

Question 1 : What is Information Gain, and how is it used in Decision Trees?

Answer:

Information Gain is a metric used in **Decision Trees** to decide which feature should be used to split the data at each node.

It measures the **reduction in uncertainty (entropy)** about the target variable after splitting the dataset based on a particular feature.

Key Points:

- **Entropy** measures the impurity or randomness in a dataset.
- **Information Gain** tells us how much entropy decreases after a split.
- The feature with the **highest Information Gain** is chosen for splitting the node.

How it is used in Decision Trees:

1. Calculate the entropy of the full dataset.
2. For each feature, calculate the entropy after splitting.
3. Compute Information Gain for each feature.
4. Select the feature with the **maximum Information Gain** to split the node.

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 two popular impurity measures used in Decision Trees .

Gini Impurity :

- Measures the probability of misclassifying a random sample .
- Formula: $Gini = 1 - \sum(p^2)$ where p is the probability of each class.
- Faster computation , often preferred for large datasets.

Entropy :

- Measures the uncertainty or randomness in the dataset .
- Formula: $Entropy = -\sum(p * \log_2(p))$ where p is the probability of each class.
- More sensitive to class distribution changes, useful for multi-class problems .

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

Answer:

Pre-Pruning in Decision Trees

Pre-pruning (also called early stopping) is a technique used in Decision Trees to prevent overfitting by stopping the growth of the tree before it becomes too complex.

Explanation:

Instead of allowing the decision tree to grow fully and then trimming it later, pre-pruning sets rules in advance to decide when the tree should stop splitting.

Common Pre-Pruning Criteria:

- Maximum depth of the tree
- Minimum number of samples required to split a node
- Minimum number of samples required at a leaf node
- Minimum Information Gain (or Gini decrease) required for a split

Why Pre-Pruning is Used:

- Reduces overfitting
- Improves generalization to unseen data
- Decreases training time and model complexity

Limitation:

- May stop the tree too early, leading to **underfitting**

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.datasets import load_iris
from sklearn.model_selection import train_test_split

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

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train Decision Tree with Gini Impurity
model = DecisionTreeClassifier(criterion='gini', random_state=42)
model.fit(X_train, y_train)

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

Output :

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

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

Answer:

Support Vector Machine (SVM)

A Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks.

Explanation:

SVM works by finding an optimal hyperplane that best separates data points of different classes in the feature space. The goal is to maximize the margin, which is the distance between the hyperplane and the closest data points from each class, called support vectors.

Key Concepts:

- **Hyperplane:** A decision boundary that separates classes
- **Margin:** The maximum distance between the hyperplane and nearest data points
- **Support Vectors:** Data points closest to the hyperplane that influence its position

Types of SVM:

- **Linear SVM:** Used when data is linearly separable
- **Non-linear SVM:** Uses kernel functions (e.g., polynomial, RBF) to handle complex data

Advantages:

- Effective in high-dimensional spaces
- Works well when the number of features is greater than the number of samples

Disadvantages:

- Computationally expensive for very large datasets
- Choice of kernel and parameters is crucial

Question 6: What is the Kernel Trick in SVM?

Answer:

Kernel Trick in Support Vector Machine (SVM)

The Kernel Trick is a technique used in Support Vector Machines (SVMs) to handle non-linearly separable data by implicitly mapping it into a higher-dimensional feature space.

Explanation:

When data cannot be separated by a straight line (or hyperplane) in its original space, the kernel trick allows SVM to compute the separation as if the data were transformed into a higher dimension—without explicitly performing the transformation. This makes the computation efficient.

Common Kernel Functions:

- Linear Kernel – for linearly separable data
- Polynomial Kernel – captures polynomial relationships
- RBF (Gaussian) Kernel – handles complex, curved decision boundaries
- Sigmoid Kernel – similar to neural networks

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:

```
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
from sklearn.preprocessing import StandardScaler

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

```

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Feature scaling (important for SVM)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

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

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

# Calculate accuracies
linear_accuracy = accuracy_score(y_test, y_pred_linear)
rbf_accuracy = accuracy_score(y_test, y_pred_rbf)

print("Linear Kernel SVM Accuracy:", linear_accuracy)
print("RBF Kernel SVM Accuracy:", rbf_accuracy)

```

Output:

```

Linear Kernel SVM Accuracy: 0.9722222222222222
RBF Kernel SVM Accuracy: 1.0

```

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

Answer:

Naïve Bayes Classifier

The **Naïve Bayes classifier** is a **supervised machine learning algorithm** based on **Bayes' Theorem**, mainly used for **classification tasks** such as text classification and spam detection.

Why it is called "Naïve":

It is called **naïve** because it makes a **strong simplifying assumption** that **all features are conditionally independent** of each other given the class label.

In real-world data, this assumption is often not true, but the model still performs surprisingly well.

How it works:

Naïve Bayes calculates the probability of a class given the input features and assigns the class with the **highest posterior probability**.

$$P(\text{Class} | \text{Features}) \propto P(\text{Class}) \times \prod_i P(\text{Feature}_i | \text{Class})$$

$P(\text{Class} | \text{Features}) \propto P(\text{Class}) \times \prod_i P(\text{Feature}_i | \text{Class})$

Advantages:

- Simple and fast to train
- Works well with high-dimensional data
- Performs well even with small datasets

Limitations:

Independence assumption is often unrealistic

Can perform poorly if features are highly correlated

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

Answer:

Differences Between Gaussian, Multinomial, and Bernoulli Naïve Bayes

Naïve Bayes classifiers differ mainly in the type of data they assume for the features and how probabilities are computed.

1. Gaussian Naïve Bayes

- Assumption: Features follow a normal (Gaussian) distribution
- Type of Data: Continuous numerical features
- Example Use Case: Height, weight, temperature, medical measurements

- Strengths: Works well with real-valued data
- Limitation: Assumes normal distribution, which may not always hold

2. Multinomial Naïve Bayes

- Assumption: Features represent counts or frequencies
 - Type of Data: Discrete count data
 - Example Use Case: Text classification (word counts, TF-IDF)
 - Strengths: Excellent for document and NLP tasks
 - Limitation: Not suitable for continuous features
-

3. Bernoulli Naïve Bayes

- Assumption: Features are binary (0 or 1)
- Type of Data: Binary / Boolean data
- Example Use Case: Presence or absence of a word in text
- Strengths: Effective when binary features matter
- Limitation: Ignores frequency information

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:

```
port load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

```
# Load the 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
)

# Train Gaussian Naïve Bayes model
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Make predictions
y_pred = gnb.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

print("Gaussian Naïve Bayes Accuracy:", accuracy)
```

Output:

Gaussian Naïve Bayes Accuracy: 0.9736842105263158