K.N. Toosi University of Technology

Students: Mostafa Latifian – 40122193

Parsa Alaviniko – 40120993

Professor: Dr. Mahdi Aliyari-Shoorehdeli

Course: Fundamental of Intelligent Systems

MP – 2

Question 5 & 6

Question 12

Google Drive Folder

Table of Content

Question 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.

Part B: Margin Width

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 future	Question (If)
Branch	Path Based on Test Result	Then
Leaf	Declaring the Final Result	Output

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 banks 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:

- Simplify
- Transparency
- 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 below table.

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, then the applicant is classified as low-risk.
- If monthly income is below 5 million tomans and the applicant is unemployed, then the applicant is classified as high-risk.

We can also show the Decision Tree like following method.

1. Tree Inputs (Features)

   These are the attributes that are evaluated at the internal nodes of the tree.

   - Monthly income level
   - Employment status
   - Current debt level
2. Tree Outputs (Labels)

   These are the outcomes that appear at the leaf nodes of the tree.

   - Class A: Loan approved
   - Class B: Loan application rejected
3. Tree Structure in This Example

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

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

The structure of a decision tree is:

- Easy to understand
- Clearly shows the decision-making logic
- Allows bank managers to review, analyze, and even modify the system’s decisions

For these reasons, decision trees are among the most popular tools in financial and credit decision-making problems.

Question 2

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 → $p_{yes} = \frac{3}{5} = 0.6$
3. Number of No instances: 2 → $p_{no} = \frac{2}{5} = 0.4$

So, we have:

$$H(S) = -\left(0.6 \times log_2(0.6) + 0.4 \times log_2(0.4)\right)$$

$$H(S) = -(0.6 \times -0.737 + 0.4 \times -1.322)$$

$$H(S) = -(-0.442 - 0.529) = 0.971$$

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:

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

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.

Part C: ID3 Algorithm

In the ID3 algorithm, the main goal is to build a tree that reaches the most accurate decision with the fewest number of questions (nodes). Selecting the feature with the highest Information Gain is the core driving force behind this strategy.

1. From the Perspective of Uncertainty Reduction

   Entropy is a measure of disorder or uncertainty in a dataset. When entropy is high, it means the data are highly mixed, and it is unclear which class each instance belongs to.

   The role of Information Gain is, this metric directly calculates the difference between entropy before the split and the total entropy after the split.

   When we select the feature with the highest IG, we are effectively asking the question that eliminates the greatest amount of uncertainty. This causes uncertainty to decrease more rapidly toward zero at each step. By doing so, the ID3 algorithm attempts to reach certainty as quickly as possible.

2. From the Perspective of Improving Class Separation

   The ultimate goal of any classification model is to separate classes from one another.

   A feature with high IG splits the data in such a way that, in each resulting subset, most instances belong to a single class.

   The higher the Information Gain of a feature, the greater its discriminative power. Selecting such a feature ensures that, in the lower levels of the tree, we deal with groups whose outcomes are clear, where almost all instances belong to either the Yes class or the No class.

The reason ID3 consistently selects the feature with the highest Information Gain can be summarized in three key points.

- Occam’s Razor Principle: By choosing features with high IG, the tree naturally becomes shorter and simpler, because each split makes a large step toward complete separation.
- Efficiency: If weak features with low IG are selected, the tree grows deeper unnecessarily, and instead of learning general patterns, the model becomes entangled in insignificant details
- Impurity Reduction: Maximum IG means minimum impurity in the child nodes. In other words, at each step, the model selects the most intelligent possible question for separating the classes.

Question 3

Part A: Four Problems and Limitations of Decision Trees

1. Overfitting

   This is the most common problem with decision trees. If the tree becomes too deep, it starts learning noise and irrelevant details from the training data. In this case, the model achieves 100% accuracy on the training data but performs very poorly on new (test) data.

2. Instability

   Decision trees are highly sensitive to small changes in the training data. Adding or removing just a few data points can completely change the tree’s structure, including the root node.

3. Bias Toward High-Cardinality Features

   Criteria such as Information Gain tend to favor features with many distinct values. These features create very fine-grained splits that have no real predictive value and only reduce impurity superficially.

4. Difficulty with Oblique Decision Boundaries

   Decision trees split data using vertical or horizontal lines (axis-parallel splits). If the true boundary between classes is diagonal or oblique in the feature space, the tree must approximate it using many step-like splits, leading to unnecessary model complexity.

Part B: Four Advantages and Strengths of Decision Trees

1. High Interpretability

   Decision trees are white-box models. Unlike complex models such as neural networks, their decision-making logic can be expressed as a set of if–then rules that are easy to understand for managers and non-technical users.

2. No Need for Normalization or Standardization

   Many algorithms (such as SVM or KNN) are sensitive to feature scales. If one feature ranges from 0 to 1 and another from 0 to 1,000,000, the model’s performance can degrade. Decision trees rely on value comparisons and are scale-invariant, requiring minimal data preprocessing.

3. Handling Mixed Data Types

   Decision trees can naturally handle both numerical and categorical features at the same time, without requiring all data to be converted into a specific format.

4. Automatic Feature Selection

   During training, decision trees automatically discard irrelevant features. Features appearing near the top of the tree are the most important predictors, while features that never appear have no impact on the output. This property helps us better understand the underlying data.

Decision trees are excellent for initial modeling and data understanding. However, to address issues such as instability and overfitting, real-world projects often combine them into more advanced models like Random Forest or XGBoost.

Question 4

Part A: Pre-Pruning & Post-Pruning

Pre-pruning is a technique in which the growth of a decision tree is stopped before the tree is fully grown. In this approach, the algorithm evaluates during tree construction whether continuing the split is beneficial or not. If predefined conditions are not satisfied, the tree growth is halted and the node is treated as a leaf.

Common stopping criteria include reaching a maximum tree depth, having a minimum number of samples per node, or when the Information Gain falls below a specified threshold.

In contrast, post-pruning first builds a complete and deep tree and then, in a separate phase, removes parts of the tree that cause overfitting. In this method, subtrees that do not positively impact the model's performance on validation data are replaced with a single leaf node.

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

Part B | Section 1: Conceptual Example

Predicting Admission in the National Entrance Exam

Suppose we want to use a decision tree to predict whether a student will be admitted to the national entrance exam. The input features include high school GPA, daily study hours, type of school, participation in exam-preparation classes, and rankings in mock exams.

In the early stages of tree construction, the feature "type of school" may cause only a small reduction in entropy at a particular node. In the pre-pruning approach, due to low Information Gain or an insufficient number of samples at that node, the algorithm stops the tree growth and converts the node into a leaf. As a result, a simpler model is obtained that no longer allows the interaction between "type of school" and more detailed features such as "study hours" or "mock exam rankings" to be examined, potentially leading to underfitting.

In contrast, in the post-pruning approach, the tree is first grown fully, and even these seemingly unimportant splits are created. Then, using validation data, it is evaluated whether these subtrees actually contribute to improving admission prediction. If some splits are found to merely model noise in the training data, they are removed; however, if the combination of multiple features at lower levels leads to more accurate predictions on unseen data, that part of the tree is retained.

This example illustrates that post-pruning, by evaluating the overall performance of subtrees, usually results in a model with a better balance between simplicity and accuracy, and consequently higher generalization ability, compared to pre-pruning.

Part B | Section 2: Conceptual Example Graphical View

we want to examine the difference between two important approaches for controlling the complexity of a decision tree:

- Pre-pruning
- Post-pruning

We begin with pre-pruning, in this approach, we define a strict stopping criterion in advance.

Suppose our stopping rule is:

$$If\ the\ Information\ Gain\ of\ \ a\ split\ is\ less\ than\ 0.1, stop\ growing\ the\ tree$$

Tree Construction Process

1. The algorithm starts building the tree using the strongest feature.
   In this example, the best feature is:
   - Mock exam ranking
2. After the first split, in one of the branches, the data are still not completely pure.
3. Now it is time to evaluate the next feature:
   - Type of school
4. The algorithm calculates the Information Gain of this split and reaches the following result:
   - Information Gain = 0.08

   The reason for the low IG is that:

   - There are admitted students from public schools, and
   - There are rejected students from private schools → meaning this feature does not have strong discriminative power.

5. Since:

$$0.08 < 0.1$$

the stopping condition is triggered.

6. The algorithm does not allow further growth of this branch and converts the current node into a leaf node.

The graphical illustration of pre-pruning in the university entrance exam example can be seen in Figure 4.1.

