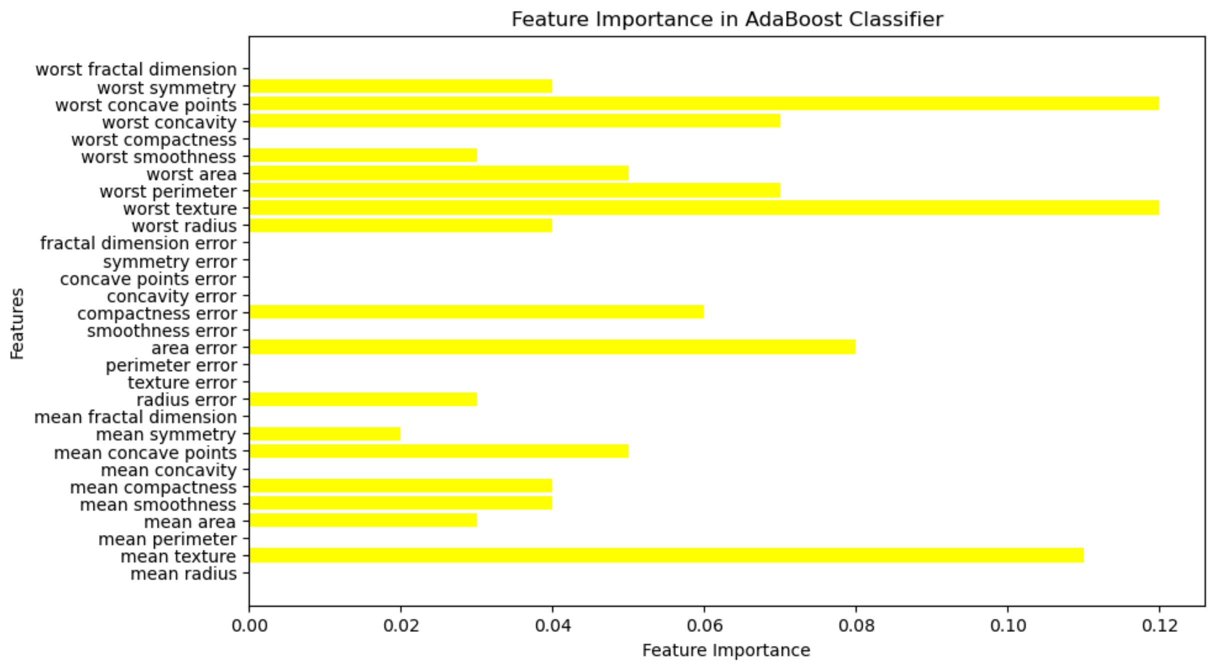
```
F1-Score: 0.9722
```

In [13]:
```
#Train an XGBoost Regressor and evaluate using Mean Squared Error (MSE)
from xgboost import XGBRegressor
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error


X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=42)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


xgb_reg = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_reg.fit(X_train, y_train)


y_pred_xgb = xgb_reg.predict(X_test)


mse = mean_squared_error(y_test, y_pred_xgb)


print(f"Mean Squared Error: {mse:.4f}")
```

```
Mean Squared Error: 4845.6934
```

