Question 1: What is a Decision Tree, and how does it work in the context of classification?

Answer: -A Decision Tree is a supervised machine learning algorithm used for classification (and regression) tasks. It models decisions and their possible consequences as a tree-like structure of nodes, branches, and leaves.

It Work in Classification:-

1.Starting at the root node, the tree looks at the input features of the data and decides which feature to split on.

2.Splitting criteria are chosen to best separate the data into classes. This is often done by metrics like:
i.Gini impurity
ii.Information gain (based on entropy)

3.The dataset is divided into subsets based on the selected feature's values.

4.This process repeats recursively for each subset (child node), creating branches and new nodes.

5.When a node is "pure" (contains data mostly from one class) or meets stopping criteria (e.g., max depth or minimum samples), it becomes a leaf node.

6.At classification time, you start at the root and follow the branches according to the feature values of the instance, reaching a leaf node which assigns the class label.

Question 2: Explain the concepts of Gini Impurity and Entropy as impurity measures. How do they impact the splits in a Decision Tree?

Answer:-Both are metrics to measure how “mixed” a set of samples is in terms of class labels. The goal of the decision tree when splitting is to reduce impurity — to make child nodes more “pure,” meaning they mostly contain samples from a single class.

1. Gini Impurity:- Measures the probability of incorrectly classifying a randomly chosen element if it were randomly labeled according to the distribution of labels in the node.

$$Gini = 1 - \sum_{i=1}^{C} pi^2$$

Where pi is the proportion of class iii in the node, and CCC is the number of classes.

Range: 0 (pure node) to maximum close to 0.5 for two classes (most impure).
Interpretation: Lower Gini means better purity.

2. Entropy (Information Entropy):- Measures the amount of disorder or uncertainty in the node.

$$Entropy = -\sum_{i=1}^{C} pi \log_2 pi$$

Range: 0 (pure node) to $\log_2 C$ (maximum disorder).

Interpretation: Lower entropy means better purity (less uncertainty).

Question 3: What is the difference between Pre-Pruning and Post-Pruning in Decision Trees? Give one practical advantage of using each.

Answer:-

Pre-Pruning (Early Stopping):-Pre-pruning stops the growth of the decision tree early, before it becomes too complex. During the building process, the algorithm decides whether to split a node or not based on some criteria.

How it works:

Before creating a new split, it checks if the split improves the model enough or meets conditions like:
i.Minimum number of samples in a node
ii.Maximum depth reached
iii.Impurity reduction is too small

Practical advantage:

Because the tree stops growing early, training is faster and the tree is smaller, which makes it easier to interpret and less prone to overfitting from the start.

Post-Pruning (Pruning After Training):- Post-pruning grows the full decision tree completely without restrictions. After that, it prunes (removes) some branches that do not improve performance on validation data.

How it works:

The fully grown tree is analyzed, and subtrees or branches that don't contribute to better accuracy or generalization are trimmed away.

Practical advantage:

This approach often leads to better accuracy and generalization because the model first learns all the patterns and then removes unnecessary complexity, which helps avoid overfitting while preserving important information.

Question 4: What is Information Gain in Decision Trees, and why is it important for choosing the best split?

Answer: -Information Gain is a metric used to measure how much uncertainty (impurity) is reduced in the dataset after a split based on a feature.

Important

1.Helps choose the best feature to split on:-The feature with the highest Information Gain is selected at each step in building the tree.

2.Guides the tree to become more accurate:By picking splits that

reduce impurity the most, the tree becomes better at separating classes.

3.Controls tree growth intelligently:

Instead of randomly splitting or using all features equally, Information Gain ensures each split adds meaningful value.

Question 5: What are some common real-world applications of Decision Trees, and what are their main advantages and limitations?

Answer:-

Main Advantages of Decision Trees:

1. **Easy to understand and interpret**
   Visual and intuitive; no complex math required to explain decisions.

2. **Works with both numerical and categorical data**
   No need to scale or normalize features.

3. **Handles missing values**
   Can still function even if some inputs are missing.

4. **Fast to train and make predictions**
   Suitable for real-time or large datasets.

Main Limitations of Decision Trees:

1. **Prone to overfitting**
   Trees can become too complex and memorize the training data, reducing accuracy on new data.

2. **Unstable with small data changes**
   A small change in the input data can lead to a very different tree structure.

3. **Bias toward features with many levels**
   May prefer attributes with more categories, even if they're not the best.

4. **Less accurate than ensemble methods**
   Single decision trees often perform worse than models like Random Forests or Gradient Boosting.

Question 6: Write a Python program to:

- **Load the Iris Dataset**
- **Train a Decision Tree Classifier using the Gini criterion**
- **Print the model's accuracy and feature importances**

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

Answer:-

```
# Import required libraries
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names

# 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)

# Train a Decision Tree Classifier using Gini criterion
clf = DecisionTreeClassifier(criterion='gini', random_state=42)
clf.fit(X_train, y_train)
```

```
# Predict on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print results
print("Model Accuracy: {:.2f}%".format(accuracy * 100))
print("\nFeature Importances:")
for name, importance in zip(feature_names,
clf.feature_importances_):
    print(f"{name}: {importance:.4f}")
```

output:-

```
Model Accuracy: 100.00%

Feature Importances:
sepal length (cm): 0.0000
sepal width (cm): 0.0000
petal length (cm): 0.4531
petal width (cm): 0.5469
```

Question 7: Write a Python program to:

● Load the Iris Dataset

● Train a Decision Tree Classifier with max_depth=3 and compare its accuracy to a fully-grown tree.

