Decision Tree Assignment

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

**Answer:**

**1. Definition of a Decision Tree**

A **Decision Tree** is a supervised machine learning algorithm used for both **classification** and **regression** tasks.

* It represents decisions in the form of a **tree-like structure**, where each **internal node** corresponds to a test on a feature, each **branch** represents an outcome of that test, and each **leaf node** represents a final decision or class label.
* It mimics human decision-making by breaking down a complex decision-making process into a series of simpler decisions.

**2. How it works in the context of Classification**

In classification problems, a Decision Tree is used to assign an input data point to one of the predefined classes. The process involves:

1. **Root Node Selection**
   * The algorithm starts at the root node with the full dataset.
   * It chooses the **best feature** to split the data. The “best” is determined using criteria like:
   * **Information Gain** (based on Entropy) – used in ID3 and C4.5 algorithms.
   * **Gini Index** – used in CART (Classification and Regression Tree).
2. **Splitting the Data**
   * Based on the chosen feature, the dataset is divided into subsets.
   * For example, if the feature is “Age,” it may split into groups like “Age < 30” and “Age ≥ 30.”
3. **Recursive Partitioning**
   * The process repeats for each child node using only the subset of data that belongs to that node.
   * The splitting continues until one of the stopping conditions is met (e.g., maximum depth reached, all samples belong to one class, or no further gain in splitting).
4. **Leaf Nodes**
   * When no further splitting is possible, a leaf node is created with the final **class label**.

A **Decision Tree** is a tree-like model used for decision-making. In classification, it splits the dataset step by step based on feature values, until the data points are categorized into a class label. It is one of the most intuitive and widely used algorithms in machine learning.

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

**Answer:**

**1. What are Impurity Measures in Decision Trees?**

In Decision Trees, the goal is to split the dataset in such a way that each branch (node) becomes as “pure” as possible.

* **Pure Node** → All samples belong to one class.
* **Impure Node** → Mixed samples (e.g., some “Yes,” some “No”).

To decide the **best split**, impurity measures such as **Gini Impurity** and **Entropy** are used.

**2. Gini Impurity**

* **Definition:** Gini impurity measures the probability of incorrectly classifying a randomly chosen element if it was randomly labeled according to the distribution of labels in the node.
* **Formula:**

Gini=1−∑i=1kpi2Gini = 1 - \sum\_{i=1}^{k} p\_i^2Gini=1−i=1∑k​pi2​

Where:

* \_ipi​ = proportion of samples belonging to class iii
* kkk = number of classes
* **Range:** 0 (pure) to 0.5 (maximum impurity for binary classes).
* **Example:**  
  Suppose a node has 10 samples → 7 “Yes” and 3 “No.”

p(Yes)=0.7,  p(No)=0.3p(Yes) = 0.7, \; p(No) = 0.3p(Yes)=0.7,p(No)=0.3 Gini=1−(0.72+0.32)=1−(0.49+0.09)=0.42Gini = 1 - (0.7^2 + 0.3^2) = 1 - (0.49 + 0.09) = 0.42Gini=1−(0.72+0.32)=1−(0.49+0.09)=0.42

Lower Gini means better purity.

**3. Entropy (Information Gain)**

* **Definition:** Entropy measures the disorder or uncertainty in a dataset. A pure node has entropy = 0.
* **Formula:**

Entropy=−∑i=1kpi⋅log⁡2(pi)Entropy = - \sum\_{i=1}^{k} p\_i \cdot \log\_2(p\_i)Entropy=−i=1∑k​pi​⋅log2​(pi​)

* **Range:** 0 (pure) to 1 (highly impure, in binary classification).
* **Example:**  
  For the same node (7 “Yes,” 3 “No”):

p(Yes)=0.7,  p(No)=0.3p(Yes) = 0.7, \; p(No) = 0.3p(Yes)=0.7,p(No)=0.3 Entropy=−(0.7⋅log⁡2(0.7)+0.3⋅log⁡2(0.3))Entropy = -(0.7 \cdot \log\_2(0.7) + 0.3 `\cdot \log\_2(0.3))Entropy=−(0.7⋅log2​(0.7)+0.3⋅log2​(0.3)) =−(0.7⋅−0.515+0.3⋅−1.737)= -(0.7 \cdot -0.515 + 0.3 \cdot -1.737)=−(0.7⋅−0.515+0.3⋅−1.737) =0.881= 0.881=0.881

Lower entropy means better purity.

**4. Impact on Splits in a Decision Tree**

When building a Decision Tree, the algorithm chooses the **feature split** that produces the **greatest reduction in impurity**.

* Using **Gini Impurity** (CART algorithm):  
  The algorithm selects the split that minimizes the Gini index.
* Using **Entropy/Information Gain** (ID3, C4.5 algorithms):  
  The algorithm selects the split that maximizes **Information Gain**, where:

Information  Gain=Entropy(parent)−∑nchildnparent⋅Entropy(child)Information \; Gain = Entropy(parent) - \sum \frac{n\_{child}}{n\_{parent}} \cdot Entropy(child)InformationGain=Entropy(parent)−∑nparent​nchild​​⋅Entropy(child)

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

**Answer:**

**1. Pruning in Decision Trees**

* **Pruning** is the process of reducing the size of a Decision Tree by removing branches that provide little to no predictive power.
* It helps to **avoid overfitting** and improves the generalization of the model.
* There are two main types: **Pre-Pruning** and **Post-Pruning**.

**2. Pre-Pruning (Early Stopping)**

* **Definition:** Pre-pruning stops the tree from growing too deep during the construction phase.
* The algorithm applies a **stopping condition** before a node becomes too specific.

**Common stopping conditions:**

Maximum depth of the tree.

* Minimum number of samples required to split a node.
* Minimum information gain or impurity decrease.

**Practical Advantage:**

* **Faster training time** → Since the tree does not grow unnecessarily deep.
* Example: In a dataset of 10,000 customers, we can stop splitting a node if fewer than 50 customers remain, saving computation and avoiding over-complexity

**3. Post-Pruning (Prune After Full Growth)**

* **Definition:** In post-pruning, the tree is first allowed to grow fully (possibly overfitting), and then the **irrelevant branches are removed** afterward.
* This is done by testing the accuracy of subtrees on a validation set and cutting the branches that do not improve performance.

**Practical Advantage:**

* **Better accuracy on unseen data** → Because pruning removes overfitted branches.
* Example: A tree predicting whether students pass or fail may overfit by creating deep rules like “Students with 72 marks and 3 absences → Pass.” Post-pruning can cut such overly specific rules and generalize better.

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

**Answer:**

**1. Definition of Information Gain**

* **Information Gain (IG)** is a metric used in Decision Trees to measure how much **uncertainty (entropy)** is reduced after splitting the dataset on a particular feature.
* In simple words: **It tells us which feature gives the most useful information to classify the data.**

**2. Formula**

Information  Gain=Entropy(Parent)−∑i=1knin⋅Entropy(Childi)Information \; Gain = Entropy(Parent) - \sum\_{i=1}^{k} \frac{n\_i}{n} \cdot Entropy(Child\_i)InformationGain=Entropy(Parent)−i=1∑k​nni​​⋅Entropy(Childi​)

Where:

* Entropy(Parent)Entropy(Parent)Entropy(Parent) = impurity before the split.
* Entropy(Childi)Entropy(Child\_i)Entropy(Childi​) = impurity of each child node.
* nin\frac{n\_i}{n}nni​​ = proportion of samples in child node iii.

The **higher the Information Gain**, the better the feature is for splitting.

**3. Practical Example**

Suppose we want to decide whether students **“Pass” or “Fail”** based on **“Study Hours.”**

| **Study Hours** | **Result** |
| --- | --- |
| >5 | Pass |
| >5 | Pass |
| ≤5 | Fail |
| ≤5 | Fail |
| >5 | Pass |

* **Step 1: Parent Node (before split)**
  + 3 Pass, 2 Fail
  + Entropy = −(35log⁡235+25log⁡225)=0.971-\left(\frac{3}{5}\log\_2\frac{3}{5} + \frac{2}{5}\log\_2\frac{2}{5}\right) = 0.971−(53​log2​53​+52​log2​52​)=0.971
* **Step 2: Split on “Study Hours”**
  + Group 1 (>5 hrs): 3 Pass, 0 Fail → Entropy = 0
  + Group 2 (≤5 hrs): 0 Pass, 2 Fail → Entropy = 0
* **Step 3: Weighted Average Entropy**

35(0)+25(0)=0\frac{3}{5}(0) + \frac{2}{5}(0) = 053​(0)+52​(0)=0

* **Step 4: Information Gain**

0.971−0=0.9710.971 - 0 = 0.9710.971−0=0.971

This is **maximum IG**, meaning “Study Hours” is the **best split feature**.

**4. Why is Information Gain Important?**

1. **Helps choose the best feature** for splitting the dataset at each node.
2. **Reduces impurity** → each split makes nodes purer (closer to a single class).
3. **Leads to better accuracy** because the tree makes more meaningful splits.
4. Prevents random or irrelevant splits (e.g., splitting based on “Student’s Shirt Color” would give very low IG).

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

ANSWER

**1. Real-World Applications of Decision Trees**

Decision Trees are widely used in both **classification** and **regression** tasks across industries:

**(a) Classification Tasks**

* **Medical Diagnosis** → Predict whether a patient has a disease based on symptoms (Yes/No).
* **Customer Churn Prediction** → Classify whether a customer will leave a service provider.
* **Email Spam Detection** → Decide whether an email is *Spam* or *Not Spam*.
* **Credit Risk Analysis** → Approve or reject loan applications based on income, credit score, etc.

**Example with Iris Dataset (Classification):**

* The **Iris dataset** contains features like *petal length, petal width, sepal length, sepal width*.
* A Decision Tree can classify a flower into one of the three classes: *Iris-setosa, Iris-versicolor, Iris-virginica*.
* The tree may split on “Petal length” first, because it best separates the classes.

**(b) Regression Tasks**

* **House Price Prediction** → Predict housing prices based on location, number of rooms, etc.
* **Sales Forecasting** → Estimate future sales based on historical trends.
* **Agriculture** → Predict crop yield based on rainfall, soil quality, and temperature.

**Example with Boston Housing Dataset (Regression):**

* The dataset contains features like *number of rooms, crime rate, proximity to highway*.
* A Decision Tree Regressor can predict the **median value of owner-occupied homes (target variable)**.
* Example: A split may happen on “Number of rooms ≥ 6” → Higher predicted price, otherwise → Lower price.

**2. Main Advantages of Decision Trees**

1. **Easy to Understand and Visualize** → Works like human decision-making, interpretable even for non-technical users.
2. **Handles Both Numerical and Categorical Data** → Flexible for real-world problems.
3. **No Need for Feature Scaling/Normalization** → Unlike SVM or KNN, data preprocessing is minimal.
4. **Works for Classification & Regression** → One algorithm, multiple applications.
5. **Feature Selection Built-in** → Automatically selects important features during splitting.

**3. Main Limitations of Decision Trees**

1. **Overfitting** → Trees can become too deep and memorize data instead of generalizing.
2. **Instability** → Small changes in data may lead to very different trees.
3. **Bias Toward Dominant Features** → Features with more levels (e.g., many categories) may get chosen unfairly.
4. **Less Accurate Alone** → Usually improved with ensemble methods like **Random Forest** or **Gradient Boosted Trees**.
5. **Continuous Variables Handling** → Can create too many splits if not pruned properly.

**4. Summary**

* Decision Trees are applied in **classification** (Iris dataset, medical diagnosis, spam detection) and **regression** (Boston Housing, price prediction, forecasting).
* **Advantages:** Easy to interpret, versatile, little preprocessing.
* **Limitations:** Overfitting, instability, and sometimes lower standalone accuracy.

**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**

ANSWER

# Import necessary libraries

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# 1. Load the Iris Dataset

iris = load\_iris()

X = iris.data # Features (sepal length, sepal width, petal length, petal width)

y = iris.target # Target classes (Setosa, Versicolor, Virginica)

# 2. Split into training and testing sets (80% train, 20% test)

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

# 3. Train a Decision Tree Classifier using Gini criterion

clf = DecisionTreeClassifier(criterion="gini", random\_state=42)

clf.fit(X\_train, y\_train)

# 4. Predict on test data

y\_pred = clf.predict(X\_test)

# 5. Print model’s accuracy

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

print("Model Accuracy:", accuracy)

# 6. Print feature importances

print("Feature Importances:")

for feature, importance in zip(iris.feature\_names, clf.feature\_importances\_):

print(f"{feature}: {importance:.4f}")

Expected Output (example run)

java

Copy

Edit

Model Accuracy: 1.0

Feature Importances:

sepal length (cm): 0.0000

sepal width (cm): 0.0000

petal length (cm): 0.4444

petal width (cm): 0.5556

**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.?**

ANSWER

# Import required libraries

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# 1. Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# 2. Split 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)

# 3. Train a fully-grown Decision Tree (no max\_depth limit)

clf\_full = DecisionTreeClassifier(random\_state=42)

clf\_full.fit(X\_train, y\_train)

# 4. Train a Decision Tree with max\_depth = 3

clf\_pruned = DecisionTreeClassifier(max\_depth=3, random\_state=42)

clf\_pruned.fit(X\_train, y\_train)

# 5. Predictions

y\_pred\_full = clf\_full.predict(X\_test)

y\_pred\_pruned = clf\_pruned.predict(X\_test)

# 6. Accuracy Scores

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

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

# 7. Print Results

print("Accuracy of Fully-Grown Tree:", accuracy\_full)

print("Accuracy of Tree with max\_depth=3:", accuracy\_pruned)

**Expected Output (example run)**

Accuracy of Fully-Grown Tree: 1.0

Accuracy of Tree with max\_depth=3: 0.9667

**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 importance?**

Answer:

# Import required libraries

from sklearn.datasets import fetch\_california\_housing

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

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean\_squared\_error

# 1. Load the California Housing dataset

housing = fetch\_california\_housing()

X = housing.data

y = housing.target

# 2. Split into training and testing sets (80% train, 20% test)

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

# 3. Train a Decision Tree Regressor

regressor = DecisionTreeRegressor(random\_state=42)

regressor.fit(X\_train, y\_train)

# 4. Predict on test data

y\_pred = regressor.predict(X\_test)

# 5. Calculate Mean Squared Error (MSE)

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

print("Mean Squared Error (MSE):", mse)

# 6. Print feature importances

print("\nFeature Importances:")

for feature, importance in zip(housing.feature\_names, regressor.feature\_importances\_):

print(f"{feature}: {importance:.4f}")

Expected Output (example run)

vbnet

Copy

Edit

Mean Squared Error (MSE): 0.2569

Feature Importances:

MedInc: 0.6403

HouseAge: 0.0427

AveRooms: 0.0815

AveBedrms: 0.0078

Population: 0.0246

AveOccup: 0.0142

Latitude: 0.0934

Longitude: 0.0955

(Values may vary slightly due to randomness.)

Explanation

Dataset: California Housing dataset predicts median house value based on features like median income, house age, average rooms, latitude, longitude, etc.

Decision Tree Regressor: Fits the training data and makes continuous predictions.

MSE (Mean Squared Error): Measures prediction error → lower is better.

Feature Importances: Show which features (e.g., MedInc = Median Income) contribute most to predicting house prices.

**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**

ANSWER

# Import required libraries

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# 1. Load the Iris Dataset

iris = load\_iris()

X = iris.data

y = iris.target

# 2. Split 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)

# 3. Define Decision Tree Classifier

clf = DecisionTreeClassifier(random\_state=42)

# 4. Define parameter grid for tuning

param\_grid = {

"max\_depth": [2, 3, 4, 5, None],

"min\_samples\_split": [2, 3, 4, 5, 10] }

# 5. GridSearchCV for tuning hyperparameters

grid\_search = GridSearchCV(clf, param\_grid, cv=5, scoring="accuracy")

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

# 6. Get best parameters and best estimator

best\_params = grid\_search.best\_params\_

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

# 7. Evaluate on test set

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

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

8. Print Results

print("Best Parameters:", best\_params)

print("Model Accuracy with Best Parameters:", accuracy)

**Expected Output (example run)**

Best Parameters: {'max\_depth': 3, 'min\_samples\_split': 2}

Model Accuracy with Best Parameters: 1.0

**Explanation**

1. **GridSearchCV** systematically tests combinations of hyperparameters (max\_depth, min\_samples\_split).
2. It uses **cross-validation (cv=5)** to avoid overfitting while tuning.
3. The best parameters are selected based on highest accuracy.
4. Final model is evaluated on the test set.

**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**

**Scenario:**

You are a data scientist in a healthcare company. The goal is to predict whether a patient has a certain disease (**Yes/No**) using a dataset with **mixed data types** (numerical + categorical) and **some missing values**.

**Step 1: Handle Missing Values**

* **Numerical Features:**
  + Fill missing values with the **mean** or **median** (depending on distribution).
  + Example: If “Blood Pressure” has missing values, replace them with the median.
* **Categorical Features:**
  + Fill missing values with the **mode** (most frequent category).
  + Example: If “Smoking Status” is missing, fill with the most common value (e.g., “Non-Smoker”).
* **Advanced Option:** Use imputation methods (like **KNNImputer**) if missing values are large.

**Step 2: Encode Categorical Features**

* Since Decision Trees require numerical input:
  + Use **Label Encoding** if categories have natural order (e.g., *Stage I, Stage II, Stage III*).
  + Use **One-Hot Encoding** if categories are unordered (e.g., *Male, Female, Other*).

Example: “Smoking Status” → {Non-Smoker, Former Smoker, Current Smoker}  
→ One-Hot Encode into 3 binary columns.

**Step 3: Train a Decision Tree Model**

* Split dataset into **training (80%)** and **testing (20%)**.
* Train a **DecisionTreeClassifier** with default parameters.
* Criterion can be "gini" or "entropy".

Python Example:

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(criterion="gini", random\_state=42)

clf.fit(X\_train, y\_train)

**Step 4: Tune Hyperparameters**

Use **GridSearchCV** or **RandomizedSearchCV** to optimize:

* **max\_depth** → prevent overfitting.
* **min\_samples\_split** → minimum samples needed to split a node.
* **min\_samples\_leaf** → minimum samples in leaf node.
* **criterion** → Gini or Entropy.

Python Example:

param\_grid = {

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

"min\_samples\_split": [2, 5, 10],

"min\_samples\_leaf": [1, 2, 4],

"criterion": ["gini", "entropy"] }

GridSearch selects the best combination for maximum accuracy.

**Step 5: Evaluate Model Performance**

* **Accuracy Score** → % of correct predictions.
* **Precision & Recall** → Important in healthcare (recall ensures sick patients are not missed).
* **F1-Score** → Balances precision & recall.
* **Confusion Matrix** → Shows true positives, false negatives (very important in disease prediction).

Example:

* If accuracy = 92%, recall = 95% → model successfully detects most diseased patients.

**Step 6: Business Value in Real-World Setting**

* **Early Disease Detection:** Helps doctors identify at-risk patients quickly.
* **Cost Reduction:** Reduces unnecessary tests by focusing on high-risk patients.
* **Decision Support System:** Assists medical staff in diagnosis.
* **Improved Patient Outcomes:** Faster treatment for correctly predicted positive cases.
* **Scalability:** Can be applied across hospitals with large patient databases.

Example:  
If the model predicts with **95% recall**, it ensures that almost all patients with the disease are identified → potentially **saves lives** by enabling early treatment.