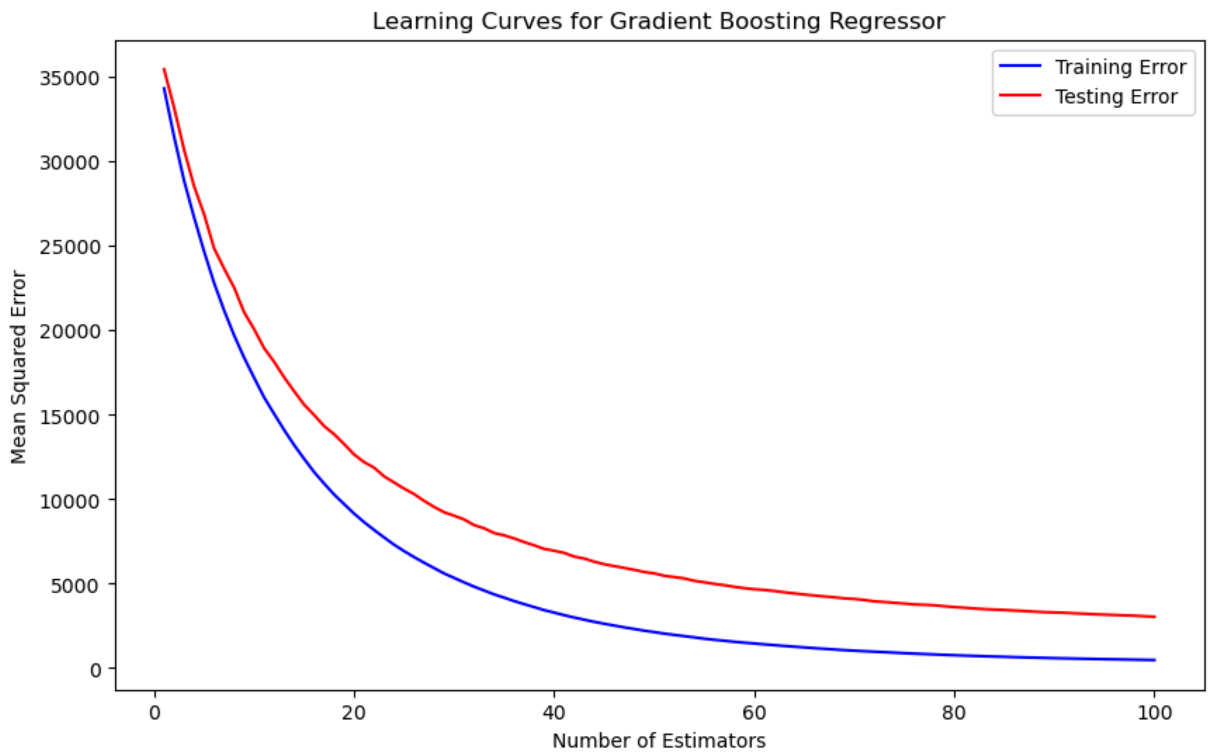
In [15]:
```
#Train an AdaBoost Classifier and visualize feature importance
import matplotlib.pyplot as plt

data = load_breast_cancer()
X, y = data.data, data.target
feature_names = data.feature_names


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


ada_clf = AdaBoostClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
```

```
ada_clf.fit(X_train, y_train)


feature_importance = ada_clf.feature_importances_


plt.figure(figsize=(10, 6))
plt.barh(feature_names, feature_importance, color='yellow')
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Feature Importance in AdaBoost Classifier")
plt.show()
```