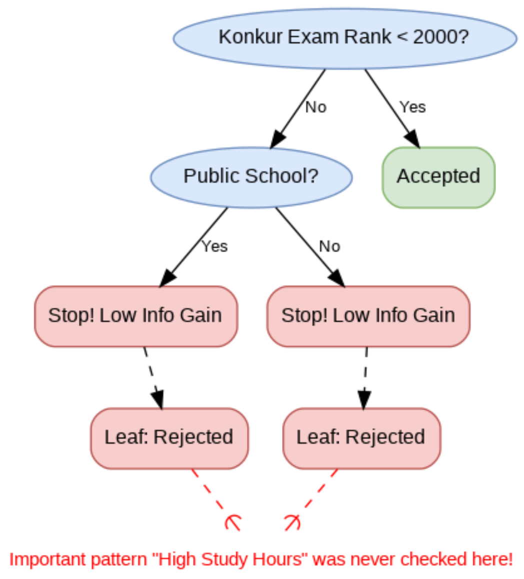


Fig. 4.1. Pre-Pruning graphical example

Due to early stopping, the model misses an important pattern: students from public schools who study many hours have a high chance of being admitted. The model suffers from underfitting and predicts “rejected” for all public-school students with a medium ranking.

Post-pruning Scenario, in this case, we allow the decision tree to grow fully, and then remove inefficient branches.

Construction and Pruning Process

1. The tree grows completely without any initial constraints.
2. A split based on “type of school” is performed, even if its initial Information Gain is low.
3. In the next step, within the “public school” branch, the tree splits based on “study hours”.

   Here, the algorithm discovers a very strong pattern: $if\ study > 6$, almost all students are admitted.

4. After the full tree is built, the algorithm evaluates the branches using validation data.
5. The branch that captures the pattern performs very well on the validation set, so it is retained.
6. Other branches that are caused by noise are removed.

This illustrates how post-pruning can preserve meaningful patterns while eliminating noise-driven complexity.

The first stage, which involves the full growth of the decision tree, is shown in Figure 4.2.

The second stage, which involves the rest growth of the decision tree, is shown in Figure 4.3.

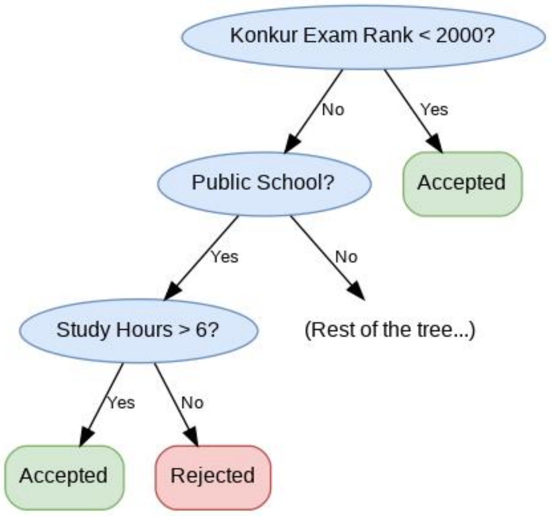


Fig. 4.2. Post-Pruning first stage

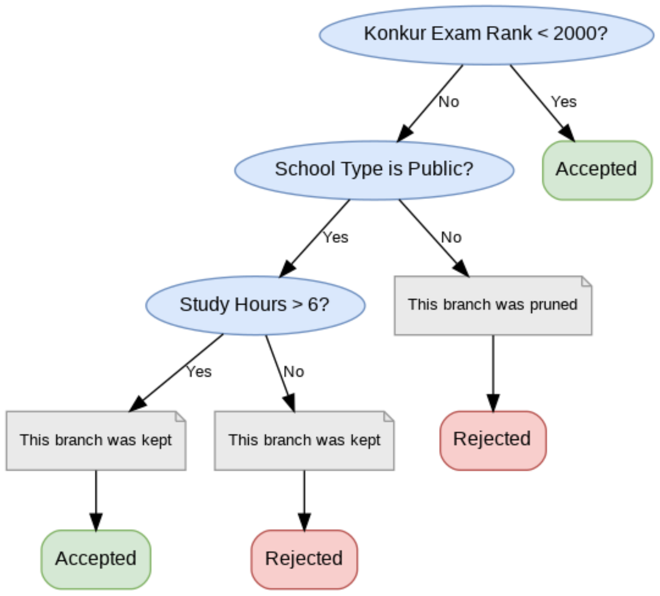


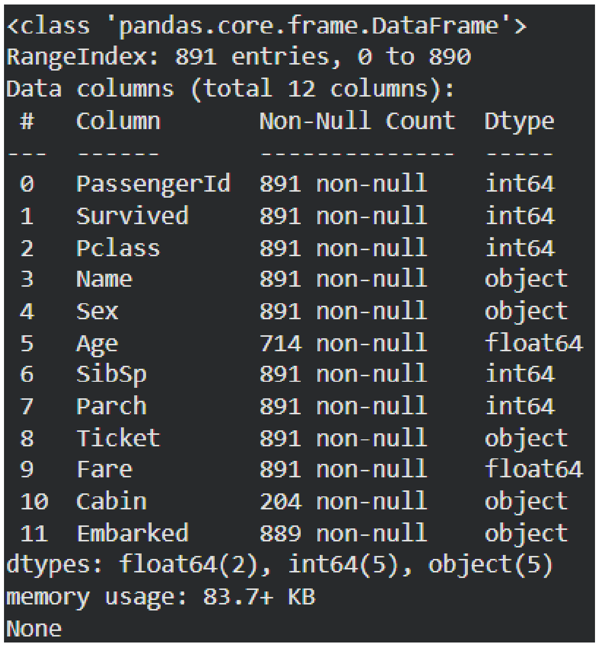
Fig 4.3. Post-Pruning second stage

The final model has learned an important and complex pattern while avoiding unnecessary complexity. As a result, it has better generalization ability and will produce more accurate predictions on new data.

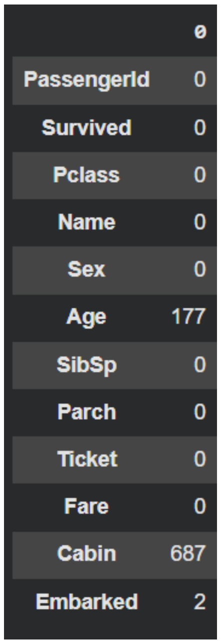
Question 5

5.1. Load and Pre-Process Data

In this project, the Titanic dataset from Kaggle was imported as the initial input for building a Decision Tree classification model. After loading the CSV file into Python, an exploratory review confirmed that the dataset contained 891 instances and 12 attributes. Several columns had missing values, "Age", "Embarked", and "Cabin". Additionally, non-informative identifiers such as "Name", "Ticket", and "Cabin" were not considered useful for model training and were therefore removed.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
```



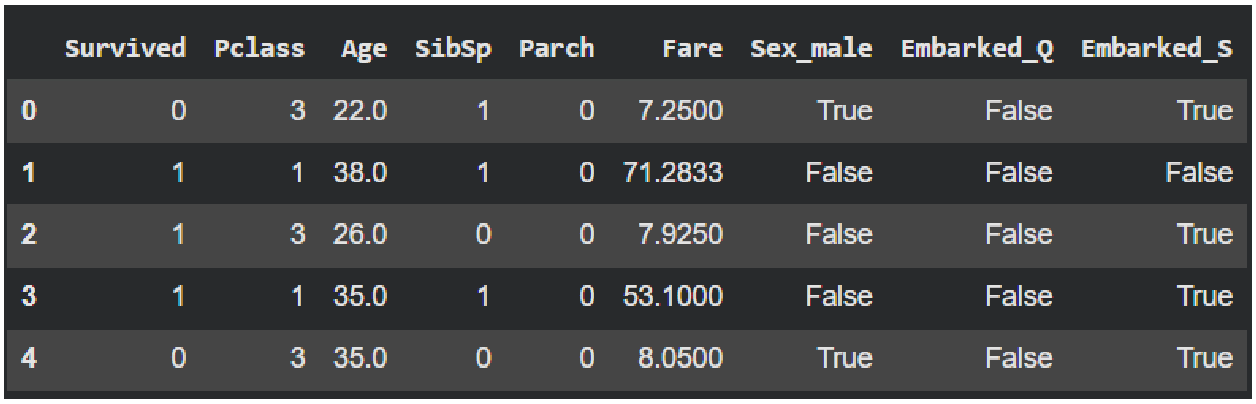
	0
PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

To handle missing values, numerical and categorical attributes were treated separately. The "Age" column was imputed using the median strategy, which is robust against outliers and appropriate for skewed distributions. For categorical variables, such as "Embarked", the most frequent category (mode) was applied to preserve the most common boarding port. After the imputation step, no missing values remained in the dataset.

