Part I: Overview of Boosting (AdaBoost)

Boosting is a machine learning technique that combines the predictions of multiple weak learners to create a strong learner. AdaBoost (Adaptive Boosting) is one of the most popular algorithms in this family. The main idea is to iteratively train weak classifiers on different distributions of the training data, focusing more on the data points that were previously misclassified.

Advantages

- Can significantly improve the performance of weak learners.
- Reduces bias and variance, making it flexible and adaptive.
- Effective for both classification and regression tasks.

Disadvantages

- Sensitive to noisy data and outliers.
- Computationally expensive due to the iterative nature of the training process.

Representation

In AdaBoost, the final hypothesis $H(x)$ is represented as a weighted sum of the weak learners $h_t(x)$:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Where:

- $h_t(x)$ is the hypothesis from the weak learner at iteration t.
- α_t is the weight assigned to hypothesis h_t, which is determined based on the classification accuracy.

Loss Function

AdaBoost minimizes the exponential loss, which is given by:

$$L(H) = \sum_{i=1}^{m} \exp(-y_i H(x_i))$$

Where:

- $y_i \in \{-1, +1\}$ is the true label for data point x_i.
- $H(x_i)$ is the combined prediction from all weak learners at point x_i.

The exponential loss heavily penalizes misclassified examples, encouraging the model to focus on difficult cases.

Optimizer

The optimizer in AdaBoost is not gradient-based but rather works by adjusting the distribution D of the training data based on the performance of the weak learner at each step. The update for the distribution after round t is:

$$D_{t+1}(i) = \frac{D_t(i) \cdot \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Where:

- $D_t(i)$ is the weight of sample i at round t.
- α_t is the weight for weak learner h_t.
- Z_t is a normalization factor to ensure that D_{t+1} is a valid probability distribution.

Pseudocode

```
# Pseudocode for AdaBoost

# Input: Training set S = {(x1, y1), (x2, y2), ..., (xm, ym)}, weak learner WL, number of rounds T
# Output: Final hypothesis H(x)

Initialize distribution D1(i) = 1/m for all i = 1 to m
```

```
for t = 1 to T:
    1. Train weak learner h_t using distribution Dt
    2. Compute error ε_t = sum(Dt(i) * [h_t(x_i) != y_i]) for all i
    3. Compute weight α_t = 0.5 * log((1 - ε_t) / ε_t)
    4. Update distribution:
        Dt+1(i) = Dt(i) * exp(-α_t * y_i * h_t(x_i))
        Normalize Dt+1 to make it a probability distribution

Output final hypothesis:
    H(x) = sign(sum(α_t * h_t(x) for t = 1 to T))
```

Part II: Adaboost Stencil Code

Part III: Unit Tests

Part IV: Check Model

Breast Cancer Wisconsin Dataset

The **Breast Cancer Wisconsin (Diagnostic) dataset** is commonly used to evaluate binary classification models due to its clean structure and medically relevant features. This dataset, obtained from the UCI Machine Learning Repository, contains 569 samples, each labeled as either **malignant** or **benign** based on various features computed from breast mass images.

Objective

Our goal is to apply the **Adaboost** algorithm to this dataset, leveraging decision stumps (i.e., single-depth decision trees) as weak learners. Adaboost iteratively adjusts the importance of misclassified samples, making it especially powerful for binary classification tasks with a strong focus on hard-to-classify instances.

Methodology

1. **Data Loading**: We load the data directly from a local file to avoid using pre-packaged datasets. The dataset includes 30 features that describe characteristics of the cell nuclei present in the image.
2. **Preprocessing**: We convert the `Diagnosis` column to a binary format, where **1** represents malignant and **0** represents benign.
3. **Training**: We apply Adaboost with 50 estimators (weak learners), where each estimator focuses on correcting the errors of the previous one.
4. **Evaluation**: We evaluate the model on the test set using **Accuracy** and **F1 Score**, the latter of which provides a balanced measure between precision and recall, useful in medical diagnoses.

Results

The Adaboost model achieved:

- **Accuracy**: 98.25%
- **F1 Score**: 97.6%

These results highlight Adaboost's robustness, achieving high performance on this diagnostic dataset, indicating its potential utility in medical classification tasks.

Reference

1. W.N. Street, W.H. Wolberg, and O.L. Mangasarian. "Nuclear feature extraction for breast tumor diagnosis." *IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology*. International Society for Optics and Photonics, 1993.
2. Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)].

Code

The implementation can be found in the associated code cell, which trains the Adaboost model on the Breast Cancer Wisconsin dataset and evaluates its performance.

In [ ]:
```python
# Import necessary libraries
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score

# Define file paths (update these paths if needed)
data_path = '/Users/liuzhenke/Desktop/Brown CSCI/DATA2060 ML/Final Project/wdbc.data'

# Define column names based on the dataset description
column_names = ['ID', 'Diagnosis'] + [f'Feature_{i}' for i in range(1, 31)]

# Load the data into a DataFrame
```

```
data = pd.read_csv(data_path, header=None, names=column_names)

# Drop ID column as it's not needed
data = data.drop(columns=['ID'])

# Convert Diagnosis column to binary values: M -> 1 (malignant), B -> 0 (benign)
data['Diagnosis'] = data['Diagnosis'].map({'M': 1, 'B': 0})

# Split features and target
X = data.drop(columns=['Diagnosis'])
y = data['Diagnosis']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Initialize Adaboost with Decision Trees as weak learners
adaboost_model = AdaBoostClassifier(n_estimators=50, random_state=42, algorithm='SAMME')

# Train the model
adaboost_model.fit(X_train, y_train)

# Predict on test data
y_pred = adaboost_model.predict(X_test)

# Evaluate accuracy and F1 score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Output results
print(f"Accuracy on test set: {accuracy:.2%}")
print(f"F1 Score on test set: {f1:.2%}")
```

```
Accuracy on test set: 98.25%
F1 Score on test set: 97.60%
```

Part V: Contributions to the Project