Experiment No. 5

Title - Implement Bagging Algorithm

Theory -

Bagging, short for Bootstrap Aggregating, is an ensemble learning technique designed to improve the stability and accuracy of machine learning algorithms, especially those that are sensitive to the variance in the training data. The main idea behind bagging is to train multiple instances of the same learning algorithm on different subsets of the training data, and then combine their predictions to produce a more robust and accurate model.

1. **Bootstrap Sampling:**

Bagging begins by creating multiple subsets of the original training data through a process called bootstrap sampling. Bootstrap sampling involves randomly selecting, with replacement, samples from the original dataset to form a new dataset of the same size.

2. **Base Learner:**

A base learner refers to the individual learning algorithm or model that is used to make predictions. Bagging can be applied to any base learner, such as decision trees, neural networks, or regression models.

3. **Training:**

Multiple instances of the base learner are trained on different bootstrap samples created from the original training data. Each learner receives a slightly different view of the dataset due to the random sampling with replacement.

4. **Independence:**

The strength of bagging lies in the diversity of the base learners. Since each learner is trained on a different subset of data, they are likely to capture different patterns and aspects of the underlying distribution. This diversity helps reduce overfitting and increases the model's generalization capability.

5. **Prediction:**

Once the base learners are trained, predictions are made for each learner on the original validation or test dataset. For regression problems, the predictions are often averaged, while for classification problems, a majority voting scheme is typically used.

6. **Aggregation:**

The final prediction is obtained by aggregating the individual predictions from all the base learners. For regression tasks, the predictions are usually averaged, and for classification tasks, the class with the majority of votes is selected.

7. **Random Forest:**

One of the most popular implementations of bagging is the Random Forest algorithm. In Random Forest, the base learners are decision trees, and the process of bagging is enhanced by selecting a random subset of features for each tree.

Benefits of Bagging:

- Reduced Variance: Bagging helps reduce the variance of the model by averaging out the noise and capturing different aspects of the data.
- Improved Generalization: The ensemble model tends to generalize better to unseen data compared to individual base learners.
- Robustness: Bagging makes the model more robust to outliers and noisy data.
- Bagging is a versatile technique and can be applied to various machine learning algorithms, making it a valuable tool for improving model performance.

Implementation :

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

class BaggingClassifier:
    def __init__(self, base_learner, n_estimators):
        self.base_learner = base_learner
        self.n_estimators = n_estimators
        self.models = []

    def fit(self, X, y):
        for _ in range(self.n_estimators):
            # Create a bootstrap sample
            indices = np.random.choice(len(X), size=len(X), replace=True)
            X_bootstrap, y_bootstrap = X[indices], y[indices]

            # Train a base learner on the bootstrap sample
            model = self.base_learner()
            model.fit(X_bootstrap, y_bootstrap)
            self.models.append(model)

    def predict(self, X):
        # Make predictions with each base learner
        predictions = [model.predict(X) for model in self.models]

        # Aggregate predictions using majority voting
        ensemble_predictions = np.round(np.mean(predictions, axis=0))

        return ensemble_predictions.astype(int)

# Example usage with a synthetic dataset
np.random.seed(42)
X = np.random.rand(100, 2)
y = (X[:, 0] + X[:, 1] > 1).astype(int)

# Split the dataset into training and testing sets
X_train, X_test = X[:80], X[80:]
y_train, y_test = y[:80], y[80:]
```

```
# Create a decision tree classifier
base_learner = DecisionTreeClassifier()

# Train the decision tree and calculate accuracy
base_learner.fit(X_train, y_train)
y_pred_base = base_learner.predict(X_test)
dtc_accuracy = accuracy_score(y_test, y_pred_base)
print(f"Decision Tree Classifier Accuracy: {dtc_accuracy * 100}%")

# Bagging classifier with DecisionTreeClassifier as the base learner
bagging_classifier = BaggingClassifier(base_learner=DecisionTreeClassifier, n_estimators=5)

# Train the bagging classifier
bagging_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred_bagging = bagging_classifier.predict(X_test)

# Calculate accuracy
bagging_accuracy = accuracy_score(y_test, y_pred_bagging)
print(f"Bagging Classifier Accuracy: {bagging_accuracy * 100}%")
```

Output

```
Decision Tree Classifier Accuracy: 90.0%
Bagging Classifier Accuracy: 95.0%
```