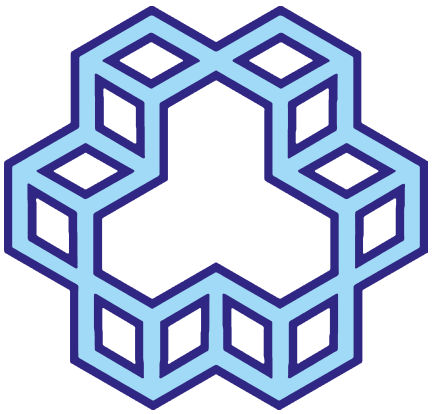

Students:
Ali Soltani_40119403

Professor: Dr. Aliyari

Course:
Fundamental of Intelligent Systems

Project Links:

Google Colab:

Question 5 and 6:

Question 12:

Github:

Date: 1404/10/5

Contents

1 Question 1 **3**
- 1.1 Part A: Decision Tree and Its Structure . . . 3
- 1.2 Part B: Margin Width (Components) . . . 3
- 1.3 Part C: A Real-World Application of Decision Trees . . . 4

2 Question 2 **6**
- 2.1 Part A: Overall Entropy Respect to Play . . . 6
- 2.2 Part B: Information Gain of the Outlook Feature . . . 6
- 2.3 Part C: ID3 Algorithm . . . 7

3 Question 3 **8**
- 3.1 Part A: Problems and Limitations of Decision Trees . . . 8
- 3.2 Part B: Advantages and Strengths of Decision Trees . . . 8

4 Question 4 **10**
- 4.1 Part A: Pre-Pruning & Post-Pruning . . . 10
- 4.2 Part B — Section 1: Conceptual Example . . . 10

5 Question 5 **11**
- 5.1 Part A: Dataset Description and Objective . . . 11
- 5.2 Part B: Data Preprocessing . . . 11
- 5.3 Part C: Model Training . . . 12

6 Question 6 **14**
- 6.1 Part B: Cost-Complexity Pruning . . . 14
  - 6.1.1 Evaluate accuracy . . . 15
- 6.2 Part C: Conclusion . . . 16

7 Question 7 **17**
- 7.1 Part A: Definition of Ensemble Learning and Why It Performs Better . . . 17
- 7.2 Part B: Difference Between Aggregation and Diversification . . . 17
- 7.3 Part C: Real-World Example — Bank Fraud Detection System . . . 18

8 Question 8: Bagging and Random Forest **19**
- 8.1 A) Bootstrap Sampling in Bagging . . . 19
- 8.2 B) Example of Three Bootstrap Samples . . . 19
- 8.3 C) Why Random Forest Outperforms a Single Decision Tree . . . 19

9 Question 9: Concept of Boosting vs. Bagging **21**
- 9.1 The Role of Sample Weights in the AdaBoost Algorithm . . . 21
- 9.2 Boosting and Noise Sensitivity . . . 22

10 Question 10 **24**
- 10.1 (a) What is Stacking and How Does It Use a Second-Level Model (Meta Learner)? . . . 24
- 10.2 (b) Why Must Cross-Validation Be Used for Training the Meta Learner? . . . 24
- 10.3 (c) Example of a Stacking System Architecture . . . 24

11 Question 11 **26**
11.1 (a) Four Advantages of Ensemble Learning 26
11.2 (b) Four Disadvantages of Ensemble Learning 26

12 Question 12 **27**
12.1 1. Random Forest Model (`rf_model`) 27
12.2 2. Gradient Boosting Model (`gb_model`) 27
12.3 Conclusion 27

Decision Tree Assignment Questions

1 Question 1

1.1 Part A: Decision Tree and Its Structure

A decision tree is a supervised learning model used for classification and regression. The main idea of a decision tree is to gradually split the data based on features, dividing the data into purer subsets, so that we can eventually make a decision (a class or numerical value) for each input.

A decision tree has a tree-like structure that includes:

- Root Node
- Internal Nodes
- Branch
- Leaf Nodes

Each path from the root to a leaf can be interpreted as an If–Then rule. A decision tree is a suitable structure for learning data because its decision-making process is transparent and interpretable, functions similarly to human decision-making logic, does not require complex preprocessing, and can effectively model nonlinear relationships between features and the output.

1.2 Part B: Margin Width (Components)

The role of decision tree components:

- **Internal Node**: An internal node is where a test or condition is performed on one of the data features. At this node, it is decided which path the data should follow based on the value of that feature.
- **Branch**: A branch represents the outcome of a condition at an internal node and indicates the data's movement path from one node to the next node or to a leaf.
- **Leaf Node**: A leaf is the final node in the tree that provides the model's ultimate decision; this decision can be a class (in classification problems) or a numerical value.

Component	**Main Role**	**Equivalent in Logic**
Internal Node	Testing a feature	Question (If)
Branch	Path Based on Test Result	Then
Leaf	Declaring the Final Result	Output

1.3 Part C: A Real-World Application of Decision Trees

Customer Credit Evaluation System for Bank Loan Approval

One of the classic and very important applications of decision trees is their use in bank credit evaluation systems to decide whether to grant loans. Suppose a bank wants to decide whether to grant a loan to a new applicant or not. Since an incorrect decision can lead to significant financial losses, the bank needs an intelligent system that can analyze applicant information and predict the risk of loan default.

Because of their Simplicity, Transparency, and High interpretability, decision trees are a very suitable option for this task.

Decision Tree Inputs (Features)

To train the model, the bank uses data from its previous customers—both those who repaid their loans on time and those who defaulted or had repayment issues. Some of the most important input features are shown in the table below:

Feature	Data Type	Description
Monthly income	Numerical	Applicant's income level
Age	Numerical	Applicant's age at loan application
Credit history	Categorical	Previous successful loans (Yes / No)
Marital status	Categorical	Single, married, divorced
Employment status	Categorical	Employee, entrepreneur, unemployed, etc.
Requested loan amount	Numerical	Amount of the requested loan
Repayment period	Numerical	Number of repayment months
Guarantor availability	Categorical	Presence of a valid guarantor (Yes / No)

The model's output is a binary classification that determines the applicant's risk category:

Output	Description
Low-risk	High probability of timely repayment → Loan approved
High-risk	High probability of repayment problems → Loan rejected

The Decision Tree Makes Decisions

By analyzing historical data, the decision tree learns which features are more important and what combinations of features best predict good or poor credit behavior. For example, the tree may derive rules such as:

- If monthly income is above 10 million tomans AND credit history is good → applicant is classified as low-risk.
- If monthly income is below 5 million tomans AND the applicant is unemployed → applicant is classified as high-risk.

Tree Structure in This Example

A hypothetical example of how data flows through this decision tree:

1. **Root Node**: Does the customer have a history of bounced checks?
    - If yes → Loan rejected
    - If no → proceed to the next step
2. **Intermediate node**: Is the monthly income greater than 15 million tomans?
    - If yes → Loan approved
    - If no → Proceed next step
3. **Intermediate node**: Is the employment status permanent?
    - If yes → Loan approved
    - If no → Loan rejected

2 Question 2

2.1 Part A: Overall Entropy Respect to Play

Entropy represents the level of impurity or uncertainty in the data. The entropy formula for a dataset with two classes is as follows:

$$H(S) = -p_{yes}\log_2(p_{yes}) - p_{no}\log_2(p_{no})$$

1. Total number of samples: 5
2. Number of Yes instances: $3 \rightarrow p_{yes} = \frac{3}{5} = 0.6$
3. Number of No instances: $2 \rightarrow p_{no} = \frac{2}{5} = 0.4$

So, we have:

$$\begin{aligned} H(S) &= -(0.6 \times \log_2(0.6) + 0.4 \times \log_2(0.4)) \\ H(S) &= -(0.6 \times -0.737 + 0.4 \times -1.322) \\ H(S) &= -(-0.442 - 0.529) = 0.971 \end{aligned}$$

2.2 Part B: Information Gain of the Outlook Feature

Information Gain shows how much a feature reduces entropy. Its formula is:

$$IG(S, Outlook) = H(S) - \sum_{v \in \{Sunny, Overcast, Rain\}} \frac{|S_v|}{|S|} H(S_v)$$

First, we calculate the entropy for each value of the Outlook feature.

1. **Sunny**: Appears in 2 rows. In both cases, the output is No. Since all labels are identical, entropy is zero.
$$H(Sunny) = 0$$
2. **Overcast**: Appears in 1 row (3). The output is Yes. This subset is also pure.
$$H(Overcast) = 0$$
3. **Rain**: Appears in 2 rows (4 and 5). The output in both cases is Yes. Again, the subset is pure.
$$H(Rain) = 0$$

Weighted average entropy for the Outlook feature is:

$$\text{Weighted\_Entropy} = \frac{2}{5}(0) + \frac{1}{5}(0) + \frac{2}{5}(0) = 0$$

Final Information Gain calculation is:

$$IG(S, Outlook) = H(S) - 0 = 0.971 - 0 = 0.971$$

The value 0.971 is the maximum possible value for this dataset. This means that the Outlook feature alone is able to perfectly (100%) separate the data, as all resulting branches are completely pure.

2.3 Part C: ID3 Algorithm

The ID3 algorithm constructs a decision tree by recursively selecting the feature with the highest Information Gain at each node. The objective is to reduce uncertainty in the dataset as quickly and effectively as possible.

Principle of Feature Selection

Entropy measures the amount of uncertainty in a dataset. A high entropy value indicates that the class labels are highly mixed, while an entropy of zero corresponds to a completely pure subset.

Information Gain quantifies how much the entropy decreases after splitting the dataset based on a given feature. It is defined as the difference between the entropy before the split and the weighted average entropy after the split.

Why ID3 Chooses the Feature with Maximum Information Gain

The preference for the feature with the highest Information Gain can be explained as follows:

1. **Maximum Uncertainty Reduction**: Selecting the feature with the highest Information Gain results in the largest decrease in entropy. This means the chosen feature provides the most informative split, reducing ambiguity in the data more than any other feature.

2. **Improved Class Purity**: A high Information Gain indicates that the resulting subsets are more homogeneous. In other words, after the split, most instances in each subset belong to the same class (either *Yes* or *No*), which simplifies further decisions.

3. **Efficient Tree Construction**: By consistently choosing the most informative feature, the ID3 algorithm builds a smaller and shallower tree. This avoids unnecessary splits on weak features and leads to a more interpretable and efficient decision tree.

In summary, ID3 uses Information Gain as its splitting criterion because it systematically minimizes entropy, maximizes class separation, and produces a compact decision tree with clear decision boundaries.

3 Question 3

3.1 Part A: Problems and Limitations of Decision Trees

1. **Overfitting**
   Overfitting is one of the most common issues in decision trees. When a tree grows too deep or has too many branches, it may start learning noise, outliers, or irrelevant patterns from the training data. This can result in very high accuracy on the training set but poor generalization on unseen test data. Techniques such as pruning, limiting tree depth, or requiring a minimum number of samples per leaf are often used to mitigate overfitting.

2. **Instability (High Variance)**
   Decision trees are highly sensitive to small changes in the training dataset. Even minor modifications, such as adding, removing, or slightly altering a few data points, can lead to a completely different tree structure. This high variance makes single decision trees less robust, and ensemble methods like Random Forests are commonly used to address this problem.

3. **Bias Toward High-Cardinality Features**
   Many splitting criteria, such as Information Gain, have a tendency to favor features with a large number of unique values (high cardinality), for example, unique ID numbers or codes. This can lead to very specific splits that do not generalize well to new data, resulting in misleading or overly complex trees.

4. **Difficulty in Modeling Oblique Decision Boundaries**
   Decision trees make axis-parallel splits, meaning that each decision rule only considers a single feature at a time. If the true decision boundary is diagonal or oblique in the feature space, the tree needs multiple step-wise splits to approximate it. This increases tree complexity, requires more nodes, and can reduce interpretability.

3.2 Part B: Advantages and Strengths of Decision Trees

1. **High Interpretability**
   Decision trees are considered white-box models. Their logic can be expressed as simple if–then rules, making the decision process easy to understand and interpret for non-technical users. This is especially valuable in domains where explainability is crucial, such as healthcare or finance.

2. **No Need for Feature Normalization**
   Decision trees are scale-invariant because they rely on comparisons between feature values rather than their absolute magnitudes. Features with very different scales (for example, income in dollars versus age in years) do not need normalization or standardization, simplifying preprocessing.

3. **Ability to Handle Mixed Data Types**
   Decision trees can naturally work with both numerical and categorical features within the same model. This flexibility allows for heterogeneous datasets without the need for extensive preprocessing or encoding.

4. **Automatic Feature Selection**
   During training, decision trees automatically identify and prioritize informative features. Features that provide the most useful splits appear closer to the root, while irrelevant or less important features are ignored or used lower in the tree. This implicit feature selection reduces the need for manual feature engineering.

4 Question 4

4.1 Part A: Pre-Pruning & Post-Pruning

Pre-pruning stops the growth of a decision tree before it is fully grown (e.g., max depth, min samples). **Post-pruning** builds a complete deep tree first, then removes parts that cause overfitting using validation data.

Feature	**Pre-pruning**	**Post-pruning**
When applied	During tree construction	After the tree is fully built
Initial tree depth	Limited and shallow	Usually deep and complete
Underfitting risk	High	Lower
Overfitting risk	Lower	Controlled through evaluation
Flexibility	Lower	Higher
Need for validation data	Usually less	Usually required

4.2 Part B — Section 1: Conceptual Example

Predicting Admission in the National Entrance Exam

Suppose we want to predict whether a student will be admitted based on features like GPA, study hours, and school type.

In **pre-pruning**, if "type of school" provides low Information Gain early on, the algorithm might stop, ignoring potential interactions with later features like "study hours," leading to underfitting.

In **post-pruning**, the tree grows fully. It might find that "type of school" combined with "study hours" at a lower level is highly predictive. If validation confirms this improves accuracy, the structure is kept; otherwise, it is removed. This usually results in a better balance between simplicity and accuracy compared to pre-pruning.

5 Question 5

5.1 Part A: Dataset Description and Objective

In this question, a real-world classification problem is addressed using the Titanic passenger dataset. The objective is to predict whether a passenger survived the disaster based on demographic and travel-related attributes.

The dataset contains the following important features:

- **Pclass**: Passenger class (1st, 2nd, or 3rd)
- **Sex**: Gender of the passenger
- **Age**: Age of the passenger
- **Fare**: Ticket fare
- **Embarked**: Port of embarkation

The target variable is `Survived`, which is a binary label indicating whether the passenger survived (1) or not (0).

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Figure 1: The first 5 lines of data

5.2 Part B: Data Preprocessing

Before training the Decision Tree model, several preprocessing steps were applied to prepare the dataset:

- **Handling Missing Values**:
  - Missing values in `Age` and `Fare` were filled using the median to reduce the effect of outliers.
  - Missing values in `Embarked` were filled using the mode.
- **Feature Selection**: Non-informative string-based features such as `Name`, `Ticket`, and `Cabin` were removed, as they do not provide direct predictive value for this task.
- **Categorical Encoding**: Categorical variables (`Sex` and `Embarked`) were converted into numerical format using One-Hot Encoding. The option `drop_first=True` was applied to avoid multicollinearity.

5.3 Part C: Model Training

After preprocessing, the dataset was divided into feature matrix X and target vector y. The data was split into training and test sets using an 80/20 ratio.

A Decision Tree Classifier was trained as the baseline model with a limited depth (`max_depth = 4`) to reduce overfitting and maintain interpretability.

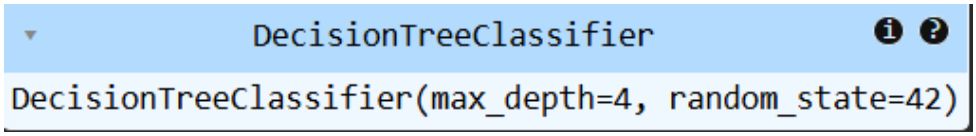

Figure 2: Create and train the model

Then, the model is trained, and its results are shown in the figure below.

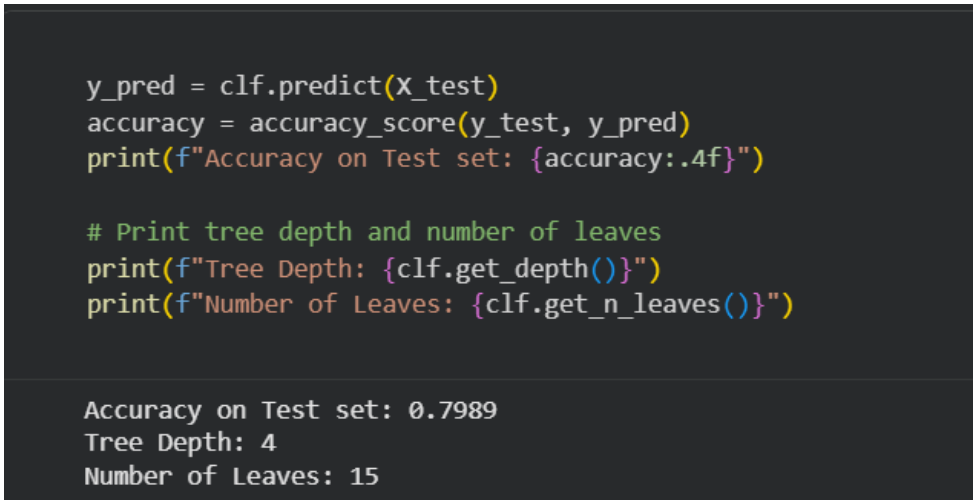

Figure 3: Evaluate the model

The figure illustrates the structure of the trained Decision Tree Classifier. The tree splits the data hierarchically based on feature values to maximize the purity of the resulting nodes.

- **Root Node:** The initial split occurs on the `Sex_male` feature (≤ 0.5), indicating that gender is the most significant predictor. The left branch corresponds to female passengers (True), while the right branch represents male passengers (False).
- **Color Coding:** The nodes are color-coded to represent the majority class: blue indicates `Survived`, while orange indicates `Not Survived`. The intensity of the color reflects the purity of the node (lower Gini impurity).
- **Sub-trees:**
  - The **left sub-tree** (Females) shows high survival rates, with further splits based on `Pclass` and `Fare`, suggesting that socio-economic status influenced survival among women.
  - The **right sub-tree** (Males) is predominantly classified as not surviving. However, a split on `Age` (≤ 6.5) highlights that young male children had different survival probabilities compared to adults.

- **Node Attributes:** Each node displays the decision criterion, the *Gini impurity* index, the total number of samples reaching that node, and the distribution of values between the two classes.

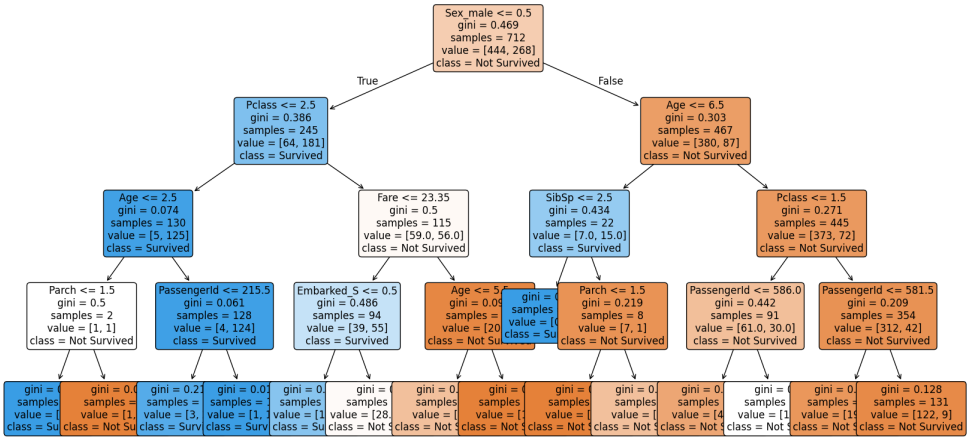

Figure 4: Decision Tree

6 Question 6

Part A: Effect of Tree Depth

To analyze the bias–variance tradeoff, multiple Decision Tree models were trained with different depth values.

- **Shallow Tree (Depth = 2)**: The model was too simple and resulted in underfitting, showing low training and test accuracy.
- **Moderate Depth (Depth = 3–5)**: These models achieved the best balance between bias and variance and produced the highest test accuracy.
- **Deep Tree (Depth = 10 or None)**: The model showed very high training accuracy but reduced test accuracy, indicating overfitting.

```
clf_full = DecisionTreeClassifier(max_depth=None, random_state=42)
clf_full.fit(X_train, y_train)


# Report accuracy
train_acc = clf_full.score(X_train, y_train)
test_acc = clf_full.score(X_test, y_test)
print(f"Full tree train accuracy: {train_acc:.4f}")
print(f"Full tree test accuracy: {test_acc:.4f}")


# Number of nodes and tree depth
print(f"Number of nodes: {clf_full.tree_.node_count}")
print(f"Tree depth: {clf_full.tree_.max_depth}")
```

```
Full tree train accuracy: 1.0000
Full tree test accuracy: 0.7207
Number of nodes: 289
Tree depth: 17
```

Figure 5: Full tree (max depth=None)

6.1 Part B: Cost-Complexity Pruning

To automatically control tree complexity, Cost-Complexity Pruning was applied. This method introduces a regularization parameter α that penalizes the number of leaf nodes.

As α increases:

- Tree complexity decreases
- Training accuracy decreases
- Test accuracy initially improves and then decreases due to underfitting

The optimal value of α was selected based on maximum test accuracy. The resulting pruned tree was significantly smaller than the full tree while maintaining strong generalization performance.

```python
for depth in [2, 3, 5, 10, None]:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X_train, y_train)
    print(f"\nTree with max_depth={depth}")
    print(f"Train accuracy: {clf.score(X_train, y_train):.4f}")
    print(f"Test accuracy: {clf.score(X_test, y_test):.4f}")
```

```
...
    Tree with max_depth=2
    Train accuracy: 0.8034
    Test accuracy: 0.7654

    Tree with max_depth=3
    Train accuracy: 0.8343
    Test accuracy: 0.7989

    Tree with max_depth=5
    Train accuracy: 0.8596
    Test accuracy: 0.7989

    Tree with max_depth=10
    Train accuracy: 0.9508
    Test accuracy: 0.7821

    Tree with max_depth=None
    Train accuracy: 1.0000
    Test accuracy: 0.7207
```

Figure 6: Part (b): Trees with different max depth values

6.1.1 Evaluate accuracy

The plot illustrates the relationship between the effective alpha parameter (`ccp_alpha`) and the model's accuracy, serving as a diagnostic tool for selecting the optimal tree complexity.

- **Axes Description:** The x-axis represents `ccp_alpha`, the regularization parameter that controls the aggressiveness of pruning. The y-axis denotes the classification accuracy.
- **Training Accuracy (Blue Line):** The training accuracy starts near 1.0 when $\alpha \approx 0$, corresponding to a fully grown, unpruned tree that fits the training data perfectly (likely overfitting). As α increases, the tree is pruned, causing the training accuracy to monotonically decrease as the model becomes simpler.
- **Testing Accuracy (Orange Line):** The testing accuracy initially improves as α increases from 0, indicating that pruning is successfully removing noise and reducing overfitting. The curve reaches a peak (around $\alpha \approx 0.003$), representing the optimal balance between bias and variance.
- **Conclusion:** Beyond the peak, the testing accuracy begins to drop, signaling that the tree has been over-pruned and is now underfitting the data. The optimal model is selected at the `ccp_alpha` value corresponding to the maximum testing accuracy.

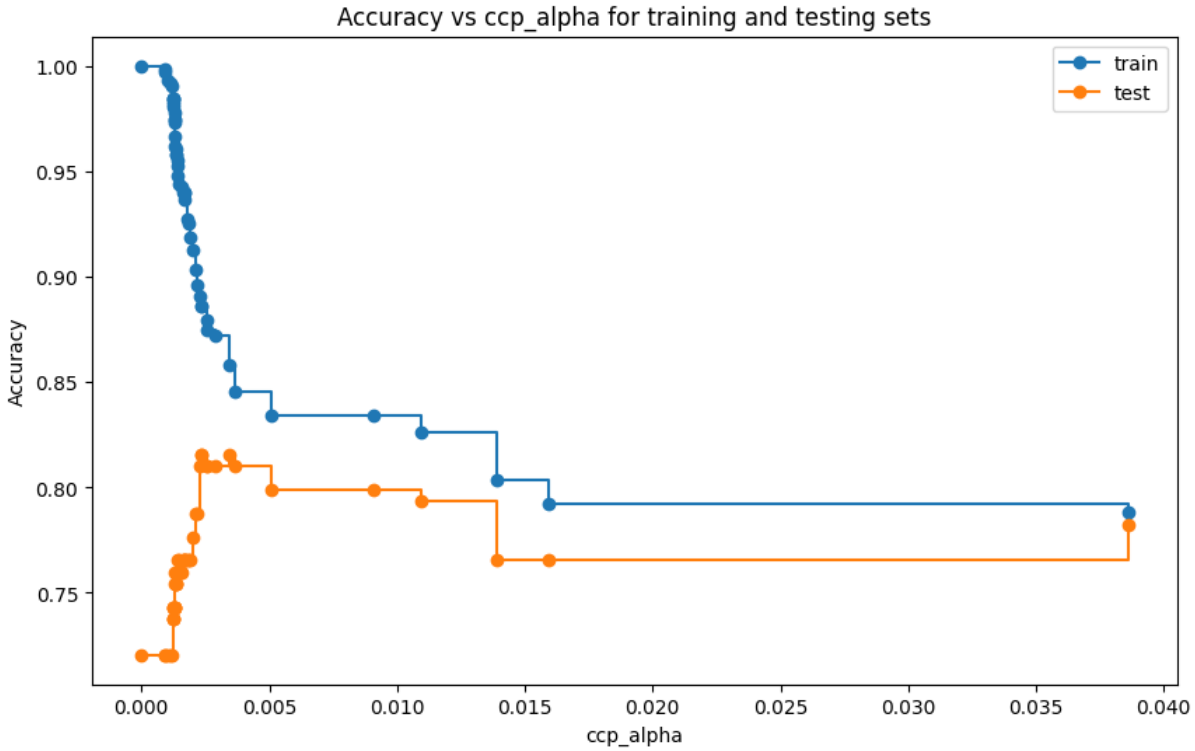

Figure 7: Evaluate accuracy vs alpha plot(train test accuracy)

6.2 Part C: Conclusion

This experiment demonstrated the importance of controlling model complexity in Decision Trees. Although fully grown trees can perfectly fit training data, they often fail to generalize.

By limiting tree depth and applying Cost-Complexity Pruning, overfitting was effectively reduced. The final pruned model achieved a strong balance between accuracy, simplicity, and interpretability, making it the most suitable solution for this classification task.

7 Question 7

7.1 Part A: Definition of Ensemble Learning and Why It Performs Better

Ensemble Learning is a machine learning paradigm in which multiple learning models, referred to as *base models* or *weak learners*, are trained to solve the same task. The individual predictions of these models are then combined to produce a single, final prediction that is generally more accurate and more robust than that of any individual model.

The fundamental motivation behind ensemble learning is that the limitations and errors of individual models can be compensated for by combining their predictions. This improvement in performance can be attributed to several key factors:

- **Variance Reduction:** Techniques such as *Bagging* (e.g., the Random Forest algorithm) train multiple models on different subsets of the training data. This reduces the model's sensitivity to variations in the training set and improves its generalization ability. By averaging predictions from models with largely uncorrelated errors, random noise and variance are significantly reduced.
- **Bias Reduction:** Methods such as *Boosting* (e.g., AdaBoost and Gradient Boosting) train models sequentially, where each subsequent model focuses on correcting the errors of its predecessor. This iterative refinement enables the ensemble to capture complex patterns in the data and reduces systematic bias.
- **Improved Prediction Accuracy:** Combining models of different types, such as neural networks, support vector machines, and decision trees, allows the ensemble to exploit the strengths of each learning algorithm. This approach, known as *Stacking*, often results in a more powerful and reliable predictive model.

7.2 Part B: Difference Between Aggregation and Diversification

Aggregation refers to the process of combining the predictions of individual models to produce a single output. The choice of aggregation method depends on the nature of the problem:

- **Regression Problems:** The final prediction is typically obtained by averaging or using a weighted average of the predictions from all models.
- **Classification Problems:**
  - *Majority Voting (Hard Voting):* The class label that receives the highest number of votes from the base models is selected as the final prediction.
  - *Averaging Probabilities (Soft Voting):* Each model outputs class probabilities, which are then averaged. The class with the highest average probability is chosen as the final output. This method often yields higher accuracy than hard voting.

Diversification is a fundamental principle in ensemble learning that ensures the base models are sufficiently different from one another. If all models are identical and commit

the same errors, aggregation will not improve performance. The goal of diversification is to minimize correlation among model errors.

Common techniques for introducing diversity include:

- Training models on different subsets of the data (e.g., Bagging).
- Using different subsets of features (e.g., Random Forest).
- Employing different learning algorithms (e.g., Decision Tree, KNN, and SVM).
- Varying hyperparameters within the same algorithm.

In summary, diversification addresses how to construct different models, while aggregation determines how their predictions are combined.

7.3 Part C: Real-World Example — Bank Fraud Detection System

A prominent real-world application of ensemble learning is fraud detection in banking systems.

Problem Objective: The goal is to classify each financial transaction into one of two categories: *Legitimate* or *Fraudulent*.

Model Inputs (Features): Each transaction is represented by a set of features, including:

- Transaction amount
- Time of transaction (hour and day)
- Transaction location (country and city)
- Merchant category
- Card-present status (Boolean)
- Historical cardholder behavior, such as:
  - Average transaction amount over the last 24 hours
  - Number of transactions in the last hour
  - Time elapsed since the previous transaction
  - Whether the transaction originates from a new device or IP address

Model Outputs: The ensemble model (e.g., Random Forest or Gradient Boosting) produces one of the following outputs:

- **Binary Classification Output:** A discrete label where 0 represents a legitimate transaction and 1 represents a fraudulent transaction. This output is commonly obtained using aggregation methods such as hard voting.
- **Probability Score:** A continuous value between 0 and 1 indicating the probability that a transaction is fraudulent. This score is typically produced through soft voting or probabilistic ensemble methods. For example, transactions with a score above 0.9 may be automatically blocked, while those with scores between 0.7 and 0.9 may trigger additional user verification.

8 Question 8: Bagging and Random Forest

Consider the following dataset:

x	y
2	0
3	0
8	1
9	1

8.1 A) Bootstrap Sampling in Bagging

Bootstrap sampling is a resampling technique used in the Bagging (Bootstrap Aggregating) method. Given an original training dataset of size n, multiple new datasets, called *bootstrap samples*, are generated by randomly sampling n observations **with replacement** from the original dataset.

Because sampling is performed with replacement, some observations may appear multiple times in a bootstrap sample, while others may not appear at all. Each bootstrap sample is used to train an independent base learner, typically a decision tree. The final prediction is obtained by aggregating the predictions of all models, using majority voting for classification or averaging for regression. This process reduces model variance and improves stability.

8.2 B) Example of Three Bootstrap Samples

The original dataset contains the following four observations:

$$(2,0),\ (3,0),\ (8,1),\ (9,1)$$

Each bootstrap sample consists of four observations sampled with replacement. One possible set of three bootstrap samples is:

- **Bootstrap B1**: $(2,0),(3,0),(3,0),(9,1)$
- **Bootstrap B2**: $(8,1),(9,1),(9,1),(2,0)$
- **Bootstrap B3**: $(3,0),(8,1),(8,1),(2,0)$

Due to sampling with replacement, duplicate observations may occur, and some original samples may be omitted in a given bootstrap dataset.

8.3 C) Why Random Forest Outperforms a Single Decision Tree

Random Forest usually performs better than a single decision tree for the following reasons:

- **Variance Reduction**: A single decision tree is highly sensitive to variations in the training data. Random Forest reduces variance by averaging predictions from multiple trees.

- **Feature Randomness**: At each split, Random Forest considers only a random subset of features, which reduces correlation among trees and increases model diversity.
- **Improved Generalization**: The ensemble of decorrelated trees reduces overfitting and improves predictive performance on unseen data.

As a result, Random Forest models are more robust and accurate compared to individual decision trees.

9 Question 9: Concept of Boosting vs. Bagging

Boosting is a powerful ensemble learning strategy designed to synthesize a single robust predictor (a strong learner) from a collection of simpler, less accurate models (weak learners).

The core philosophy of Boosting is sequential learning. Unlike methods that train models in isolation, Boosting trains them one after another. Each subsequent model is explicitly tasked with rectifying the errors made by its predecessors. Essentially, if the first model misclassifies certain data points, the second model is constructed to focus specifically on those "difficult" instances.

Table 1: Comparison of Bagging and Boosting

Feature	**Bagging**	**Boosting**
Construction Method	Models are built independently and in parallel.	Models are built sequentially, with each dependent on the previous one.
Primary Objective	Variance reduction.	Reduction of both Bias and Variance.
Data Weighting	All samples have an equal probability of selection.	Hard-to-classify samples are assigned higher weights.
Result Combination	Simple averaging or majority voting.	Weighted averaging based on model performance.

9.1 The Role of Sample Weights in the AdaBoost Algorithm

In the AdaBoost (Adaptive Boosting) algorithm, sample weights act as the central mechanism driving the learning process.

Concept: Initially, every data point is treated equally. However, after each training iteration, the algorithm evaluates its performance. Before training the next model, AdaBoost increases the importance (weight) of samples that were misclassified. This effectively forces the new model to "zoom in" on these difficult cases, prioritizing them over data points that are already easy to classify.

The weights essentially signal to the algorithm: *"You haven't mastered these specific points yet; make them your priority in the next step."*

Initialization

At the start, we assign a uniform weight to every training example in the dataset of size N.

$$w_i^{(1)} = \frac{1}{N} \quad \text{for } i = 1, \ldots, N$$

Computation of Model Error

In each iteration t, a weak learner is trained, and we calculate its weighted error rate. This metric accounts for the importance of each error:

$$\epsilon_t = \sum_{i=1}^{N} w_i^{(t)} \cdot \mathbb{I}(h_t(x_i) \neq y_i)$$

Here, $\mathbb{I}$ is an indicator function that returns 1 if the prediction is wrong and 0 otherwise. Because of the term w_i, mistakes on "important" (highly weighted) samples result in a much higher penalty for the model.

Computation of Model Weight
Next, we determine the authority of this specific model in the final decision. We calculate α_t, which represents the model's voting power:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

- If the error ϵ_t is small, α_t becomes large and positive (the model is deemed reliable).
- If the error is 50% (random guessing), α_t drops to zero (the model is ignored).

Updating Sample Weights
This step is critical for learning from mistakes. We update the weights for the next iteration $t + 1$:

$$w_i^{(t+1)} = w_i^{(t)} \cdot \exp\left(-\alpha_t \cdot y_i \cdot h_t(x_i)\right)$$

- **Correct Predictions** ($y_i = h_t(x_i)$): The exponent becomes negative, reducing the weight w_i. The algorithm pays less attention to this easy instance in the future.
- **Incorrect Predictions** ($y_i \neq h_t(x_i)$): The exponent becomes positive, increasing the weight w_i. This data point becomes "louder," forcing the next model to focus on learning it correctly.

Final Prediction
The ultimate classification is a weighted vote across all T trained models.

$$H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

Here, α_t ensures that accurate models have a stronger influence on the final output than weaker ones.

Table 2: Types of Weights in AdaBoost

Weight Type	**Symbol**	**Primary Function**
Sample Weights	w_i	Forces the subsequent model to focus on difficult samples and correct previous errors.
Model Weights	α_t	Determines the influence or "voting power" of a specific model in the final ensemble based on its accuracy.

9.2 Boosting and Noise Sensitivity

Boosting is capable of building highly accurate models because it iteratively attacks errors, reducing both bias and overall error. This step-by-step focus allows it to uncover complex patterns that a single model would miss.

However, this strength is also a vulnerability. Boosting is notably sensitive to noisy data and outliers. If the training data contains noise (e.g., incorrect labels), the algorithm will treat these as "hard" samples and aggressively increase their weights. Consequently, the model may waste capacity trying to learn the noise, leading to overfitting and poor performance on new, unseen data.

10 Question 10

10.1 (a) What is Stacking and How Does It Use a Second-Level Model (Meta Learner)?

Stacking, also known as *Stacked Generalization*, is an ensemble machine learning technique that aims to improve predictive performance by combining the outputs of multiple diverse models.

The stacking approach is organized into two training levels:

- **Level 0 (Base Learners):** At this level, several different models (such as Decision Trees, Support Vector Machines, and Neural Networks) are trained on the original dataset. Each base learner independently generates predictions for the same input data.
- **Level 1 (Meta Learner):** The predictions produced by the base learners are collected and used as new input features for a second-level model, known as the Meta Learner.

Role of the Meta Learner: Unlike simple ensemble methods such as majority voting or averaging, the Meta Learner learns how to optimally combine the predictions of the base models. It identifies which models are more reliable in different situations and effectively corrects their individual biases, resulting in improved overall performance.

10.2 (b) Why Must Cross-Validation Be Used for Training the Meta Learner?

Cross-Validation is essential in stacking to prevent **data leakage** and **overfitting**.

The Problem: If base models are trained on the training dataset and their predictions on the same dataset are directly used as inputs for the Meta Learner, those predictions will be overly optimistic. Since the base models have already seen the true labels, the Meta Learner may simply learn to trust the model that memorizes the training data best, rather than learning generalizable patterns.

The Solution: By applying K-Fold Cross-Validation, *out-of-fold* predictions are generated. In this process, each data point is predicted by a base model that was not trained on that specific data point. As a result, the predictions used to train the Meta Learner accurately reflect model behavior on unseen data, leading to a more robust and generalizable ensemble.

10.3 (c) Example of a Stacking System Architecture

A common two-layer stacking architecture for a binary classification problem is described below.

Input: A dataset containing customer information used to predict customer churn (Yes/No).

Level 0 (Base Models):

- Model A: Random Forest
- Model B: Support Vector Machine (SVM)

- Model C: K-Nearest Neighbors (KNN)

Intermediate Prediction Stage: For a specific customer, each base model predicts the probability of churn:

$$\text{Random Forest} \rightarrow 0.8$$
$$\text{SVM} \rightarrow 0.7$$
$$\text{KNN} \rightarrow 0.9$$

New Feature Vector:

$$[0.8,\ 0.7,\ 0.9]$$

Level 1 (Meta Learner):

- **Algorithm:** Logistic Regression

The Meta Learner takes the vector $[0.8,\ 0.7,\ 0.9]$ as input and combines the base model predictions. By assigning appropriate weights to each base learner, it produces a final and more accurate churn probability, for example:

$$0.85$$

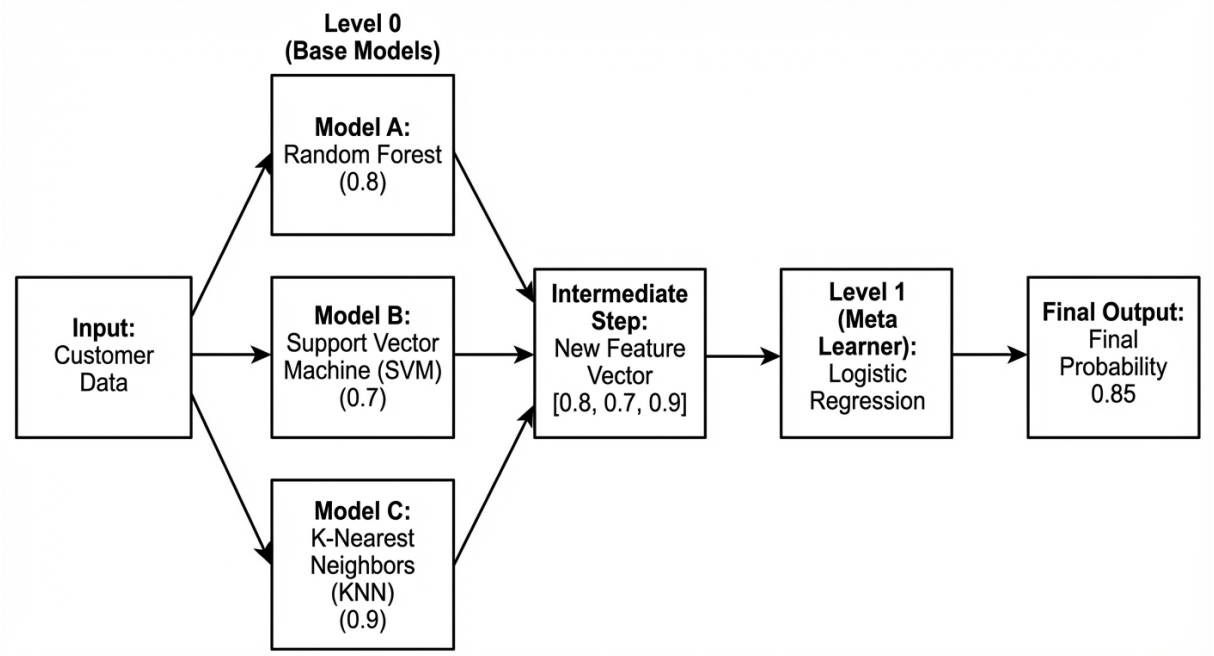

Figure 8: Figure 1: 2-Layer Stacking Ensemble Architecture for Churn Prediction.

This diagram illustrates the data flow in a Stacking system. Input data is first processed by three diverse Level 0 Base Learners (Random Forest, SVM, and KNN), which generate individual probability predictions. These predictions (0.8, 0.7, 0.9) are aggregated into a new feature vector and passed to a Level 1 Meta Learner (Logistic Regression), which synthesizes them to produce a final, more accurate prediction (0.85).

11 Question 11

11.1 (a) Four Advantages of Ensemble Learning

Ensemble learning combines multiple base models to produce a single, improved predictive model. The main advantages of ensemble learning are as follows:

1. **Higher Accuracy**
   By aggregating the predictions of several models, ensemble methods typically achieve higher accuracy and lower error rates compared to any individual base model.

2. **Reduced Variance and Overfitting**
   Methods such as Bagging (e.g., Random Forests) reduce variance by averaging the predictions of multiple learners. This process prevents the model from memorizing noise in the training data and improves generalization.

3. **Robustness**
   Ensemble models are more stable and less sensitive to outliers. If one base learner makes an incorrect prediction for a particular data point, other models can compensate for this error, leading to more reliable overall predictions.

4. **Ability to Handle Complex Relationships**
   Ensembles are capable of modeling complex and non-linear patterns in data more effectively than simple single models, such as logistic regression or shallow decision trees.

11.2 (b) Four Disadvantages of Ensemble Learning

Despite their strong performance, ensemble learning methods also have several limitations:

1. **Loss of Interpretability (Black-Box Nature)**
   While individual models such as single decision trees are easy to interpret, ensembles consisting of many models are difficult to explain. Understanding the exact reasoning behind a specific prediction becomes challenging.

2. **High Computational Cost During Training**
   Training an ensemble requires fitting multiple models, which significantly increases computational time and resource usage (CPU/GPU) compared to training a single model.

3. **Slower Inference Speed**
   During prediction, each input must be processed by all base models in the ensemble before a final output is produced. This can limit the use of ensemble methods in real-time or low-latency applications.

4. **Storage and Deployment Complexity**
   Ensembles require more storage space, as multiple models must be saved instead of one. Additionally, deploying and maintaining such systems in production environments is more complex.

12 Question 12

Based on the accuracy metrics provided in the code outputs, here is the diagnosis regarding Underfitting and Overfitting for each model:

12.1 1. Random Forest Model (`rf_model`)

- **Training Accuracy:** 0.9789 ($\approx 97.9\%$)
- **Testing Accuracy:** 0.8045 ($\approx 80.4\%$)
- **Diagnosis: Severe Overfitting**.
- **Reasoning:** There is a substantial gap ($\approx 17.5\%$) between the training and testing accuracy. The model has achieved near-perfect performance on the training data, indicating it has likely "memorized" the noise and specific patterns of the training set rather than learning generalizable features. This is a classic sign of *High Variance*.

12.2 2. Gradient Boosting Model (`gb_model`)

- **Training Accuracy:** 0.9045 ($\approx 90.5\%$)
- **Testing Accuracy:** 0.8045 ($\approx 80.4\%$)
- **Diagnosis: Moderate Overfitting**.
- **Reasoning:** While the gap between training and testing ($\approx 10\%$) is smaller compared to the Random Forest model, it still indicates overfitting. The model performs significantly better on seen data than on unseen data.

12.3 Conclusion

Neither model is suffering from Underfitting.

- If the models were underfitting, the **Training Accuracy** would be low (e.g., $< 60 - 70\%$), indicating an inability to capture the underlying trend of the data.
- Since both models have high training scores ($> 90\%$), the issue is strictly related to generalization (Overfitting).