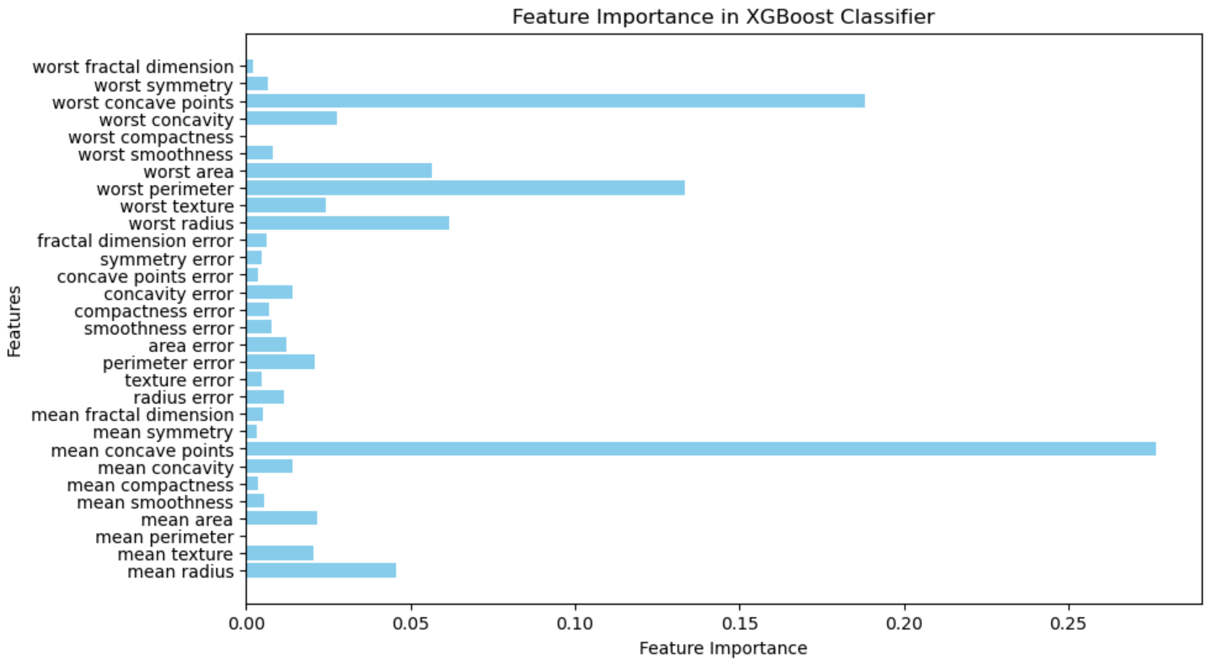
In [17]:
```
#Train a Gradient Boosting Regressor and plot learning curves

import numpy as np

X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=42)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize the Gradient Boosting Regressor
gb_reg = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_stat


train_errors = []
test_errors = []

for n_estimators in range(1, 101):
    gb_reg.set_params(n_estimators=n_estimators)
    gb_reg.fit(X_train, y_train)


    y_train_pred = gb_reg.predict(X_train)
```

```
    y_test_pred = gb_reg.predict(X_test)


    train_errors.append(mean_squared_error(y_train, y_train_pred))
    test_errors.append(mean_squared_error(y_test, y_test_pred))

plt.figure(figsize=(10, 6))
plt.plot(range(1, 101), train_errors, label="Training Error", color="blue")
plt.plot(range(1, 101), test_errors, label="Testing Error", color="red")
plt.xlabel("Number of Estimators")
plt.ylabel("Mean Squared Error")
plt.title("Learning Curves for Gradient Boosting Regressor")
plt.legend()
plt.show()
```



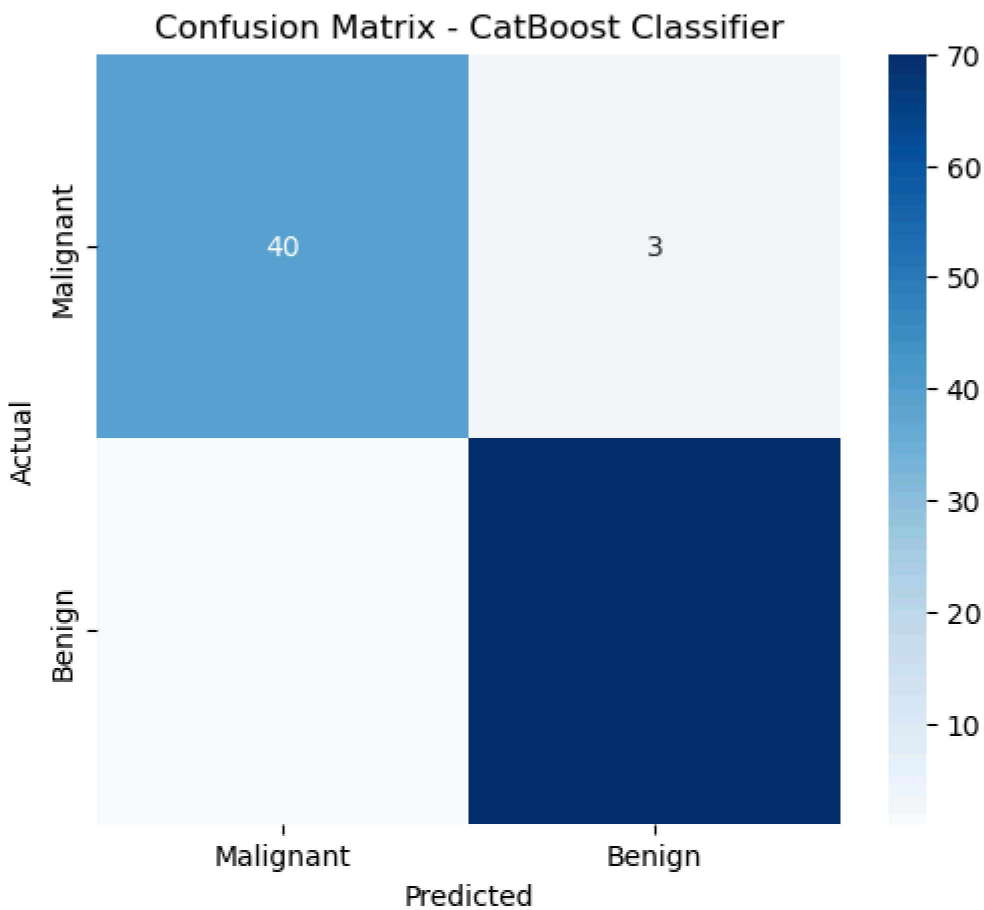
In [18]:
```
#Train an XGBoost Classifier and visualize feature importances



data = load_breast_cancer()
X, y = data.data, data.target
feature_names = data.feature_names


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


xgb_clf = XGBClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_clf.fit(X_train, y_train)


feature_importance = xgb_clf.feature_importances_
```

```
plt.figure(figsize=(10, 6))
plt.barh(feature_names, feature_importance, color='skyblue')
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Feature Importance in XGBoost Classifier")
plt.show()
```