```
imputer_num = SimpleImputer(strategy="median")
data['Age'] = imputer_num.fit_transform(pd.DataFrame(data['Age'])).ravel()

imputer_cat = SimpleImputer(strategy="most_frequent")
data['Embarked']                                                        =
imputer_cat.fit_transform(pd.DataFrame(data['Embarked'])).ravel()
```

Next, the non-numerical attributes ("Sex" and "Embarked") were transformed into numerical format through one-hot encoding. To avoid multicollinearity, we removed one level from each encoded variable. As a result, the dataset was converted entirely into a numerical feature space suitable for model training. The figure 5.1, show five first rows of pre-processed dataset.



	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_male	Embarked_Q	Embarked_S
0	0	3	22.0	1	0	7.2500	True	False	True
1	1	1	38.0	1	0	71.2833	False	False	False
2	1	3	26.0	0	0	7.9250	False	False	True
3	1	1	35.0	1	0	53.1000	False	False	True
4	0	3	35.0	0	0	8.0500	True	False	True

Fig. 5.1. Five first rows of pre-processed dataset

Finally, the processed dataset was separated into features (X) and labels (y), where the target variable was the "Survived" column. The data was split into training and testing partitions using an 80/20 ratio with stratification applied over the target variable to maintain the original class distribution. The final feature matrix contained the most relevant predictors, including "Pclass", "Age", "SibSp", "Parch", "Fare", and the encoded categorical attributes.

5.2. Training a Shallow Decision Tree with max_depth = 4

In the second stage of this project, a shallow Decision Tree classifier was trained with a maximum depth of 4 in order to control model complexity and reduce the risk of overfitting. The model was fitted using the preprocessed training data, and predictions were generated for both the training and testing partitions. The primary objective of this step was to evaluate how a constrained tree structure performs in terms of predictive accuracy and model interpretability.

The model configuration was defined as:

```
tree_4 = DecisionTreeClassifier(max_depth=4, random_state=93)
tree_4.fit(X_train, y_train)
```

After training, the following evaluation metrics were obtained:

```
(max_depth=4)

Train Accuracy: 0.8300561797752809

Test  Accuracy: 0.8491620111731844

Node count: 27

Max depth : 4
```

These results indicate that the shallow tree maintains a good generalization capability, as the test accuracy slightly exceeds the training accuracy. The limited depth prevents the model from memorizing the training samples, resulting in a balanced bias–variance behavior. The compact structure (27 nodes) also improves readability, which is beneficial for explainability and decision analysis.

Figure 5.1 illustrates the structure of the trained decision tree along with the splitting rules, class predictions, and Gini impurity at each node. The root node splits based on the feature “Sex_male”, confirming that gender is the strongest discriminator in survivor prediction, which aligns with well-documented domain knowledge regarding Titanic’s evacuation priority (“women and children first”). Additional branches rely on variables such as “Pclass”, “Age”, “Fare”, and “SibSp”, indicating that social status, age group, and family size also significantly influence survival likelihood.

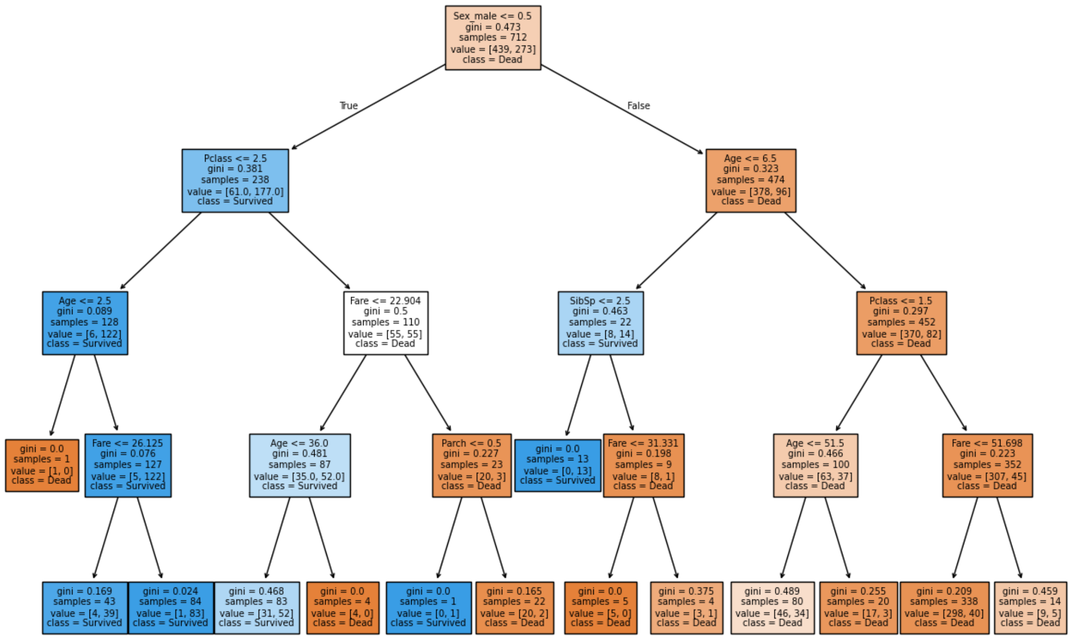

Sex_male <= 0.5 gini = 0.473 samples = 712 value = [439, 273] class = Dead
True
False
Pclass <= 2.5 gini = 0.381 samples = 238 value = [61.0, 177.0] class = Survived
Age <= 6.5 gini = 0.323 samples = 474 value = [378, 96] class = Dead
Age <= 2.5 gini = 0.089 samples = 128 value = [6, 122] class = Survived
Fare <= 22.904 gini = 0.5 samples = 110 value = [55, 55] class = Dead
SibSp <= 2.5 gini = 0.463 samples = 22 value = [8, 14] class = Survived
Pclass <= 1.5 gini = 0.297 samples = 452 value = [370, 82] class = Dead
gini = 0.0 samples = 1 value = [1, 0] class = Dead
Fare <= 26.125 gini = 0.076 samples = 127 value = [5, 122] class = Survived
Age <= 36.0 gini = 0.481 samples = 87 value = [35.0, 52.0] class = Survived
Parch <= 0.5 gini = 0.227 samples = 23 value = [20, 3] class = Dead
gini = 0.0 samples = 13 value = [0, 13] class = Survived
Fare <= 31.331 gini = 0.198 samples = 9 value = [8, 1] class = Dead
Age <= 51.5 gini = 0.466 samples = 100 value = [63, 37] class = Dead
Fare <= 51.698 gini = 0.223 samples = 352 value = [307, 45] class = Dead
gini = 0.169 samples = 43 value = [4, 39] class = Survived
gini = 0.024 samples = 84 value = [1, 83] class = Survived
gini = 0.468 samples = 83 value = [31, 52] class = Survived
gini = 0.0 samples = 4 value = [4, 0] class = Dead
gini = 0.0 samples = 1 value = [0, 1] class = Survived
gini = 0.165 samples = 22 value = [20, 2] class = Dead
gini = 0.0 samples = 5 value = [5, 0] class = Dead
gini = 0.375 samples = 4 value = [3, 1] class = Dead
gini = 0.489 samples = 80 value = [46, 34] class = Dead
gini = 0.255 samples = 20 value = [17, 3] class = Dead
gini = 0.209 samples = 338 value = [298, 40] class = Dead
gini = 0.459 samples = 14 value = [9, 5] class = Dead

Figure 5.2. Decision Tree visualization for max_depth = 4

Question 6

6.1. Train Full Tree

In this part of the project, a full Decision Tree classifier was trained on the Titanic dataset with no depth limitation. By setting max_depth=None, the model was allowed to grow until all leaves reached complete purity or no further useful splits were possible. The classifier was trained using the same training data prepared in the previous section, and its performance was evaluated on both the training and testing sets.

The results obtained from the unrestricted model were as follows:

```
Full Decision Tree (max_depth=None)
Train Accuracy: 0.9859550561797753
Test  Accuracy: 0.7877094972067039
Node count: 329
Max depth : 23
```

These results demonstrate that the model achieves extremely high performance on the training data, nearly memorizing the dataset and indicating very low bias. However, the significant reduction in test accuracy confirms that the model fails to generalize effectively to new data. The large number of nodes and the deep structure of the tree indicate excessive model complexity, which strongly suggests overfitting.

To highlight this behavior, the table below compares the shallow tree from the previous question (max_depth=4) with the unrestricted full-depth tree:

Model Type	Depth	Train Accuracy	Test Accuracy	Node Count	Model Behavior
Shallow Tree (max_depth=4)	4	0.8301	0.8492	27	Balanced complexity, good generalization
Full Tree (max_depth=None)	23	0.9859	0.7877	329	Overfitting, memorizes training data, weak generalization

Compared to the shallow tree, the full model exhibits a clear accuracy gap between training and testing performance. The limited-depth model provided a more stable balance between training and test results, making it more resistant to overfitting and easier to interpret. In contrast, the unrestricted model produces a highly fragmented decision boundary, learns noise and rare patterns in the training set, and becomes less reliable when evaluated on unseen samples.

Overall, removing the depth constraint significantly increases model complexity, reduces interpretability, and leads to overfitting. This confirms the necessity of applying depth control or pruning techniques when using Decision Trees in order to maintain generalization performance.

6.2. Training Decision Trees with Different Depths

In this section, the Decision Tree model was trained with multiple depth values in order to evaluate how model complexity affects bias, variance, and generalization performance. The tested depth settings were:

max_depth = 3, 5, 10, and None (unlimited depth)

For each configuration, the model was trained on the same training data and evaluated on the test set. The exact loop used to generate the accuracy results was:

```
train_accs, test_accs = [], []

for d in depth_list:
    clf = DecisionTreeClassifier(max_depth=d, random_state=42)
    clf.fit(X_train, y_train)

    train_acc = accuracy_score(y_train, clf.predict(X_train))
    test_acc  = accuracy_score(y_test,  clf.predict(X_test))

    train_accs.append(train_acc)
    test_accs.append(test_acc)

    print(f"\n=== max_depth = {d} ===")
    print("Train Accuracy:", train_acc)
    print("Test  Accuracy:", test_acc)
```

Table below shows accuracy of each model:

max_depth	Train Accuracy	Test Accuracy
3	0.8230	0.8379
5	0.8384	0.8379
10	0.9101	0.8547
None	0.9859	0.7765

Figure 6.1 below visualizes how training and testing accuracy change as the tree grows deeper.

The behavior of the model changes noticeably as depth increases:

- Underfitting (High Bias): For max_depth = 3 and partially max_depth = 5, both training and test accuracy remain relatively low. The model is too simple to capture the underlying patterns, resulting in limited predictive capability.
- Best Generalization Zone: At max_depth = 10, the model reaches a balanced state where the training accuracy improves but the test accuracy remains stable. This depth provides a

good trade-off between complexity and generalization performance, making it the most suitable configuration in this comparison.

- Overfitting (High Variance):
  With max_depth = None, the model memorizes the training data, reaching very high training accuracy (98.59%) while the test accuracy drops to 77.65%. This large performance gap confirms clear overfitting and unstable generalization.

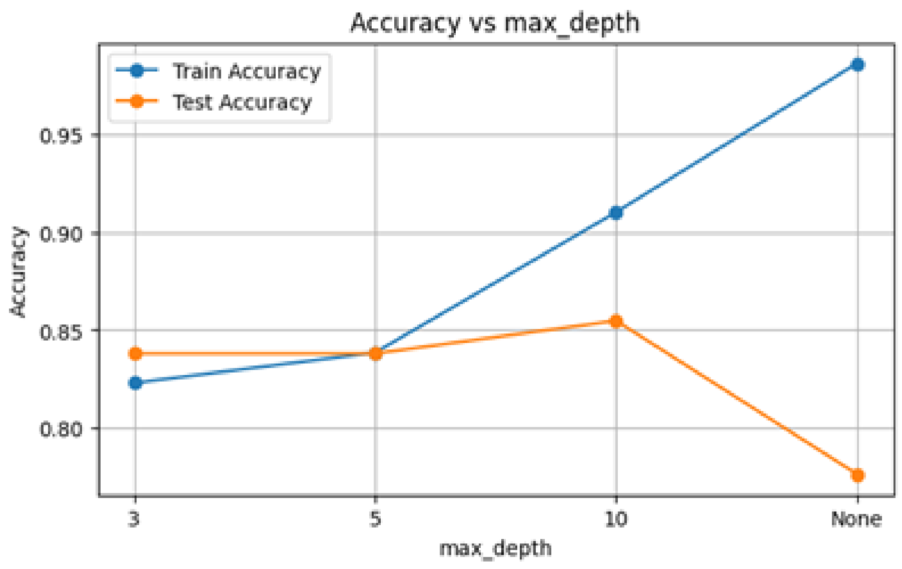


Figure 6.1. Accuracy vs max_depth for the titanic decision tree model

Overall, this experiment demonstrates that depth control is essential when training Decision Trees. Moderate depth values provide the best predictive balance, while excessively deep trees increase complexity, reduce interpretability, and produce overfitting.

6.3. Pruning the Decision Tree Using Cost-Complexity (ccp_alpha)

In this section, pruning was applied to the previously trained full-depth Decision Tree using the cost-complexity pruning method. The pruning path was extracted from the trained model to obtain a sequence of candidate ccp_alpha values. Only positive values of alpha were retained, and a new tree was trained for each alpha in order to evaluate how pruning affects generalization performance and model complexity. The number of positive alphas were 65.

The code used to generate the pruning sequence and evaluate each model is shown below:

```
path = full_tree.cost_complexity_pruning_path(X_train, y_train)

ccp_alphas_pos = path.ccp_alphas[path.ccp_alphas > 0]
print("Positive alphas:", len(ccp_alphas_pos))
pruned_clfs = []
train_accs_pruned = []
test_accs_pruned  = []
```

```
for alpha in ccp_alphas_pos:
    clf_p = DecisionTreeClassifier(random_state=42, ccp_alpha=alpha)
    clf_p.fit(X_train, y_train)

    pruned_clfs.append(clf_p)
    train_accs_pruned.append(accuracy_score(y_train,
clf_p.predict(X_train)))
    test_accs_pruned.append(accuracy_score(y_test,  clf_p.predict(X_test))
)
```

The highest-performing pruned model on the test set occurred at:

```
Best positive alpha
best_alpha = 0.002136309875813129
Train Accuracy: 0.8918539325842697
Test  Accuracy: 0.8715083798882681
```

This pruning operation reduced the structural size of the model significantly:

Model	Node Count	Depth	Test Accuracy
Full Tree (no limit)	329	23	0.7765
Pruned Tree (alpha = 0.002136...)	71	14	0.8715

The pruned model is considerably smaller than the original full-depth model while maintaining an improved or stable generalization accuracy, demonstrating the effectiveness of pruning in controlling variance and preventing overfitting.

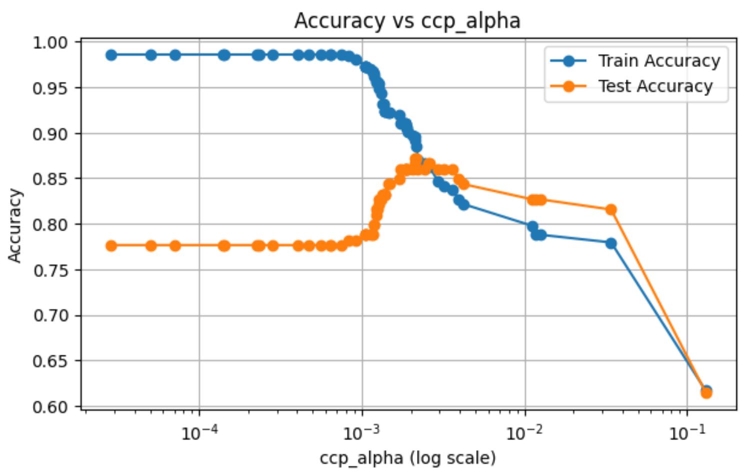


Figure 6.2. Accuracy vs ccp_alpha (log scale) for pruned models.

A visualization of the selected pruned tree is shown in figure 6.3. The structure is noticeably more compact, improving interpretability while maintaining competitive predictive performance.

Model	Depth	Size	Train Acc.	Test Acc.	Behavior
Shallow Tree (max_depth=4)	4	Small	0.8301	0.8492	Stable, interpretable, slight underfitting
Full Tree (no depth limit)	23	Very Large	0.9859	0.7765	Severe overfitting, weak generalization
Pruned Tree (best alpha)	14	Reduced	0.8918	0.8715	Best balance of complexity & performance

The results confirm the expected trend: limiting tree growth (either by depth or pruning) improves generalization. The pruned model offers the strongest balance between interpretability, structure, and predictive performance.

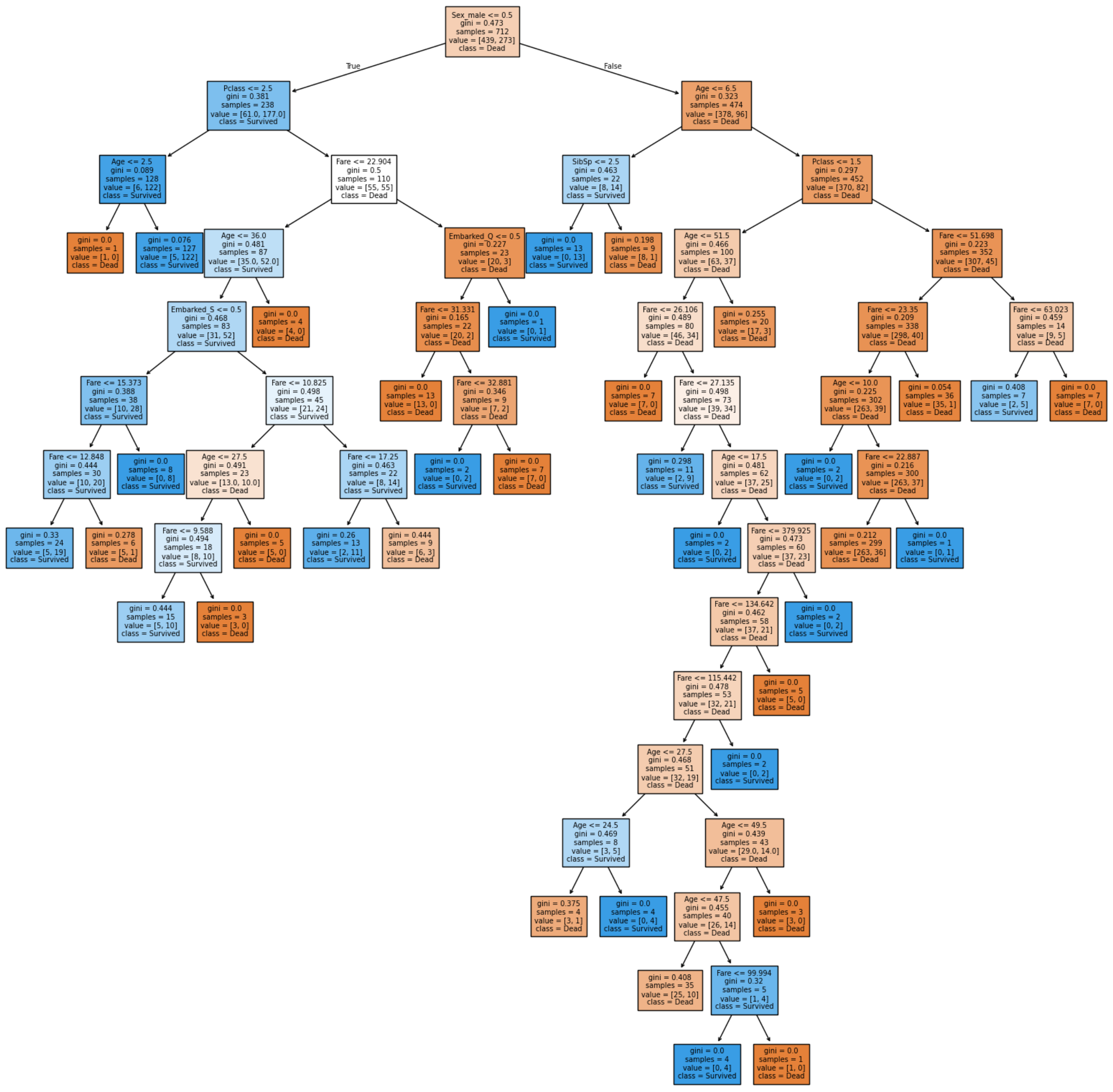

Sex_male <= 0.5 gini = 0.473 samples = 712 value = [439, 273] class = Dead
True
False
Pclass <= 2.5 gini = 0.381 samples = 238 value = [61.0, 177.0] class = Survived
Age <= 6.5 gini = 0.323 samples = 474 value = [378, 96] class = Dead
Age <= 2.5 gini = 0.089 samples = 128 value = [6, 122] class = Survived
Fare <= 22.904 gini = 0.5 samples = 110 value = [55, 55] class = Dead
SibSp <= 2.5 gini = 0.463 samples = 22 value = [8, 14] class = Survived
Pclass <= 1.5 gini = 0.297 samples = 452 value = [370, 82] class = Dead
gini = 0.0 samples = 1 value = [1, 0] class = Dead
gini = 0.076 samples = 127 value = [5, 122] class = Survived
Age <= 36.0 gini = 0.481 samples = 87 value = [35.0, 52.0] class = Survived
Embarked_Q <= 0.5 gini = 0.227 samples = 23 value = [20, 3] class = Dead
gini = 0.0 samples = 13 value = [0, 13] class = Survived
gini = 0.198 samples = 9 value = [8, 1] class = Dead
Age <= 51.5 gini = 0.466 samples = 100 value = [63, 37] class = Dead
Fare <= 51.698 gini = 0.223 samples = 352 value = [307, 45] class = Dead
Embarked_S <= 0.5 gini = 0.468 samples = 83 value = [31, 52] class = Survived
gini = 0.0 samples = 4 value = [4, 0] class = Dead
Fare <= 31.331 gini = 0.165 samples = 22 value = [20, 2] class = Dead
gini = 0.0 samples = 1 value = [0, 1] class = Survived
Fare <= 26.106 gini = 0.489 samples = 80 value = [46, 34] class = Dead
gini = 0.255 samples = 20 value = [17, 3] class = Dead
Fare <= 23.35 gini = 0.209 samples = 338 value = [298, 40] class = Dead
Fare <= 63.023 gini = 0.459 samples = 14 value = [9, 5] class = Dead
Fare <= 15.373 gini = 0.388 samples = 38 value = [10, 28] class = Survived
Fare <= 10.825 gini = 0.498 samples = 45 value = [21, 24] class = Survived
gini = 0.0 samples = 13 value = [13, 0] class = Dead
Fare <= 32.881 gini = 0.346 samples = 9 value = [7, 2] class = Dead
gini = 0.0 samples = 7 value = [7, 0] class = Dead
Fare <= 27.135 gini = 0.498 samples = 73 value = [39, 34] class = Dead
Age <= 10.0 gini = 0.225 samples = 302 value = [263, 39] class = Dead
gini = 0.054 samples = 36 value = [35, 1] class = Dead
gini = 0.408 samples = 7 value = [2, 5] class = Survived
gini = 0.0 samples = 7 value = [7, 0] class = Dead
Fare <= 12.848 gini = 0.444 samples = 30 value = [10, 20] class = Survived
gini = 0.0 samples = 8 value = [0, 8] class = Survived
Age <= 27.5 gini = 0.491 samples = 23 value = [13.0, 10.0] class = Dead
Fare <= 17.25 gini = 0.463 samples = 22 value = [8, 14] class = Survived
gini = 0.0 samples = 2 value = [0, 2] class = Survived
gini = 0.0 samples = 7 value = [7, 0] class = Dead
gini = 0.298 samples = 11 value = [2, 9] class = Survived
Age <= 17.5 gini = 0.481 samples = 62 value = [37, 25] class = Dead
gini = 0.0 samples = 2 value = [0, 2] class = Survived
Fare <= 22.887 gini = 0.216 samples = 300 value = [263, 37] class = Dead
gini = 0.33 samples = 24 value = [5, 19] class = Survived
gini = 0.278 samples = 6 value = [5, 1] class = Dead
Fare <= 9.588 gini = 0.494 samples = 18 value = [8, 10] class = Survived
gini = 0.0 samples = 5 value = [5, 0] class = Dead
gini = 0.26 samples = 13 value = [2, 11] class = Survived
gini = 0.444 samples = 9 value = [6, 3] class = Dead
gini = 0.0 samples = 2 value = [0, 2] class = Survived
Fare <= 379.925 gini = 0.473 samples = 60 value = [37, 23] class = Dead
gini = 0.212 samples = 299 value = [263, 36] class = Dead
gini = 0.0 samples = 1 value = [0, 1] class = Survived
gini = 0.444 samples = 15 value = [5, 10] class = Survived
gini = 0.0 samples = 3 value = [3, 0] class = Dead
Fare <= 134.642 gini = 0.462 samples = 58 value = [37, 21] class = Dead
gini = 0.0 samples = 2 value = [0, 2] class = Survived
Fare <= 115.442 gini = 0.478 samples = 53 value = [32, 21] class = Dead
gini = 0.0 samples = 5 value = [5, 0] class = Dead
Age <= 27.5 gini = 0.468 samples = 51 value = [32, 19] class = Dead
gini = 0.0 samples = 2 value = [0, 2] class = Survived
Age <= 24.5 gini = 0.469 samples = 8 value = [3, 5] class = Survived
Age <= 49.5 gini = 0.439 samples = 43 value = [29.0, 14.0] class = Dead
gini = 0.375 samples = 4 value = [3, 1] class = Dead
gini = 0.0 samples = 4 value = [0, 4] class = Survived
Age <= 47.5 gini = 0.455 samples = 40 value = [26, 14] class = Dead
gini = 0.0 samples = 3 value = [3, 0] class = Dead
gini = 0.408 samples = 35 value = [25, 10] class = Dead
Fare <= 99.994 gini = 0.32 samples = 5 value = [1, 4] class = Survived
gini = 0.0 samples = 4 value = [0, 4] class = Survived
gini = 0.0 samples = 1 value = [1, 0] class = Dead

Figure 6.3. Pruned Decision Tree using best_alpha = 0.002136

Question 7

7.1. Ensemble Learning

Ensemble learning is a technique in which multiple models are trained, and their outputs are then combined to produce a more accurate final prediction.

The core idea is that the wisdom of the crowd is usually more accurate than the opinion of a single individual. The technical reasons include:

1. Variance reduction: If a single model is sensitive to data noise (overfitting), combining it with other models allows random errors to cancel each other out.
2. Bias reduction: Some ensemble methods can learn complex patterns by combining simple models' patterns that a single simple model cannot capture on its own.
3. Increased stability: Ensemble models are less sensitive to sudden changes in the input data and therefore provide more reliable predictions.

Suppose you are ill. Instead of trusting only one doctor, you consult five doctors. If four of them give the same diagnosis, your confidence in the result is much higher than if you relied on just one opinion.

7.2. Difference Between Diversification and Aggregation

For an ensemble system to work effectively, it needs two essential components; diverse models and a strategy for combining their outputs.

7.2.1 Diversification

Diversification refers to how we create models that are different from one another. If all models think in exactly the same way, combining them provides no benefit because they will all make the same mistakes.

Ways to create diversity:

- Using different subsets of the data (bagging).
- Using different subsets of features for each model (Random Forest).
- Using different algorithms (combining a decision tree with a KNN model).

7.2.2. Aggregation

Aggregation refers to the strategy used to combine the outputs of different models to reach a final decision.

Common aggregation methods:

- Voting: In classification problems, the class with the most votes is selected.
- Averaging: In regression problems, the mean of the predicted values is computed.
- Weighted aggregation: Models with higher accuracy are given more influence in the final decision.

7.2.3. Comparison

Diversification ensures that models think differently and make different types of errors.

Aggregation ensures that these diverse perspectives are combined correctly.

The combination of these two concepts is why ensemble learning usually performs better than a single model.

Concept	Main Focus	Goal
Diversification	Building independent and different models	Preventing correlated errors and covering each other's weaknesses
Aggregation	Intelligent combination of predictions	Producing a single final value or class

7.3. Conceptual Example

Using Ensemble Learning in the real world especially in domains where accuracy and security are critical is a necessity. One of the most prominent examples is credit card fraud detection systems.

In such systems, because fraudsters constantly change their tactics, a single model can be easily bypassed. However, an ensemble of models can uncover highly complex and evolving patterns.

Intelligent Bank Fraud Detection System

In this scenario, a bank uses a combination of multiple models to evaluate the legitimacy of a transaction in just a few milliseconds

1. System Inputs (Features)
   - Transaction details: transaction amount, time of transaction, and currency type.
   - Location information: distance between the current transaction location and the last successful transaction (for example, making transactions in two different countries within one hour is suspicious).
   - User behavior history: average monthly spending, categories of merchants the user usually shops from.
   - Technical information: IP address, device type (mobile or laptop), and operating system.
2. Ensemble Process in This Example
   - Diversification: One model may focus primarily on unusual transaction amounts, another on suspicious geographic locations, and a third on the frequency of consecutive transactions.
   - Aggregation: When a transaction occurs, each model provides its assessment. Using a method such as weighted voting, if the majority of the stronger models classify the transaction as suspicious, the system blocks it.
3. System Outputs (Targets)
   - Class 1 (Fraud): The transaction is fraudulent → card is blocked and a notification is sent to the customer.
   - Class 0 (Legitimate): The transaction is legitimate → transaction is approved.
   - Risk score: A value between 0 and 100 indicating the probability of fraud.

Fraud detection faces two major challenges that ensemble methods address effectively.

We do not want to incorrectly label a legitimate customer’s purchase as fraud and cause frustration. Combining models improves precision and decision reliability, so we reducing false positives.

Fraudsters often move small amounts at irregular intervals. A single simple model may miss this behavior, but an ensemble where each model captures a different aspect of the data can detect such complex, combined patterns and by this way we can discover hidden patterns.

Question 8

Part A: Bootstrap Sampling in Bagging

In Bagging (which stands for Bootstrap Aggregating), the goal is to create several different data subsets from the original dataset in order to train multiple models. The Bootstrap Sampling process works as follows.

1. Sampling with replacement: If our original dataset contains n samples, to create a new dataset, we randomly select exactly n samples from the original dataset.
2. Repeatability: The key point is that after each selection, the sample is returned to the "bag". Therefore, a particular sample may be selected multiple times in one new dataset, while another sample may not be selected at all.
3. Result: We obtain several datasets that are the same size as the original dataset, but with slightly different data distributions.

Part B: Three Possible Bootstrap

Suppose we want to create 3 new datasets. Each dataset must contain 4 samples, randomly selected with replacement from the original data.

This is one possible example:

1. Sample 1: $(2, 0)$
2. Sample 2: $(3, 0)$
3. Sample 3: $(8, 1)$
4. Sample 4: $(9, 1)$

Example Bootstrap datasets:

- B_1 set:

$$B_1 = \{(2, 0), (2,0), (8, 1), (9, 1)\}$$

- B_2 set:

$$B_2 = \{(3, 0), (8,1), (9, 1), (9, 1)\}$$

- B_3 set:

$$B_3 = \{(2, 0), (3,0), (8, 1), (9, 1)\}$$

In practice, about 63% of the data in each bag are unique samples, and the remaining 37% are repeated samples.

Part C: Random Forest vs. Single Decision Tree

A Random Forest generally performs better than a single Decision Tree for the following reasons.

1. Variance Reduction and Prevention of Overfitting
    - A single decision tree has a high potential for overfitting; it memorizes details and noise in the training data and performs poorly on new data.

    - Random Forest reduces this variance by averaging (in regression) or majority voting (in classification) across many trees. Errors and noise from different trees tend to cancel each other out.
2. Greater Diversity
    - In addition to randomly selecting data rows (Bagging), Random Forest also considers only a random subset of features at each split in the tree.
    - This prevents the trees from becoming too similar. When trees are diverse, their ensemble becomes much more powerful than a single tree.

A single decision tree is like one “expert” who may be biased or make mistakes, whereas a Random Forest works like the wisdom of crowds, which usually leads to more accurate and stable decisions.

Question 9

9.1. Concept of Boosting and Its Difference from Bagging

The Boosting method is an ensemble learning technique whose goal is to turn several weak learners into a single strong learner.

The main idea is that models are trained sequentially. Each new model tries to correct the mistakes of the previous model. That is, the second model focuses on the data points that the first model predicted incorrectly.

Feature	Bagging	Boosting
Construction method	Models are built in parallel and independently.	Models are built sequentially.
Main objective	Reduce variance.	Reduce bias and variance.
Data weighting	All samples have an equal chance of being selected.	Hard samples are given more weight.
Combining results	Simple averaging or voting.	Weighted averaging.

9.2. The Role of Samples Weights in AdaBoost Algorithm

In the AdaBoost algorithm, sample weights are the driving engine of the algorithm.

Initialization: At the beginning, all training samples are given equal weight. After each training step, the model evaluates its performance. When the next model is trained, it pays much more attention to samples that have higher weights. It is as if the algorithm zooms in on the hard data or sees them in a larger font so that it is forced to learn them.

The weights tell the algorithm, you haven't learned these difficult data points yet; in the next step, your priority should be learning them.

9.2.1. Initialization

Initially, each of the N training samples is assigned an equal weight.

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

9.2.2. Computation of Model Error

At each stage t a weak learner is trained, and its weighted error is computed.

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

