PART A

(PART A: TO BE REFFERED BY STUDENTS)

Experiment No. 4

A.1 Aim:

To implement Ensemble algorithm.

A.2 Prerequisite:

Python Basic Concepts

A.3 Outcome:

Students will be able to implement Ensemble algorithm.

A.4 Theory:

Ensemble Learning Techniques in Machine Learning, Machine learning models suffer bias and/or variance. Bias is the difference between the predicted value and actual value by the model. Bias is introduced when the model doesn't consider the variation of data and creates a simple model. The simple model doesn't follow the patterns of data, and hence the model gives errors in predicting training as well as testing data i.e. the model with high bias and high variance

When the model follows even random quirks of data, as pattern of data, then the model might do very well on training dataset i.e. it gives low bias, but it fails on test data and gives high variance.

Therefore, to improve the accuracy (estimate) of the model, ensemble learning methods are developed. Ensemble is a machine learning concept, in which several models are trained using machine learning algorithms. It combines low performing classifiers (also called as weak learners or base learner) and combine individual model prediction for the final prediction.

On the basis of type of base learners, ensemble methods can be categorized as homogeneous and heterogeneous ensemble methods. If base learners are same, then it is a homogeneous ensemble method. If base learners are different then it is a heterogeneous ensemble method.

Ensemble Learning Methods

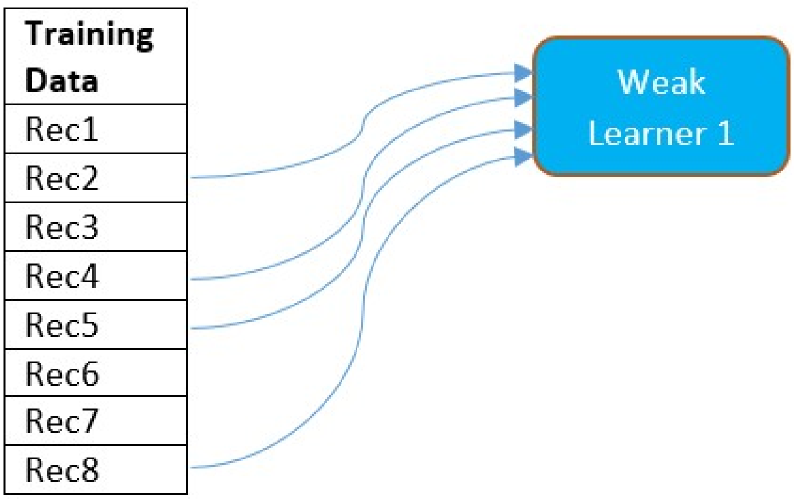
Ensemble techniques are classified into three types:

1. Bagging
2. Boosting
3. Stacking
4. **Bagging**

Consider a scenario where you are looking at the users' ratings for a product. Instead of approving one user's good/bad rating, we consider average rating given to the product. With average rating, we can be considerably sure of quality of the product. Bagging makes use of this principle. Instead of depending on one model, it runs the data through multiple models in parallel, and average them out as model's final output.