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

Answer:-

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and test sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train Decision Tree with max_depth=3 (pre-pruned tree)
dt_limited = DecisionTreeClassifier(max_depth=3,
random_state=42)
dt_limited.fit(X_train, y_train)
y_pred_limited = dt_limited.predict(X_test)
accuracy_limited = accuracy_score(y_test, y_pred_limited)

# Train fully-grown Decision Tree (no max_depth)
dt_full = DecisionTreeClassifier(random_state=42)
dt_full.fit(X_train, y_train)
y_pred_full = dt_full.predict(X_test)
accuracy_full = accuracy_score(y_test, y_pred_full)

# Print the results
print("Accuracy with max_depth=3: {:.2f}%".format(accuracy_limited
* 100))
print("Accuracy with fully-grown tree: {:.2f}%".format(accuracy_full *
100))
```

output:-

```
Accuracy with max_depth=3: 100.00%
Accuracy with fully-grown tree: 100.00%
```

Question 8: Write a Python program to:

- **Load the California Housing dataset from sklearn**
- **Train a Decision Tree Regressor**

● Print the Mean Squared Error (MSE) and feature importances (Include your Python code and output in the code box below.)

Answer:-

```
# Import necessary libraries
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load the California Housing dataset
data = fetch_california_housing()
X = data.data
y = data.target
feature_names = data.feature_names

# Split data into training and testing sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train a Decision Tree Regressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

# Predict on the test set
y_pred = regressor.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Print results
print("Mean Squared Error (MSE): {:.4f}".format(mse))
print("\nFeature Importances:")
for name, importance in zip(feature_names,
regressor.feature_importances_):
    print(f"{name}: {importance:.4f}")
```

output:-
Mean Squared Error (MSE): 0.5402

Feature Importances:
MedInc: 0.5812
HouseAge: 0.0306
AveRooms: 0.0773
AveBedrms: 0.0229
Population: 0.0304
AveOccup: 0.0296
Latitude: 0.1056
Longitude: 0.1224

Question 9: Write a Python program to:
● Load the Iris Dataset
● Tune the Decision Tree's max_depth and min_samples_split using GridSearchCV
● Print the best parameters and the resulting model accuracy (Include your Python code and output in the code box below.)
Answer:-

```
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import accuracy_score

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

# Split into training and testing sets (80/20)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Define parameter grid for GridSearch
param_grid = {
    'max_depth': [2, 3, 4, 5, None],
    'min_samples_split': [2, 3, 4, 5]
}

# Initialize the Decision Tree Classifier
dt = DecisionTreeClassifier(random_state=42)

# Use GridSearchCV for tuning
grid_search = GridSearchCV(estimator=dt,
param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best model
best_model = grid_search.best_estimator_

# Predict and evaluate
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Print results
print("Best Parameters:", grid_search.best_params_)
print("Model Accuracy on Test Set: {:.2f}%".format(accuracy * 100))
```

Output:-

```
Best Parameters: {'max_depth': 3, 'min_samples_split': 2}
Model Accuracy on Test Set: 100.00%
```

Question 10: Imagine you're working as a data scientist for a healthcare company that wants to predict whether a patient has a certain disease. You have a large dataset with mixed data

types and some missing values. Explain the step-by-step process you would follow to:
● Handle the missing values
● Encode the categorical features
● Train a Decision Tree model
● Tune its hyperparameters
● Evaluate its performance And describe what business value this model could provide in the real-world setting.
Answer:-
1. Handle the Missing Values
Numerical Features:
Use mean or median imputation:
```
from sklearn.impute import SimpleImputer
num_imputer = SimpleImputer(strategy='median')
X_num = num_imputer.fit_transform(X_num)
```
Categorical Features:
Use most frequent or create a special category like `"Missing"`:
```
cat_imputer = SimpleImputer(strategy='most_frequent')
X_cat = cat_imputer.fit_transform(X_cat)
```
Optional (advanced):
Use KNN imputation or model-based imputation for better accuracy.

2. Encode the Categorical Features

Use One-Hot Encoding for nominal (unordered) categories:

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(handle_unknown='ignore')
X_cat_encoded = encoder.fit_transform(X_cat)
```

Use Ordinal Encoding for ordered categories, if they exist

3. Train the Decision Tree Model

i.Combine preprocessed numerical and encoded categorical data
ii.Train a Decision Tree:

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
```

4. Tune Hyperparameters

Use GridSearchCV or RandomizedSearchCV to find optimal values for:

`max_depth`

`min_samples_split`

`min_samples_leaf`

`criterion` (`gini` or `entropy`)

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search =
GridSearchCV(DecisionTreeClassifier(random_state=42),
param_grid, cv=5)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
```

5. Evaluate Model Performance

Use a separate test set (or cross-validation) to evaluate:

1.Accuracy
2.Precision, Recall, F1-score
3.ROC AUC (for binary classification)
4.Confusion Matrix

```
from sklearn.metrics import classification_report, roc_auc_score
y_pred = best_model.predict(X_test)
print(classification_report(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test,
best_model.predict_proba(X_test)[:, 1]))
```