In [19]:
```
#Train a CatBoost Classifier and plot the confusion matrix
import seaborn as sns
from sklearn.metrics import confusion_matrix


data = load_breast_cancer()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


cat_clf = CatBoostClassifier(n_estimators=100, learning_rate=0.1, verbose=0, random
cat_clf.fit(X_train, y_train)


y_pred_cat = cat_clf.predict(X_test)


cm = confusion_matrix(y_test, y_pred_cat)


plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Malignant", "Benig
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

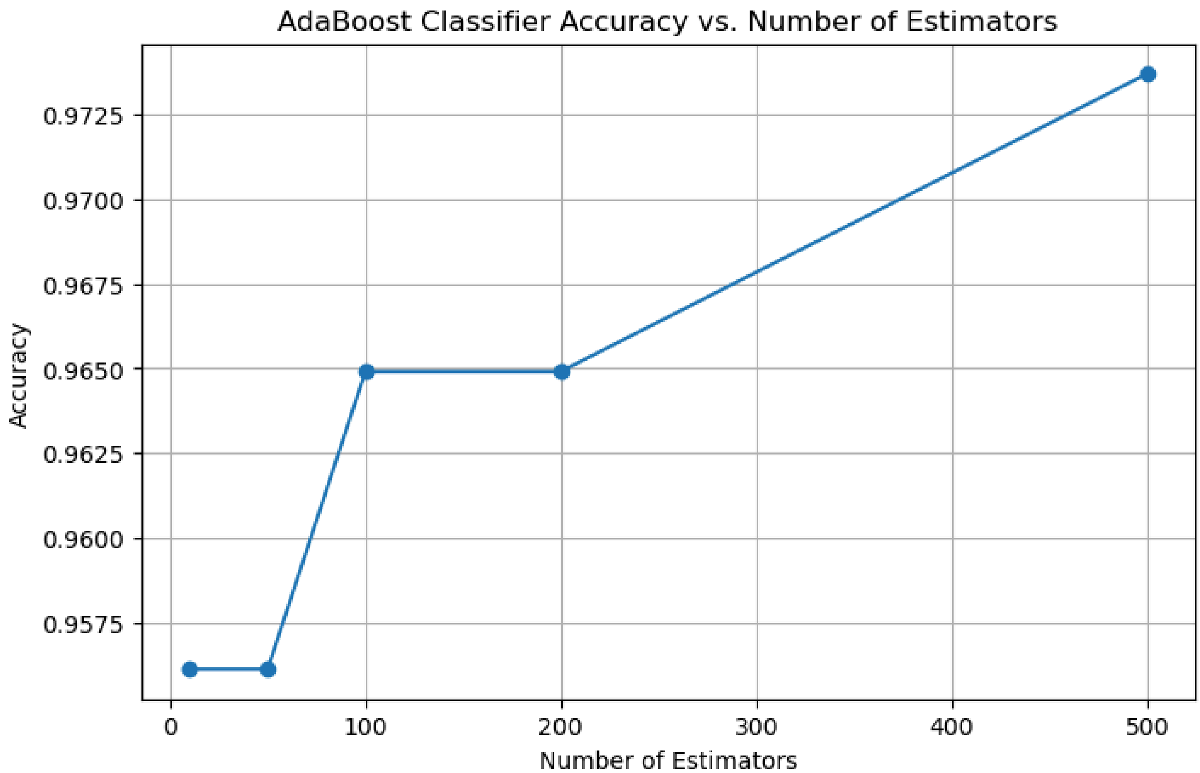
```
plt.title("Confusion Matrix - CatBoost Classifier")
plt.show()
```



In [20]:
```
#Train an AdaBoost Classifier with different numbers of estimators and compare accu