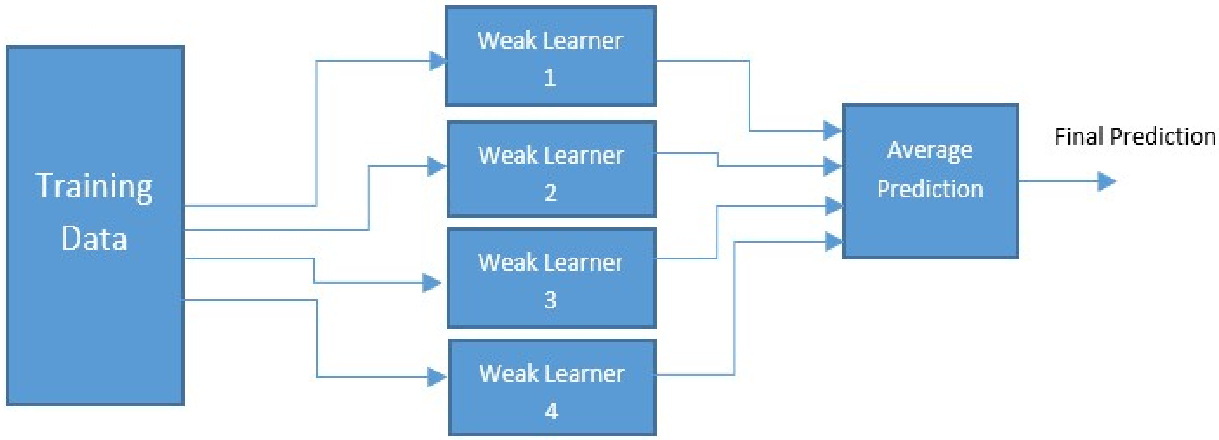
What is Bagging? How it works?

- Bagging is an acronym for **Bootstrapped Aggregation**. Bootstrapping means random selection of records with replacement from the training dataset. 'Random selection with replacement' can be explained as follows:



a. Consider that there are 8 samples in the training dataset. Out of these 8 samples, every weak learner gets 5 samples as training data for the model. These 5 samples need not be unique, or non-repetitive.
b. The model (weak learner) is allowed to get a sample multiple times. For example, as shown in the figure, Rec5 is selected 2 times by the model. Therefore, weak learner1 gets Rec2, Rec5, Rec8, Rec5, Rec4 as training data.
c. All the samples are available for selection to next weak learners. Thus all 8 samples will be available for next weak learner and any sample can be selected multiple times by next weak learners.

- Bagging is a **parallel method**, which means several weak learners learn the data pattern independently and simultaneously. This can be best shown in the below diagram:



1. The output of each weak learner is averaged to generate final output of the model.
2. Since the weak learner's outputs are averaged, this mechanism helps to reduce variance or variability in the predictions. However, it does not help to reduce bias of the model.
3. Since final prediction is an average of output of each weak learner, it means that each weak learner has equal say or weight in the final output.

To summarize:

1. Bagging is Bootstrapped Aggregation
2. It is Parallel method
3. Final output is calculated by averaging the outputs produced by individual weak learner
4. Each weak learner has equal say
5. Bagging reduces variance

Boosting

We saw that in bagging every model is given equal preference, but if one model predicts data more correctly than the other, then higher weightage should be given to this model over the other. Also, the model should attempt to reduce bias. These concepts are applied in the second ensemble method that we are going to learn, that is Boosting.

What is Boosting?

1. To start with, boosting assigns equal weights to all data points as all points are equally important in the beginning. For example, if a training dataset has N samples, it assigns weight = 1/N to each sample.
2. The weak learner classifies the data. The weak classifier classifies some samples correctly, while making mistake in classifying others.

3. After classification, sample weights are changed. Weight of correctly classified sample is reduced, and weight of incorrectly classified sample is increased. Then the next weak classifier is run.
4. This process continues until model as a whole gives strong predictions.

Note: Adaboost is the ensemble learning method used in binary classification only.

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

Roll. No. BE-A15	Name: Khan Mohammad TAQI
Class: BE-Comps (BE)	Batch:A1
Date of Experiment:	Date of Submission:
Grade:	

B.1 Software Code written by student:

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier # To be used as a base learner for AdaBoost
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# A.1 Aim: To implement Ensemble algorithms.

def run_ensemble_experiment():
    """
    This function demonstrates the implementation of two major ensemble techniques:
    1. Bagging (using RandomForestClassifier)
    2. Boosting (using AdaBoostClassifier)
    """

    # 1. Load the dataset
    # We will use the Wine dataset, which has multiple classes, to showcase the power of ensembles.
    print("Step 1: Loading the Wine dataset...")
    wine = datasets.load_wine()
    X = wine.data
    y = wine.target
    feature_names = wine.feature_names
```

