

Name: Girish Nandanwar	Course: 22CT744 – Lab. Machine Learning
Roll No: A-56	Department: Computer Technology

## Practical No.8

**Aim:** To implement Ensemble methods (bagging and boosting)

### Theory:

#### 1. Introduction to Ensemble Learning

Ensemble learning is a technique that combines **multiple machine learning models** (called *weak learners*) to produce a more powerful, accurate, and generalized model (called a *strong learner*).

The idea is simple:

“Instead of relying on one model, we use many models and combine their outputs to reduce bias, variance, or both.”

There are two primary types of ensemble methods:

- **Bagging (Bootstrap Aggregation)** – reduces variance
  - **Boosting** – reduces bias
- 

#### 2. Bagging (Bootstrap Aggregating)

##### ◊ Concept

Bagging builds **multiple independent models** (usually of the same type, like Decision Trees) on **different random subsets** of the dataset.

The subsets are generated by *bootstrapping* — i.e., sampling with replacement.

Each model gives a prediction, and the final output is obtained by:

- **Voting** (for classification)

- **Averaging** (for regression)

- ◊ **Goal**

To reduce **variance** and avoid overfitting by averaging out the predictions of multiple high-variance models.

- ◊ **Steps:**

1. Create multiple bootstrapped samples from the training data.
2. Train a base learner (e.g., Decision Tree) on each sample.
3. Aggregate all predictions (majority vote or mean).

- ◊ **Example Algorithms:**

- **Random Forest** → A type of bagging ensemble with decision trees + random feature selection.
- **BaggingClassifier** → Generic bagging model in scikit-learn.

- ◊ **Advantages:**

- Reduces overfitting.
- Improves stability and accuracy.
- Works well for unstable models (like Decision Trees).

- ◊ **Disadvantages:**

- Computationally expensive.
- Does not significantly reduce bias.

---

### 3. Boosting

- ◊ **Concept**

Boosting builds models **sequentially**, where each new model focuses on **correcting the errors** of the previous ones.

It assigns **higher weights** to misclassified samples so that the next model pays more attention to difficult cases.

The final prediction is a **weighted combination** of all weak models.

◊ **Goal**

To reduce **bias** and improve model accuracy by focusing on hard-to-predict instances.

◊ **Steps:**

1. Initialize all sample weights equally.
2. Train a weak learner.
3. Increase the weights of misclassified samples.
4. Train the next learner on this weighted data.
5. Combine all models to form the final strong model.

◊ **Example Algorithms:**

- **AdaBoost (Adaptive Boosting)**
- **Gradient Boosting**
- **XGBoost / LightGBM / CatBoost** (advanced versions)

◊ **Advantages:**

- High accuracy.
- Works well with weak learners.
- Handles bias effectively.

◊ **Disadvantages:**

- Sensitive to noise and outliers.
- Can overfit if not tuned properly.
- Sequential nature makes it slower to train.

## Code:

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# =====
# Create the dataset
# =====
data = {
    "Country": ["France", "Spain", "Germany", "Spain", "Germany", "France", "Spain", "France",
    "Germany", "France"],
    "Age": [44, 27, 30, 38, 40, 35, None, 48, 50, 37],
    "Salary": [72000, 48000, 54000, 61000, None, 58000, 52000, 79000, 83000, 67000],
    "Purchased": ["No", "Yes", "No", "No", "Yes", "Yes", "No", "Yes", "No", "Yes"]
}

df = pd.DataFrame(data)

# =====
# Data Preprocessing
# =====
df["Age"].fillna(df["Age"].mean(), inplace=True)
df["Salary"].fillna(df["Salary"].mean(), inplace=True)

# Encode categorical variables
le_country = LabelEncoder()
df["Country"] = le_country.fit_transform(df["Country"])

le_purchase = LabelEncoder()
df["Purchased"] = le_purchase.fit_transform(df["Purchased"])

```

```

# Split features and target
X = df[["Country", "Age", "Salary"]]
y = df["Purchased"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# =====
# Bagging Classifier
# =====
base_model = DecisionTreeClassifier(random_state=42)
bagging = BaggingClassifier(base_estimator=base_model, n_estimators=10, random_state=42)
bagging.fit(X_train, y_train)
y_pred_bagging = bagging.predict(X_test)

print(" ◇ BAGGING CLASSIFIER RESULTS ◇ ")
print("Accuracy:", accuracy_score(y_test, y_pred_bagging))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_bagging))
print("\nClassification Report:\n", classification_report(y_test, y_pred_bagging))

# =====
# AdaBoost Classifier (Boosting)
# =====
boosting = AdaBoostClassifier(base_estimator=base_model, n_estimators=10, learning_rate=1.0,
random_state=42)
boosting.fit(X_train, y_train)
y_pred_boosting = boosting.predict(X_test)

print("\n\n ◇ ADABOOST CLASSIFIER RESULTS ◇ ")
print("Accuracy:", accuracy_score(y_test, y_pred_boosting))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_boosting))
print("\nClassification Report:\n", classification_report(y_test, y_pred_boosting))

```

## **Result:**

```
Confusion Matrix:
```

```
[[0 1]
 [2 0]]
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1.0
1	0.00	0.00	0.00	2.0
accuracy			0.00	3.0
macro avg	0.00	0.00	0.00	3.0
weighted avg	0.00	0.00	0.00	3.0

#### BAGGING CLASSIFIER RESULTS

```
Confusion Matrix:
```

```
[[0 1]
 [2 0]]
```

```
Classification Report:
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1.0
1	0.00	0.00	0.00	2.0
accuracy			0.00	3.0
macro avg	0.00	0.00	0.00	3.0
weighted avg	0.00	0.00	0.00	3.0

#### ADABOOST CLASSIFIER RESULTS

## Conclusion:

After running the code:

- The **Bagging Classifier** will give stable accuracy due to averaging multiple independent trees.
- The **Boosting Classifier** (AdaBoost) may give higher accuracy by focusing on difficult-to-predict samples.