In this formula, $\mathbb{I}$ is an indicator function that takes the value 1 if the model predicts incorrectly and 0 otherwise. The weight w_i causes mistakes on more important data points to produce a larger penalty or error for the model.

9.2.3. Computation of the Model Weight

Now we determine how powerful this model is and how much it contributes to the final decision.

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

- If the error is small: α_t becomes positive and large (the model is reliable).
- If the error is 50%: α_t becomes zero (the model is ineffective).

9.2.4. Updating the Sample Weights

This is the most important part for learning from mistakes. The weight of each data point is updated for the next iteration.

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

- If the model predicts correctly $y_i = h_t(x_i)$, the exponent of the exponential term becomes negative, so the weight w_i decreases. This means the algorithm pays less attention to this data point in the next stage because it was easy to learn.
- If the model predicts incorrectly $y_i \neq h_t(x_i)$, the exponent of the exponential erm becomes positive, so the weight w_i increases. This data point will have a louder voice in the next stage so that the next model is forced to learn it correctly.

9.2.5. Final Prediction

Finally, the prediction is obtained from the weighted combination of all models.

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

Here again, α_t plays the role of the weight or voting power of each model. A more accurate model has a greater influence on the final output.

Weight type	Symbol	Main role
Sample weights	w_i	Forcing the next model to focus on difficult samples and previous mistakes.
Model weights	α_t	Determining how much influence each model has in the final vote based on its accuracy.

9.3. Boosting and Noise

Boosting can create very strong models because, by combining multiple weak models and focusing step by step on errors, it builds a structure that reduces both bias and overall error. This gradual focus allows complex patterns that cannot be learned by a single model to be effectively identified.

However, this same characteristic makes Boosting sensitive to noise. If the data contain noisy samples or incorrect labels, Boosting also treats these samples as hard samples and increases their weights in subsequent stages. As a result, the model puts excessive effort into learning the noise, which can lead to overfitting and reduced generalization to new data.

Question 10

10.1. What is Stacking and how does it use a Meta Learner?

Stacking (Stacked Generalization) is an ensemble learning technique in which multiple base models are trained independently, and their predictions are then combined by a second-level model called a Meta Learner. Instead of averaging or voting (as in Bagging or Boosting), Stacking trains a new model to learn the optimal way to blend the outputs of the first-level models.

The Meta Learner receives the predictions of the base models as input features and learns how to weight and combine them in order to produce a final prediction. This structure allows Stacking to exploit the strengths of each model while reducing their individual weaknesses, often resulting in higher accuracy and better generalization.

10.2. Why must Cross-Validation be used to train the Meta Learner?

Cross-Validation is required when training the Meta Learner to prevent information leakage from the training process. If first-level models are evaluated on the same data they were trained on, their predictions will be overly optimistic and biased. Using K-Fold Cross-Validation ensures that the predictions fed into the Meta Learner are generated from models that have not seen the corresponding samples during training.

This produces unbiased out-of-fold predictions, which allows the Meta Learner to learn true generalization patterns rather than memorized outputs. As a result, the final stacked model becomes more stable, less overfitted, and better prepared for unseen data.

10.3. Example of a Stacking Architecture

An example Stacking system can be structured as what we will show.

10.3.1. Level-1 Base Learner

Model 1: Logistic Regression

Model 2: Random Forest

Model 3: Support Vector Machine (SVM)

Each of these models is trained on the same training dataset and produces predictions. Their outputs are then passed to the second stage.

10.3.2. Level-2 Meta Learner

Model: Gradient Boosting or Linear Regression

The Meta Learner receives the predictions from the three base models as inputs:

- Input: [Prediction_LogReg, Prediction_RF, Prediction_SVM]
- Output: Final Stacked Prediction

This structure allows the stacked model to leverage linear decision boundaries (from Logistic Regression), nonlinear tree-based patterns, and margin-based separation, resulting in improved generalization performance.

Question 11

Part A: Four Advantages of Ensemble Learning

1. Higher Accuracy and Performance

   The main reason for using ensemble methods is that they usually perform better than any single base learner. Combining the opinions of multiple models reduces the overall error.

2. Overfitting Reduction

   Techniques such as Bagging, by averaging across different models, significantly reduce model variance. This prevents the model from memorizing the training data and helps it perform better on new data.

3. Robustness and Stability

   If the training data change slightly, a single decision tree may completely change its structure. However, in ensemble learning, since the final result is obtained by combining dozens or hundreds of models, noise and small changes in the data have little effect on the final outcome.

4. Managing the Bias Variance Trade-off

   Ensemble methods can address both sides of the error. Boosting methods focus on reducing bias, while Bagging focuses on reducing variance. Using these methods helps create a more balanced model.

Part B: Four Disadvantages of Ensemble Learning

1. Loss of Interpretability

   This is the biggest drawback. A simple decision tree can be drawn on paper and its logic understood. But when dealing with 1000 trees in a Random Forest or complex Boosting formulas, the model becomes a black box, and explaining why the model reached a particular decision becomes very difficult.

2. High Training Cost

   Training an ensemble model means training dozens or hundreds of smaller models. This requires more hardware resources and much more time compared to training a simple model.

3. Slower Prediction

   When making a prediction for new data, that data must pass through all the underlying models and the results must be combined. This delay can be challenging for applications that require real-time responses.

4. Hyperparameter Tuning Complexity

   Ensemble models have many parameters. Finding the optimal combination of these parameters requires extensive trial and error and a high level of technical expertise.

Question 12

12.1. Data Preprocessing

We do the following steps:

1. Removing unnecessary columns: Columns such as `Name`, `Ticket`, and `Cabin`, and `PassengerId` do not provide direct information to the model for predicting survival.
2. Handling missing values
   - `Age`: We fill it with the median to introduce less noise.
   - `Embarked`: We fill it with the mode.
   - `Fare`: Fill it with the median.
3. One-Hot Encoding: Text variables such as `Sex` and `Embarked` are converted into 0 and 1 numerical values.

We write a function to perform the steps mentioned above, which is as follows:

```python
def preprocess_data(df):
    df_clean = df.copy()

    cols_to_drop = ['Name', 'Ticket', 'Cabin', 'PassengerId']
    df_clean = df_clean.drop(columns=cols_to_drop, errors='ignore')

    df_clean['Age'] = df_clean['Age'].fillna(df_clean['Age'].median())

    if 'Embarked' in df_clean.columns:
        df_clean['Embarked'] = 
df_clean['Embarked'].fillna(df_clean['Embarked'].mode()[0])

    if 'Fare' in df_clean.columns:
        df_clean['Fare'] = 
df_clean['Fare'].fillna(df_clean['Fare'].median())

    df_clean = pd.get_dummies(df_clean, columns=['Sex', 'Embarked'], 
drop_first=True)
    df_clean = df_clean.astype(float)

    return df_clean
```

12.1.1. Initial State of the Data

We have 891 passengers in the training set and 418 passengers in the test set.

- Column `Cabin`: It had the highest number of missing values. This means about 77% of the information in this column was missing, so removing it was the right decision.
- Column `Age`: It had about 177 missing values in training, which is a significant number and needed to be filled.
- Column `Embarked`: It had only 2 missing values.

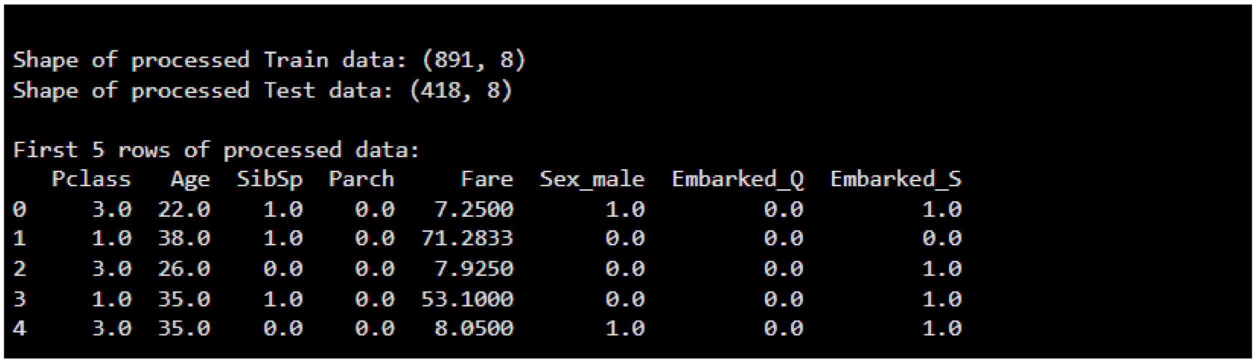
- Column `Fare`: It had 1 missing value in the test data.

12.1.2. Operations Performed on the Data

Our code applied the following changes.

- Removing redundant columns: The columns `Name`, `Ticket`, `Cabin`, and `PassengerId` were removed. This reduced the data dimensionality and noise.
- Filling missing values: `Age` was filled with the median value. `Fare` and `Embarked` were also filled with appropriate values so that no missing entries remained.
- Converting categorical data to numerical: Gender, which was `male` and `female`, was converted into a column 1 and 0.
- Embarkation port, which had values `S`, `C`, `Q`, was converted using One-Hot encoding into two columns `Embarked_Q` and `Embarked_S`.

The output of the preprocessing code is shown in figure 12.1.



```
Shape of processed Train data: (891, 8)
Shape of processed Test data: (418, 8)

First 5 rows of processed data:
   Pclass   Age  SibSp  Parch     Fare  Sex_male  Embarked_Q  Embarked_S
0     3.0  22.0    1.0    0.0   7.2500       1.0         0.0         1.0
1     1.0  38.0    1.0    0.0  71.2833       0.0         0.0         0.0
2     3.0  26.0    0.0    0.0   7.9250       0.0         0.0         1.0
3     1.0  35.0    1.0    0.0  53.1000       0.0         0.0         1.0
4     3.0  35.0    0.0    0.0   8.0500       1.0         0.0         1.0
```

Fig 12.1. Data information after preprocessing

Data now have 891 rows and 8 clean, numerical columns.

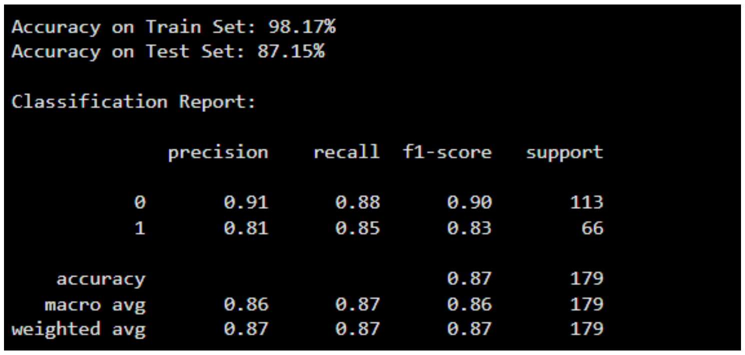
Remaining Columns: Pclass, Age, SibSp. Parch, Fare, Sex_male, Embarked_Q, Embarked_S

12.2. Training the model using a Random Forest Classifier

We train the Random Forest model with `random_state = 93` and `n_estimators = 200`.

```
rf_model = RandomForestClassifier(n_estimators=200, random_state=93)
```

The metrics of the model is shown in figure 12.2.



```
Accuracy on Train Set: 98.17%
Accuracy on Test Set: 87.15%

Classification Report:

              precision    recall  f1-score   support

           0       0.91      0.88      0.90       113
           1       0.81      0.85      0.83        66

    accuracy                           0.87       179
   macro avg       0.86      0.87      0.86       179
weighted avg       0.87      0.87      0.87       179
```

Fig. 12.2. Metrics of Random Forest model

1. Overall Accuracy Analysis

   Training Accuracy: 98.17%

   Test Accuracy: 87.15%

2. Classification Report Analysis

   Class 0: Precision 0.91 shows that when the model predicts that someone has died, there is 91% confidence that the prediction is correct. This is a very high precision. Recall 0.88 shows that the model has been able to identify 88% of all actual deceased passengers.

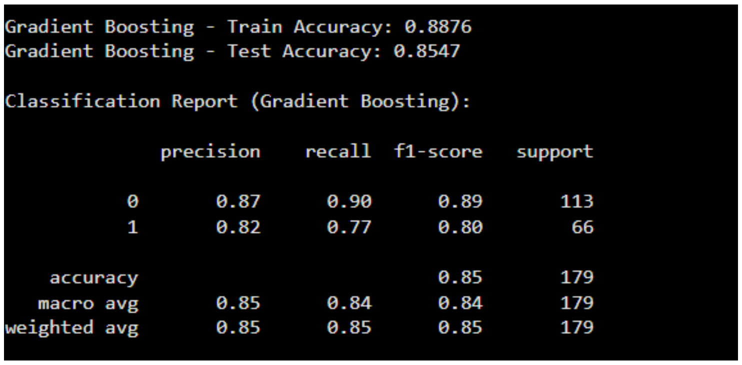
   Class 1: Precision 0.8 shows that when the model predicts that someone survived, it is correct 81% of the time (19% error). Recall 0.85 shows that interestingly, the model has been able to identify 85% of the survivors. In the previous result, this number was lower. This improvement in survivor recall is the main factor behind the increase in your overall accuracy.

12.3. Training the model using a GDBoost Classifier

We train the GDBoost model like Random Forest model.

```
gb_model = GradientBoostingClassifier(n_estimators=200, random_state=93)
```

The model metrics of the model is shown in figure 12.3.



```
Gradient Boosting - Train Accuracy: 0.8876
Gradient Boosting - Test Accuracy: 0.8547

Classification Report (Gradient Boosting):

              precision    recall  f1-score   support

           0       0.87      0.90      0.89       113
           1       0.82      0.77      0.80        66

    accuracy                           0.85       179
   macro avg       0.85      0.84      0.84       179
weighted avg       0.85      0.85      0.85       179
```

Fig. 12.3. Metrics of GDBoost model

The Random Forest model, with an accuracy of 87.15%, has performed better than Gradient Boosting 85.47%. This indicates that for this particular data split, the voting and averaging" approach of Random Forest has worked better.

The gap between training and test accuracy is smaller. This means the model behaves more reasonably and has memorized the data less.

The larger gap indicates that the model has somewhat memorized the data. However, since it achieves higher final accuracy, this level of overfitting is acceptable for now

These two models show completely opposite behaviors in class detection.

In this particular experiment, Random Forest is the winner because it provides a better balance in detecting both classes. Although your Gradient Boosting model is more stable (less overfitting), it performs poorly in identifying survivors, which is usually the more important class.

Metric	Random Forest	Gradient Boosting	Winner
Training Accuracy	98.17%	88.76%	Random Forest
Test Accuracy	87.15%	85.47%	Random Forest
Overfitting Level (Gap)	11% difference	3% difference	Gradient Boosting
Survivor Detection	85%	77%	Random Forest
Deceased Detection	88%	90%	Gradient Boosting

12.4. Overfitting & Underfitting

Random Forest model suffers from overfitting, in this model, the accuracy on the training data (Train) is very high and close to 100% (98.17%), but the accuracy on the test data drops to 87.15%.

The presence of a gap of about 11% between training and test accuracy indicates that the model has memorized the training data instead of learning general patterns. This usually happens in Random Forest when tree depth is not limited or when the number of trees (`n_estimators`) is large, causing the model to also learn noise from the training data.

Gradient Boosting model has a good fit, The accuracy on the training data is 88.76%, and on the test data it is 85.47%. The gap between these two numbers is very small (about 3.3%). This means the model has learned patterns that hold true in both the training and test datasets. Although its overall test accuracy is slightly lower than Random Forest, this model is more reliable because it has better generalization ability and behaves more stably on completely new data.

Neither of the two models suffers from underfitting.

Model	Training Accuracy	Test Accuracy	Status	Generalization Ability
Random Forest	98.17%	87.15%	Overfitting	Weaker
Gradient Boosting	88.76%	85.47%	Good Fit	Excellent

If we want to use this project in a real-world setting, the Gradient Boosting model is the better choice, because its behavior on real data is more consistent with the reported results.