data = load_breast_cancer()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


n_estimators_list = [10, 50, 100, 200, 500]


accuracy_scores = []


for n in n_estimators_list:
    ada_clf = AdaBoostClassifier(n_estimators=n, learning_rate=0.1, random_state=42
    ada_clf.fit(X_train, y_train)
    y_pred = ada_clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracy_scores.append(acc)
    print(f"n_estimators={n}, Accuracy: {acc:.4f}")


plt.figure(figsize=(8, 5))
```

```
plt.plot(n_estimators_list, accuracy_scores, marker='o', linestyle='-')
plt.xlabel("Number of Estimators")
plt.ylabel("Accuracy")
plt.title("AdaBoost Classifier Accuracy vs. Number of Estimators")
plt.grid(True)
plt.show()
```

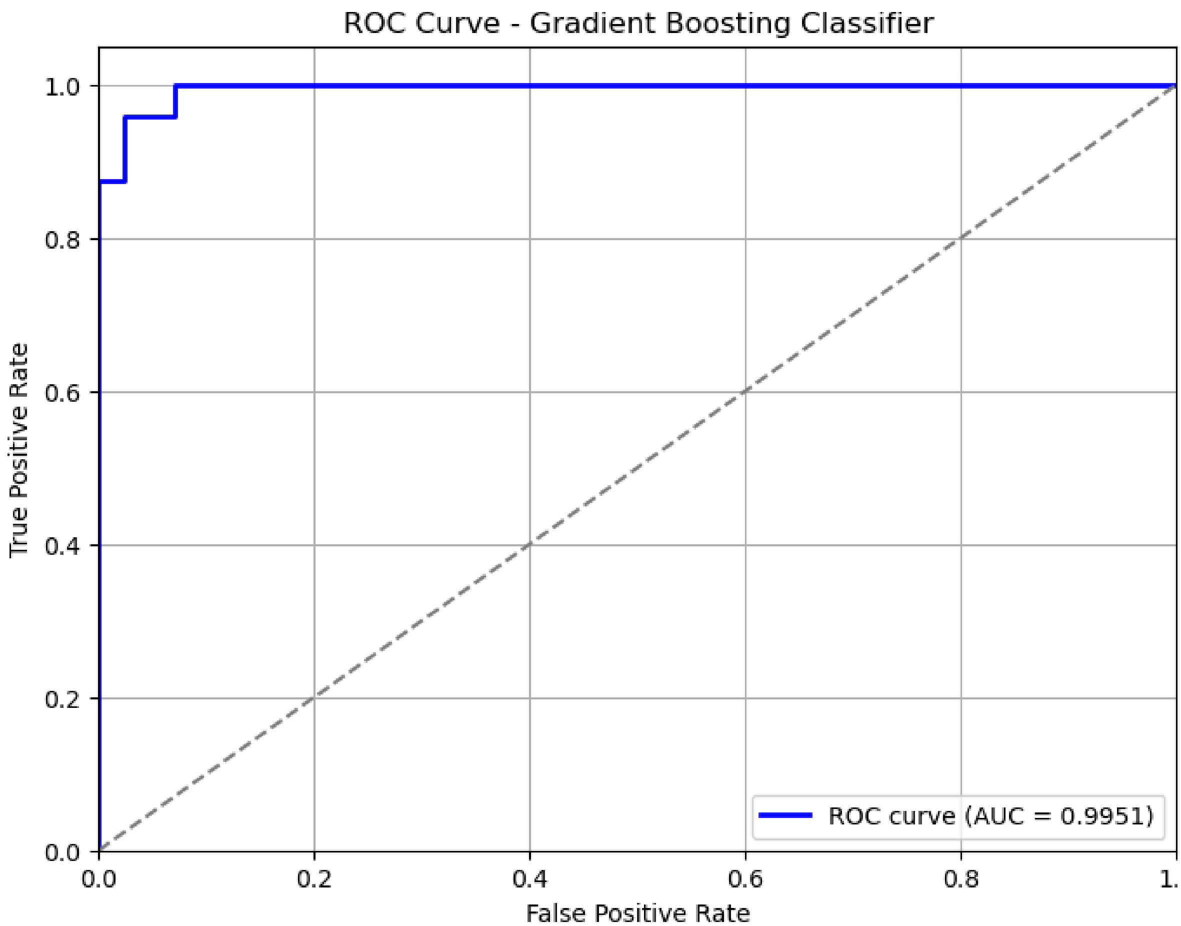
```
n_estimators=10, Accuracy: 0.9561
n_estimators=50, Accuracy: 0.9561
n_estimators=100, Accuracy: 0.9649
n_estimators=200, Accuracy: 0.9649
n_estimators=500, Accuracy: 0.9737
```



In [21]:

```
# Train a Gradient Boosting Classifier and visualize the ROC curve
from sklearn.metrics import roc_curve, auc


data = load_breast_cancer()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


gb_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_sta
gb_clf.fit(X_train, y_train)


y_scores = gb_clf.predict_proba(X_test)[:, 1]


fpr, tpr, _ = roc_curve(y_test, y_scores)
roc_auc = auc(fpr, tpr)
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f"ROC curve (AUC = {roc_auc:.4f})")
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Gradient Boosting Classifier")
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



In [22]:
```
#Train an XGBoost Regressor and tune the learning rate using GridSearchCV
from xgboost import XGBRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import  GridSearchCV



X, y = make_regression(n_samples=1000, n_features=20, noise=0.1, random_state=42)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

```
xgb_reg = XGBRegressor(n_estimators=100, random_state=42)


param_grid = {
    'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3]
}


grid_search = GridSearchCV(xgb_reg, param_grid, cv=5, scoring='neg_mean_squared_err
grid_search.fit(X_train, y_train)


best_learning_rate = grid_search.best_params_['learning_rate']
print(f"Best Learning Rate: {best_learning_rate}")


best_xgb_reg = XGBRegressor(n_estimators=100, learning_rate=best_learning_rate, ran
best_xgb_reg.fit(X_train, y_train)


y_pred = best_xgb_reg.predict(X_test)


mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
```

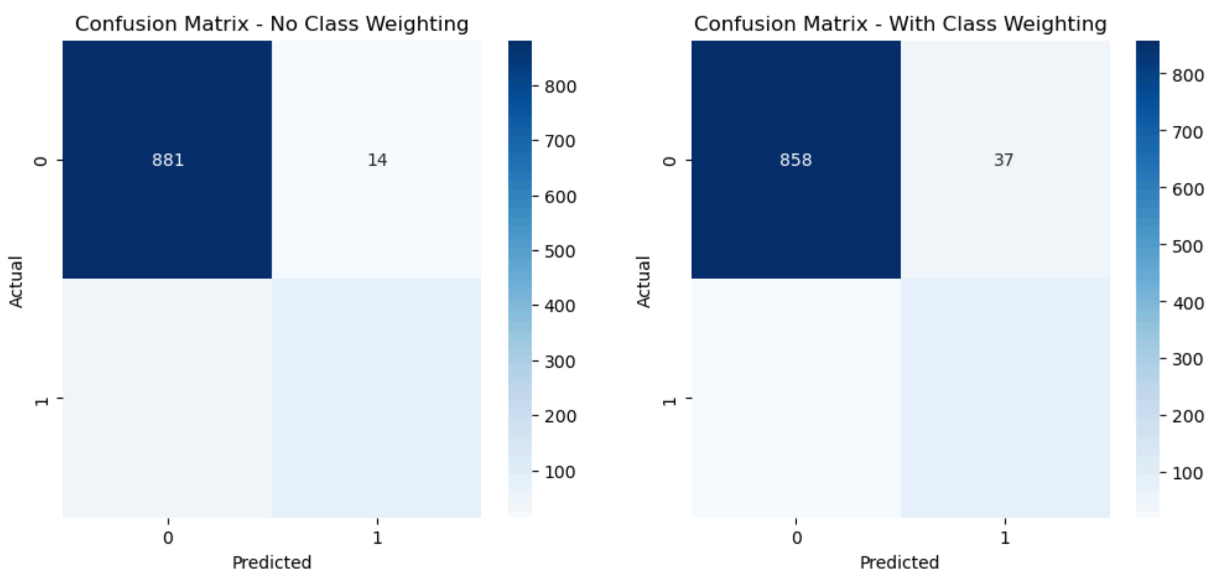
```
Best Learning Rate: 0.1
Mean Squared Error: 4845.6934
```

In [24]:

```
#Train a CatBoost Classifier on an imbalanced dataset and compare performance with
from sklearn.metrics import classification_report

X, y = make_classification(n_samples=5000, n_features=20, weights=[0.9, 0.1], rando


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


cat_clf_no_weight = CatBoostClassifier(iterations=100, learning_rate=0.1, verbose=0
cat_clf_no_weight.fit(X_train, y_train)
y_pred_no_weight = cat_clf_no_weight.predict(X_test)


class_weights = {0: 1, 1: 10}
cat_clf_weighted = CatBoostClassifier(iterations=100, learning_rate=0.1, class_weig
cat_clf_weighted.fit(X_train, y_train)
y_pred_weighted = cat_clf_weighted.predict(X_test)


print("Without Class Weighting:\n", classification_report(y_test, y_pred_no_weight)
print("With Class Weighting:\n", classification_report(y_test, y_pred_weighted))


fig, axes = plt.subplots(1, 2, figsize=(12, 5))

cm_no_weight = confusion_matrix(y_test, y_pred_no_weight)
```

```
cm_weighted = confusion_matrix(y_test, y_pred_weighted)

sns.heatmap(cm_no_weight, annot=True, fmt="d", cmap="Blues", ax=axes[0])
axes[0].set_title("Confusion Matrix - No Class Weighting")
axes[0].set_xlabel("Predicted")
axes[0].set_ylabel("Actual")

sns.heatmap(cm_weighted, annot=True, fmt="d", cmap="Blues", ax=axes[1])
axes[1].set_title("Confusion Matrix - With Class Weighting")
axes[1].set_xlabel("Predicted")
axes[1].set_ylabel("Actual")

