**Assignment Code: DA-AG-014**

Ensemble Learning | **Assignment**

**Instructions:** Carefully read each question. Use Google Docs, Microsoft Word, or a similar tool to create a document where you type out each question along with its answer. Save the document as a PDF, and then upload it to the LMS. Please do not zip or archive the files before uploading them. Each question carries 20 marks.

**Total Marks**: 200

**Question 1:** What is Ensemble Learning in machine learning? Explain the key idea behind it.

**Answer:**

Ensemble Learning is a technique in machine learning where multiple models (often called "weak learners") are combined to create a stronger, more accurate model (the "ensemble").

Key Idea Behind Ensemble Learning

- Diversity and Combination: Ensemble methods leverage the diversity of multiple models to improve overall performance. By combining predictions (through voting, averaging, etc.), the ensemble reduces the impact of individual model errors.

- Improving Accuracy and Robustness: Ensembles can improve accuracy over single models and make the overall model more robust to noise and overfitting.

**Question 2:** What is the difference between Bagging and Boosting?

**Answer:**

Bagging and Boosting are two popular ensemble learning techniques used in machine learning. Here's how they differ:

- **Purpose and Approach:**

- Bagging (Bootstrap Aggregating): Reduces variance by training models on different bootstrap samples of the data and averaging predictions.

- Boosting: Reduces bias by sequentially training models, with each new model focusing on the errors of the previous ones.

- **Model Training:**

- Bagging: Models are trained independently on different data subsets.

- Boosting: Models are trained sequentially, with later models correcting earlier ones.

- **Effect on Performance:**

- Bagging: Helps reduce overfitting and variance.

- Boosting: Can reduce bias and improve accuracy but might lead to overfitting if not tuned properly.

**Examples of Algorithms**

- Bagging: Random Forest (an ensemble of decision trees).

- Boosting: AdaBoost, Gradient Boosting Machines (like XGBoost).

**Question 3:** What is bootstrap sampling and what role does it play in Bagging methods like Random Forest?

**Answer:**

Bootstrap sampling is a statistical technique where you create multiple subsets of data by sampling with replacement from the original dataset. Each bootstrap sample is typically the same size as the original dataset but might contain duplicates of some data points and omit others.

**Role in Bagging Methods like Random Forest**

In Bagging (Bootstrap Aggregating) methods like Random Forest:

- Creating Diverse Trees: Each decision tree in the Random Forest is trained on a different bootstrap sample of the data. This introduces diversity among the trees.

- Reducing Overfitting: By averaging predictions from trees trained on different bootstrap samples, Bagging reduces variance and helps prevent overfitting.

- Improving Stability: The overall model becomes more stable and often more accurate due to the aggregation of multiple trees.

**Question 4:** What are Out-of-Bag (OOB) samples and how is OOB score used to evaluate ensemble models?

**Answer:**

In Bagging methods like Random Forest, Out-of-Bag (OOB) samples refer to the data points that are not included in a particular bootstrap sample used to train a tree. On average, about 1/3 of the data points are left out of each bootstrap sample.

**OOB Score for Evaluating Ensemble Models**

The OOB score is an estimate of the model's performance:

- How it's calculated: For each data point, use only the trees where that data point was OOB (not in the bootstrap sample used for training) to make predictions. Aggregate these predictions to get an OOB estimate for that data point.

- Use for evaluation: The OOB score (like OOB accuracy for classification) gives an unbiased estimate of the model's generalization performance without needing a separate test set.

**Question 5:** Compare feature importance analysis in a single Decision Tree vs. a Random Forest.

**Answer:**

Feature importance can be calculated differently in a single Decision Tree versus a Random Forest.

- **Single Decision Tree:**

- Feature importance is based on how much each feature contributes to reducing impurity (like Gini impurity for classification) in the tree.

- Can be highly dependent on the specific tree structure and prone to overfitting.

- **Random Forest:**

- Feature importance is typically averaged across all trees in the forest.

- Provides a more robust and stable estimate of feature importance due to averaging over many trees.

**Question 6:** Write a Python program to:

* Load the Breast Cancer dataset using

sklearn.datasets.load\_breast\_cancer()

* Train a Random Forest Classifier
* Print the top 5 most important features based on feature importance scores.

(*Include your Python code and output in the code box below.*)

**Answer:**

# Import necessary libraries

from sklearn.datasets import load\_breast\_cancer

from sklearn.ensemble import RandomForestClassifier

import pandas as pd

def main():

# Load the Breast Cancer dataset

data = load\_breast\_cancer()

X = pd.DataFrame(data.data, columns=data.feature\_names)

y = data.target

# Train a Random Forest Classifier

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X, y)

# Get feature importances

importances = model.feature\_importances\_

feature\_names = data.feature\_names

# Print top 5 most important features

top\_features = sorted(zip(feature\_names, importances), key=lambda x: x[1], reverse=True)[:5]

print("Top 5 most important features:")

for feature, importance in top\_features:

print(f"{feature}: {importance:.3f}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:

Top 5 most important features:

worst area: 0.139

worst concave points: 0.132

mean concave points: 0.107

worst radius: 0.083

worst perimeter: 0.081

**Question 7**: Write a Python program to:

* Train a Bagging Classifier using Decision Trees on the Iris dataset
* Evaluate its accuracy and compare with a single Decision Tree

(*Include your Python code and output in the code box below.*)

**Answer:**

Python Program: Bagging Classifier vs. Single Decision Tree on Iris Dataset

Here's a Python program that trains a Bagging Classifier using Decision Trees on the Iris dataset, evaluates its accuracy, and compares it with a single Decision Tree.

# Import necessary libraries

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

def main():

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split data into train/test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a single Decision Tree

single\_dt = DecisionTreeClassifier(random\_state=42)

single\_dt.fit(X\_train, y\_train)

single\_dt\_pred = single\_dt.predict(X\_test)

single\_dt\_accuracy = accuracy\_score(y\_test, single\_dt\_pred)

# Train a Bagging Classifier using Decision Trees

bagging\_clf = BaggingClassifier(estimator=DecisionTreeClassifier(), n\_estimators=10, random\_state=42)

bagging\_clf.fit(X\_train, y\_train)

bagging\_pred = bagging\_clf.predict(X\_test)

bagging\_accuracy = accuracy\_score(y\_test, bagging\_pred)

# Compare accuracies

print(f"\*\*Single Decision Tree Accuracy\*\*: {single\_dt\_accuracy:.3f}")

print(f"\*\*Bagging Classifier Accuracy\*\*: {bagging\_accuracy:.3f}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:

\*\*Single Decision Tree Accuracy\*\*: 1.000

\*\*Bagging Classifier Accuracy\*\*: 1.000

**Question 8**: Write a Python program to:

* Train a Random Forest Classifier
* Tune hyperparameters max\_depth and n\_estimators using GridSearchCV
* Print the best parameters and final accuracy

(*Include your Python code and output in the code box below.*)

**Answer:**

# Import necessary libraries

from sklearn.datasets import load\_breast\_cancer

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import accuracy\_score

def main():

# Load the Breast Cancer dataset

data = load\_breast\_cancer()

X = data.data

y = data.target

# Split data into train/test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define hyperparameter grid

param\_grid = {

'max\_depth': [3, 5, 10, None],

'n\_estimators': [50, 100, 200]

}

# Train a Random Forest Classifier with GridSearchCV

model = RandomForestClassifier(random\_state=42)

grid\_search = GridSearchCV(model, param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# Print best parameters and final accuracy

print(f"Best parameters: {grid\_search.best\_params\_}")

best\_model = grid\_search.best\_estimator\_

y\_pred = best\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy with best parameters: {accuracy:.3f}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:  
Best parameters: {'max\_depth': 10, 'n\_estimators': 200}

Accuracy with best parameters: 0.965

**Question 9**: Write a Python program to:

* Train a Bagging Regressor and a Random Forest Regressor on the California Housing dataset
* Compare their Mean Squared Errors (MSE) (*Include your Python code and output in the code box below.*) **Answer:**

# Import necessary libraries

from sklearn.datasets import fetch\_california\_housing

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import BaggingRegressor, RandomForestRegressor

from sklearn.metrics import mean\_squared\_error

def main():

# Load the California Housing dataset

data = fetch\_california\_housing()

X = data.data

y = data.target

# Split data into train/test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train a Bagging Regressor

bagging\_model = BaggingRegressor(n\_estimators=100, random\_state=42)

bagging\_model.fit(X\_train, y\_train)

y\_pred\_bagging = bagging\_model.predict(X\_test)

mse\_bagging = mean\_squared\_error(y\_test, y\_pred\_bagging)

# Train a Random Forest Regressor

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

y\_pred\_rf = rf\_model.predict(X\_test)

mse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf)

# Compare MSEs

print(f"Bagging Regressor MSE: {mse\_bagging:.3f}")

print(f"Random Forest Regressor MSE: {mse\_rf:.3f}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output:

Bagging Regressor MSE: 0.256

Random Forest Regressor MSE: 0.255

**Question 10:** You are working as a data scientist at a financial institution to predict loan default. You have access to customer demographic and transaction history data.

You decide to use ensemble techniques to increase model performance. Explain your step-by-step approach to:

* Choose between Bagging or Boosting
* Handle overfitting
* Select base models
* Evaluate performance using cross-validation
* Justify how ensemble learning improves decision-making in this real-world context.

(*Include your Python code and output in the code box below.*)

**Answer:**

Step-by-Step Approach for Predicting Loan Default Using Ensemble Techniques

1. **Choose Between Bagging or Boosting**

- Considerations: Boosting (like Gradient Boosting) often performs well with structured data and can handle complex relationships. Bagging (like Random Forest) is robust and less prone to overfitting.

- Choice for loan default: Gradient Boosting is often a strong choice due to its performance in many Kaggle competitions for structured data.

2. **Handle Overfitting**

- Techniques: Use regularization parameters (like max\_depth, learning\_rate in Gradient Boosting), early stopping, and cross-validation.

3. **Select Base Models**

- For ensemble: Decision trees. They're weak learners but effective in ensembles.

4. **Evaluate Performance Using Cross-Validation**

- Cross-validation: Use k-fold cross-validation to assess model performance on different data splits.

5. **How Ensemble Learning Improves Decision-Making**

- Better predictions: Ensemble methods reduce overfitting and often increase accuracy.

- Risk assessment: More accurate predictions of loan default help in better risk management for financial institutions.

**Python Code and Output**

# Import necessary libraries

from sklearn.datasets import make\_classification

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier

import numpy as np

# Simulate loan default dataset (for illustration)

X, y = make\_classification(n\_samples=1000, n\_features=10, random\_state=42)

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Gradient Boosting

gb\_model = GradientBoostingClassifier(n\_estimators=100, max\_depth=3, learning\_rate=0.1, random\_state=42)

gb\_model.fit(X\_train, y\_train)

# Train Random Forest

rf\_model = RandomForestClassifier(n\_estimators=100, max\_depth=3, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Evaluate using cross-validation

gb\_scores = cross\_val\_score(gb\_model, X\_train, y\_train, cv=5)

rf\_scores = cross\_val\_score(rf\_model, X\_train, y\_train, cv=5)

print(f"Gradient Boosting CV Accuracy: {np.mean(gb\_scores):.3f}")

print(f"Random Forest CV Accuracy: {np.mean(rf\_scores):.3f}")

Output:

Gradient Boosting CV Accuracy: 0.900

Random Forest CV Accuracy: 0.878