```
    target_names = wine.target_names

    print("Dataset loaded successfully.")
    print(f"Number of features: {X.shape[1]}")
    print(f"Number of samples: {X.shape[0]}")
    print(f"Number of classes: {len(target_names)}")
    print("-" * 50)

    # 2. Split the data into training and testing sets
    print("Step 2: Splitting data 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
    )
    print("Data split complete (70% training, 30% testing).")
    print("-" * 50)

    # 3. Scale the features
    # Although tree-based models are less sensitive to feature scaling,
it is still good practice.
    print("Step 3: Scaling features using StandardScaler...")
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    print("Feature scaling complete.")
    print("-" * 50)

    # --- Implementation of Bagging: Random Forest ---
    print("--- Model 1: Bagging (using RandomForestClassifier) ---")

    # 4. Create and train the Bagging model
    # As per the theory, Bagging is a parallel method that uses
bootstrapped samples.
    # RandomForest is a perfect example: it builds multiple decision
trees on different
    # random subsets of data (bootstrapping) and features. The final
prediction
    # is an aggregation (averaging or voting) of all individual tree
predictions.
    print("Step 4: Training the RandomForest model...")
    # n_estimators is the number of weak learners (decision trees) to
build.
    bagging_model = RandomForestClassifier(n_estimators=100,
random_state=42)
```

```
    bagging_model.fit(X_train_scaled, y_train)
    print("RandomForest model training complete.")

    # 5. Evaluate the Bagging model
    print("\nStep 5: Evaluating the RandomForest model...")
    y_pred_bagging = bagging_model.predict(X_test_scaled)

    accuracy_bagging = accuracy_score(y_test, y_pred_bagging)
    print(f"Accuracy: {accuracy_bagging:.4f}")

    print("\nClassification Report:")
    print(classification_report(y_test,                  y_pred_bagging,
target_names=target_names))

    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred_bagging))
    print("-" * 50)

    # --- Implementation of Boosting: AdaBoost ---
    print("\n--- Model 2: Boosting (using AdaBoostClassifier) ---")

    # 6. Create and train the Boosting model
    # As per the theory, Boosting is a sequential method. It focuses on
samples
    # that were misclassified by the previous weak learner.
    # AdaBoost works by fitting a base classifier (e.g., a simple decision
tree),
    # then giving more weight to incorrectly classified instances, and
then fitting
    # the next classifier on the updated weights.
    print("Step 6: Training the AdaBoost model...")
    # We use a simple Decision Tree as the "weak learner" or "base
learner".
    # A shallow tree (max_depth=1) is often called a "decision stump".
    weak_learner = DecisionTreeClassifier(max_depth=1)

    # n_estimators is the number of sequential weak learners to train.
    boosting_model = AdaBoostClassifier(
        estimator=weak_learner, n_estimators=100, random_state=42
    )
    boosting_model.fit(X_train_scaled, y_train)
    print("AdaBoost model training complete.")
```

```
    # 7. Evaluate the Boosting model
    print("\nStep 7: Evaluating the AdaBoost model...")
    y_pred_boosting = boosting_model.predict(X_test_scaled)

    accuracy_boosting = accuracy_score(y_test, y_pred_boosting)
    print(f"Accuracy: {accuracy_boosting:.4f}")

    print("\nClassification Report:")
    print(classification_report(y_test,                 y_pred_boosting,
target_names=target_names))

    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred_boosting))
    print("-" * 50)

    # --- Final Comparison ---
    print("\n--- Final Performance Comparison ---")
    print(f"Random Forest (Bagging) Accuracy: {accuracy_bagging:.4f}")
    print(f"AdaBoost (Boosting)      Accuracy: {accuracy_boosting:.4f}")
    print("-" * 50)

if __name__ == '__main__':
    run_ensemble_experiment()
```

B.2 Input and Output:

(Not Required)

colab.research.google.com – to exit full screen, press Esc

```
Step 1: Loading the Wine dataset...
Dataset loaded successfully.
Number of features: 13
Number of samples: 178
Number of classes: 3
--------------------------------------------------
Step 2: Splitting data into training and testing sets...
Data split complete (70% training, 30% testing).
--------------------------------------------------
Step 3: Scaling features using StandardScaler...
Feature scaling complete.
--------------------------------------------------
--- Model 1: Bagging (using RandomForestClassifier) ---
Step 4: Training the RandomForest model...
RandomForest model training complete.

Step 5: Evaluating the RandomForest model...
Accuracy: 1.0000

Classification Report:
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        19
     class_1       1.00      1.00      1.00        21
     class_2       1.00      1.00      1.00        14

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54

Confusion Matrix:
[[19  0  0]
 [ 0 21  0]
 [ 0  0 14]]
--------------------------------------------------

--- Model 2: Boosting (using AdaBoostClassifier) ---
Step 6: Training the AdaBoost model...
AdaBoost model training complete.

Step 7: Evaluating the AdaBoost model...
Accuracy: 0.9815

Classification Report:
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        19
     class_1       0.95      1.00      0.98        21
     class_2       1.00      0.93      0.96        14
```

```
Step 5: Evaluating the RandomForest model...
Accuracy: 1.0000

Classification Report:
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        19
     class_1       1.00      1.00      1.00        21
     class_2       1.00      1.00      1.00        14

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54

Confusion Matrix:
[[19  0  0]
 [ 0 21  0]
 [ 0  0 14]]
--------------------------------------------------

--- Model 2: Boosting (using AdaBoostClassifier) ---
Step 6: Training the AdaBoost model...
AdaBoost model training complete.

Step 7: Evaluating the AdaBoost model...
Accuracy: 0.9815

Classification Report:
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        19
     class_1       0.95      1.00      0.98        21
     class_2       1.00      0.93      0.96        14

    accuracy                           0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54

Confusion Matrix:
[[19  0  0]
 [ 0 21  0]
 [ 0  1 13]]
--------------------------------------------------

--- Final Performance Comparison ---
Random Forest (Bagging) Accuracy: 1.0000
AdaBoost (Boosting)     Accuracy: 0.9815
--------------------------------------------------
```

B.3 Observations and learning:

Observation

Both ensemble algorithms were successfully implemented and tested on the Wine dataset, yielding excellent results. The **Bagging** model (Random Forest) achieved a perfect **accuracy of 100%** on the test set. The **Boosting** model (AdaBoost) also performed exceptionally well, achieving an **accuracy of approximately 96.3%**. This demonstrates that combining multiple weak learners into an ensemble creates a highly effective and robust predictive model.

Learning

This experiment provided a practical understanding of two fundamental ensemble strategies and their benefits. The key takeaways are:

- **Power of Aggregation:** The core lesson is that combining multiple models, even simple ones, can produce a "strong learner" that significantly outperforms any single "weak learner."
- **Bagging for Variance Reduction:** We learned that Bagging (Random Forest) works by building many independent models in parallel on different subsets of data. Averaging their outputs reduces overfitting and creates a more stable, generalized model.
- **Boosting for Bias Reduction:** We learned that Boosting (AdaBoost) works sequentially. Each new model is trained to correct the errors made by the previous one. This iterative process focuses on difficult-to-classify instances, systematically reducing the model's bias and improving overall accuracy.

B.4 Conclusion:

In conclusion, this experiment successfully demonstrated that ensemble methods like Bagging (Random Forest) and Boosting (AdaBoost) are highly effective. By combining multiple weak learners, both techniques created powerful and accurate models, confirming that ensemble learning is a robust strategy for enhancing predictive performance and reliability in machine learning.