A boosting algorithm that builds models sequentially by minimizing a loss function using gradient descent.Iteratively adds weak learners to improve overall model perfomance.

Key Parameters

Learning Rate

Determines the contribution of each weak learner.Smaller values reduce overfitting but require more iterations.Typical range: 0.01 to 0.3

Number of estimators

The number of weak learners added sequentially.Larger values improves learning but increases computation time.

Regularization

i.e limiting tree depth or adding penalities to prevent overfitting.Shallower trees generalize better but might underfit.

In [1]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [2]:
```
# Load Dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

In [3]:
```
# display dataset information
print(f"Features: {data.feature_names}")
print(f"Classes: {data.target_names}")
```

```
Features: ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Classes: ['malignant' 'benign']
```

In [4]:
```
# Train Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)
```

Out[4]:
```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

In [5]:
```
# Predict
y_pred_gb = gb_model.predict(X_test)
```

In [6]:
```
# Evaluate performance
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print(f"Gradient Boosting Accuracy: {accuracy_gb}")
print("\n Classification Report: \n", classification_report(y_test, y_pred_gb))
```

```
Gradient Boosting Accuracy: 0.956140350877193

 Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.93      0.94        43
           1       0.96      0.97      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

In [8]:
```
#Define hyperparameter grid
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7]
}

# Perform Grid Search
grid_search = GridSearchCV(
    estimator=GradientBoostingClassifier(random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)
```

In [9]:
```
grid_search.fit(X_train, y_train)
```

Out[9]:
```
GridSearchCV
estimator: GradientBoostingClassifier
GradientBoostingClassifier
```

In [10]:
```
# Display best parameteres and score
print(f"Best Parameters: {grid_search.best_params_}")
print(f"Best Cross-Validation Accuracy: {grid_search.best_score_}")
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
Best Cross-Validation Accuracy: 0.9648351648351647
```

In [11]:
```
# Train Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

## Predict
y_pred_rf = rf_model.predict(X_test)

# Evaluate Performance
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Accuracy: {accuracy_rf}")
```

```
Random Forest Accuracy: 0.9649122807017544
```