plt.show()
```

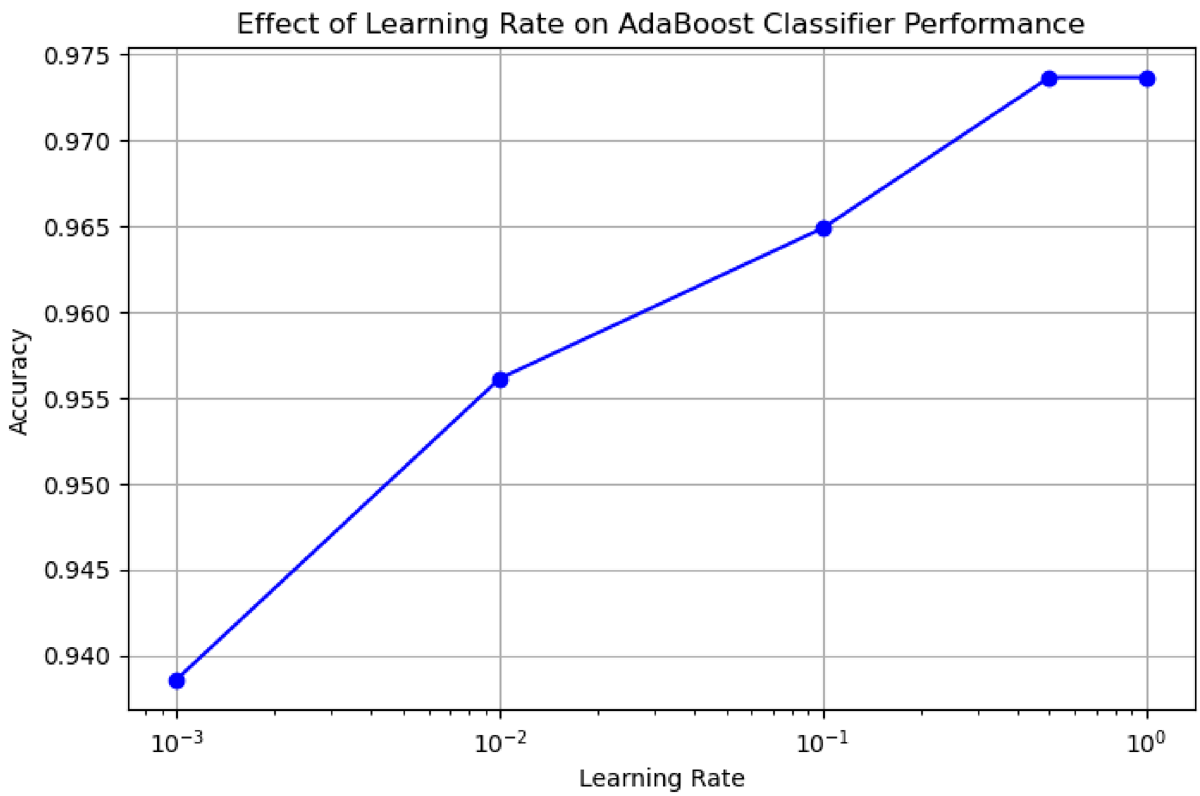
```
Without Class Weighting:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       895
           1       0.85      0.73      0.79       105

    accuracy                           0.96      1000
   macro avg       0.91      0.86      0.88      1000
weighted avg       0.96      0.96      0.96      1000

With Class Weighting:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       895
           1       0.69      0.80      0.74       105

    accuracy                           0.94      1000
   macro avg       0.84      0.88      0.86      1000
weighted avg       0.95      0.94      0.94      1000
```



In [25]:
```
#Train an AdaBoost Classifier and analyze the effect of different learning rates

data = load_breast_cancer()
X, y = data.data, data.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


learning_rates = [0.001, 0.01, 0.1, 0.5, 1.0]


accuracy_scores = []

for lr in learning_rates:
    ada_clf = AdaBoostClassifier(n_estimators=100, learning_rate=lr, random_state=4
    ada_clf.fit(X_train, y_train)
    y_pred = ada_clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    accuracy_scores.append(acc)
    print(f"Learning Rate={lr}, Accuracy: {acc:.4f}")


plt.figure(figsize=(8, 5))
plt.plot(learning_rates, accuracy_scores, marker='o', linestyle='-', color='b')
plt.xlabel("Learning Rate")
plt.ylabel("Accuracy")
plt.title("Effect of Learning Rate on AdaBoost Classifier Performance")
plt.xscale("log")  # Log scale for better visualization
plt.grid(True)
plt.show()
```

```
Learning Rate=0.001, Accuracy: 0.9386
Learning Rate=0.01, Accuracy: 0.9561
Learning Rate=0.1, Accuracy: 0.9649
Learning Rate=0.5, Accuracy: 0.9737
Learning Rate=1.0, Accuracy: 0.9737
```



In [26]:
```
# Train an XGBoost Classifier for multi-class classification and evaluate using log
import xgboost as xgb
from sklearn.datasets import load_digits
from sklearn.metrics import log_loss


data = load_digits()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


xgb_clf = xgb.XGBClassifier(objective="multi:softprob", num_class=10, n_estimators=
xgb_clf.fit(X_train, y_train)


y_pred_proba = xgb_clf.predict_proba(X_test)


logloss = log_loss(y_test, y_pred_proba)
print(f"Log-Loss: {logloss:.4f}")
```

```
Log-Loss: 0.1732
```

In [ ]: