Assignment Code: DA-AG-009

Supervised Classification: Decision Trees, SVM, and Naive Bayes| **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 Information Gain, and how is it used in Decision Trees?

Answer:

- Information Gain is a measure that tells how much a feature reduces the impurity or uncertainty in the data.
- It is based on the concept of entropy, which calculates the level of disorder in a dataset.
- Information Gain is calculated as: Entropy before split − Entropy after split.
- A Decision Tree computes the Information Gain for every feature in the dataset.
- The feature with the highest Information Gain is chosen for splitting the data at that node.
- A higher Information Gain means the feature separates the classes more clearly.
- Therefore, Decision Trees use Information Gain to create the most accurate and pure splits while building the tree.

Question 2: What is the difference between Gini Impurity and Entropy?

Hint: Directly compares the two main impurity measures, highlighting strengths, weaknesses, and appropriate use cases.

Answer:

- Gini Impurity and Entropy are both measures used to calculate how impure or mixed a dataset is.
- Gini Impurity measures the probability of incorrectly classifying a randomly chosen sample.
- Entropy measures the amount of uncertainty or randomness in the dataset.
- Gini is calculated using squared probabilities, while Entropy uses log-based probabilities.
- Gini is faster to compute because it does not use logarithms.
- Entropy is slightly slower but gives a more mathematically precise measure of impurity.
- Gini tends to create splits that isolate the most frequent class first.
- Entropy treats all classes more evenly and may give slightly different split choices.
- Gini is often used in CART (Classification and Regression Trees).
- Entropy is commonly used in ID3 and C4.5 decision tree algorithms.
- Both work well, but Gini is preferred when speed matters, while Entropy is preferred for more balanced, information-theoretic analysis**.**

Question 3:What is Pre-Pruning in Decision Trees?

Answer:

1. Pre-pruning is a technique used to stop the Decision Tree from growing too large during the training process.
2. It prevents the tree from creating splits that may lead to overfitting.
3. In pre-pruning, we apply certain conditions before making a split.
4. Examples of these conditions include minimum number of samples required to split, maximum depth, or minimum information gain.
5. If a split does not meet these conditions, the tree stops growing at that node.
6. This keeps the tree simpler and helps it generalize better on new data.
7. Pre-pruning improves the model's performance by avoiding unnecessary branches.

Question 4:Write a Python program to train a Decision Tree Classifier using Gini Impurity as the criterion and print the feature importances (practical).

Hint: Use criterion='gini' in DecisionTreeClassifier and access .feature_importances_.

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

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

# Load a sample dataset (Iris dataset)
data = load_iris()
X = data.data
y = data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create Decision Tree Classifier using Gini Impurity
model = DecisionTreeClassifier(criterion='gini', random_state=42)

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

# Print feature importances
print("Feature Importances:")
for name, importance in zip(data.feature_names, model.feature_importances_):
    print(f"{name}: {importance:.4f}")

# Make a prediction (optional)
pred = model.predict(X_test)
print("\nSample Prediction on Test Set:", pred[:5])
```

#OUTPUT

```
Feature Importances:
sepal length (cm): 0.0000
sepal width (cm): 0.0167
petal length (cm): 0.9061
petal width (cm): 0.0772

Sample Prediction on Test Set: [1 0 2 1 1]
```

Question 5: What is a Support Vector Machine (SVM)?

Answer:

- A Support Vector Machine (SVM) is a powerful supervised machine learning algorithm used for classification and regression tasks.
- SVM works by finding the best boundary (called a hyperplane) that separates different classes in the data.
- The goal is to choose the hyperplane that maximizes the margin between the classes.
- Support vectors are the data points that lie closest to the hyperplane and help define its position.
- A larger margin generally leads to better generalization and higher accuracy.
- SVM can handle both linear and non-linear data using kernel functions like linear, polynomial, and RBF.
- SVM is widely used because it performs well in high-dimensional spaces and is effective even with small datasets.

Question 6: What is the Kernel Trick in SVM?

Answer:

- The Kernel Trick is a technique used in SVM to handle non-linear data without explicitly transforming it to a higher dimension.
- It allows the model to create complex decision boundaries in the original input space.
- Instead of calculating new features manually, the kernel function computes the similarity between data points in the higher-dimensional space.
- This makes the computation fast and efficient because no actual transformation is performed.
- Common kernel functions include Linear, Polynomial, and RBF (Radial Basis Function).
- Using the Kernel Trick, SVM can classify data that is not linearly separable.
- It helps SVM learn flexible and powerful decision boundaries while keeping the computation practical.

Question 7: Write a Python program to train two SVM classifiers with Linear and RBF kernels on the Wine dataset, then compare their accuracies.

Hint:Use SVC(kernel='linear') and SVC(kernel='rbf'), then compare accuracy scores after fitting on the same dataset.

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

```
 # Import required libraries
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Wine dataset
data = load_wine()
X = data.data
y = data.target

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train SVM with Linear kernel
svm_linear = SVC(kernel='linear', random_state=42)
svm_linear.fit(X_train, y_train)
y_pred_linear = svm_linear.predict(X_test)
accuracy_linear = accuracy_score(y_test, y_pred_linear)

# Train SVM with RBF kernel
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_train, y_train)
y_pred_rbf = svm_rbf.predict(X_test)
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)

# Print accuracies
print(f"Accuracy with Linear Kernel: {accuracy_linear:.4f}")
print(f"Accuracy with RBF Kernel: {accuracy_rbf:.4f}")

#OUTPUT

Accuracy with Linear Kernel: 1.0000
Accuracy with RBF Kernel: 0.8056
```

Question 8: What is the Naïve Bayes classifier, and why is it called "Naïve"?

Answer:

- Naïve Bayes is a supervised machine learning algorithm used for classification tasks.

- It is based on Bayes' Theorem, which calculates the probability of a class given the input features.

- The algorithm assumes that all features are independent of each other given the class.

- This assumption of independence is rarely true in real-world data, which is why it is called "Naïve."

- Despite this simplification, Naïve Bayes performs well in many practical applications like spam detection and text classification.

Question 9: Explain the differences between Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernoulli Naïve Bayes **Answer:**

- Gaussian Naïve Bayes assumes that the features follow a continuous Gaussian (normal) distribution.

- It is suitable for datasets with continuous numerical features, like height, weight, or temperature.

- Multinomial Naïve Bayes is designed for discrete count data, like word frequencies in text classification.

- It works well for problems where features represent the number of times an event occurs.

- Bernoulli Naïve Bayes works with binary/Boolean features, i.e., features that are either 0 or 1.

- It is often used for text classification with features like "word present" or "word absent."

- In short:

    - Gaussian → continuous numeric data
    - Multinomial → count-based discrete data
    - Bernoulli → binary/Boolean data

Question 10: Breast Cancer Dataset

Write a Python program to train a Gaussian Naïve Bayes classifier on the Breast Cancer dataset and evaluate accuracy.

Hint:Use GaussianNB() from sklearn.naive_bayes and the Breast Cancer dataset from sklearn.datasets.

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

```
 # Import required libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create Gaussian Naïve Bayes classifier
gnb = GaussianNB()

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

# Make predictions on test set
y_pred = gnb.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Gaussian Naïve Bayes: {accuracy:.4f}")

#OUTPUT

Accuracy of Gaussian Naïve Bayes: 0.9737
```