CAP 5610
Assignment #1 Solution

January 26, 2025

Arman Sayan

---

1 Decision Tree Basics [30 points]

The goal of this assignment is to test and reinforce your understanding of Decision Tree Classifiers.

1. [5 points] How many unique, perfect binary trees of depth 3 can be drawn if we have 5 attributes? By depth, we mean the depth of the splits, not including the nodes that only contain a label (see Figure 1). So, a tree that checks just one attribute is a depth one tree. By perfect binary tree, we mean every node has either 0 or 2 children, and every leaf is at the same depth. Note also that a tree with the same attributes but organized at different depths is considered "unique". Do not include trees that test the same attribute along the same path in the tree.

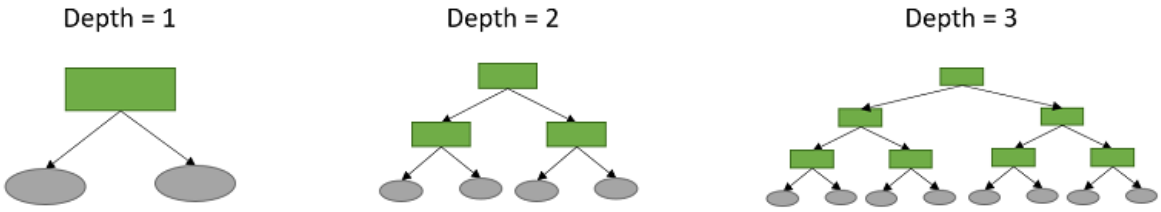

Figure 1: Example of perfect binary trees with different depths.

Ans:

Based on these definitions, to calculate the number of unique, perfect binary trees of depth 3 with 5 attributes, we need to clear out some key facts about this tree.

First of all, since the depth of the tree is 3, This means the tree has 7 internal nodes, namely 1 root, 2 children, and 4 grandchildren, and every path from the root to a leaf is of length 3.

Secondly, since this tree has the perfect binary tree structure, every internal node should split into exactly two children, and every leaf should be at the same depth.

Now that we clearly understand the structure of the tree, we can calculate the number of unique trees.

The first step is to choose 3 attributes for a path. There are 5 attributes which can be (A,B,C,D,E), and for any given path, we need to select 3 distinct attributes. The number of ways to do this is

$$\binom{5}{3} = \frac{5!}{3!(5-3)!} = 10$$

So, there are 10 ways to choose which 3 attributes will appear along a given path.

The second step is to choose the order of the attributes along the path. Once we have selected 3 attributes for a path, we can arrange them in 3! ways since the order of the attributes matters.

$$3! = 6$$

So, for each choice of attributes, there are 6 unique ways to organize them along a path.

Since the tree is a perfect binary tree, all paths must have the same set of attributes arranged in the same way.

Thus, we only count how many ways we can assign attributes to the tree consistently, which is

Number of ways to choose 3 attributes $\times$
Number of ways to arrange them in depth order

$$\boldsymbol{\binom{5}{3} \cdot 3! = 10 \cdot 6 = 60}$$

Therefore, there are **60** unique, perfect binary trees of depth 3 with 5 attributes.

2. [5 points] In general, for a problem with A attributes, how many unique perfect D depth trees can be drawn? Assume $A >> D$.

Ans:

To calculate the number of unique perfect binary trees of depth D with A attributes $A >> D$, we need to generalize the reasoning we used in earlier question.

The first step is to choose D attributes from the A available attributes. The number of ways to do this is actually

$$\binom{A}{D} = \frac{A!}{D!(A-D)!}$$

The second step is Once the D attributes are chosen, we must arrange them in a specific order to determine their position in the tree, which might be root, children, grandchildren, etc., the number of ways to arrange D attributes is

$$D!$$

The last step is to combine these values to calculate the total number of unique trees. The general formula for total number of unique perfect binary trees of depth D with A attributes is

$$\textbf{Total Trees} = \binom{\boldsymbol{A}}{\boldsymbol{D}} \cdot \boldsymbol{D!} = \frac{\boldsymbol{A!}}{\boldsymbol{D!(A-D)!}} \cdot \boldsymbol{D!} = \frac{\boldsymbol{A!}}{\boldsymbol{(A-D)!}}$$

3. [10 points] Consider the following dataset from Table 1 for this problem. Given the five attributes on the left, we want to predict if the student got an A in the course. Create 2 decision trees for this dataset. For the first, only go to depth 1. For the second go to depth 2. For all trees, use the ID3 entropy algorithm from class. For each node of the tree, show the decision, the number of positive and negative examples and show the entropy at that node.

   Hint: There are a lot of calculations here. You may want to do this programatically.

Early	Finished HMK	Senior	Likes Coffee	Liked The Last Jedi	A
1	1	0	0	1	1
1	1	1	0	1	1
0	0	1	0	0	0
0	1	1	0	1	0
0	1	1	0	0	1
0	0	1	1	1	1
1	0	0	0	1	0
0	1	0	1	1	1
0	0	1	0	1	1
1	0	0	0	0	0
1	1	1	0	0	1
0	1	1	1	1	0
0	0	0	0	1	0
1	0	0	1	0	1

Table 1: Toy Data-set for Task 1: Decision Tree Basics.

Ans:

To create two decision trees (depth 1 and depth 2) using the ID3 entropy algorithm, we need to calculate the entropy for the root node and each split, and then select the attribute that minimizes the weighted average entropy, namely maximizes information gain at each step, and split the dataset accordingly.

The first step is to calculate the entropy for the root node.

$$H(S) = -p_1 \log_2 p_1 - p_0 \log_2 p_0$$

where p_1 is the probability of $(A = 1)$ and p_0 is the probability of $(A = 0)$.

The second step is for each attribute, we need to calculate the weighted average entropy of splitting on that attribute as below

$$H_{split} = \sum_{v \in Values} \frac{|S_v|}{|S|} H(S_v)$$

where S_v is the subset of the dataset where the attribute takes the value v, and $H(S_v)$ is its entropy.

The third step is to calculate the information gain for each attribute, which is

$$IG = H(S) - H_{split}$$

The attribute with the highest information gain is selected as the attribute to be splitted.

The last step is to repeat this process for each subset until the tree reaches the desired depth.

The Python implementation of this algorithm is included in the appendix named as

CAP_5610_Assignment_1_Solution_Arman_Sayan.ipynb

The decision trees for depth 1 and depth 2 computed by this Python code are visualized in the following figures:

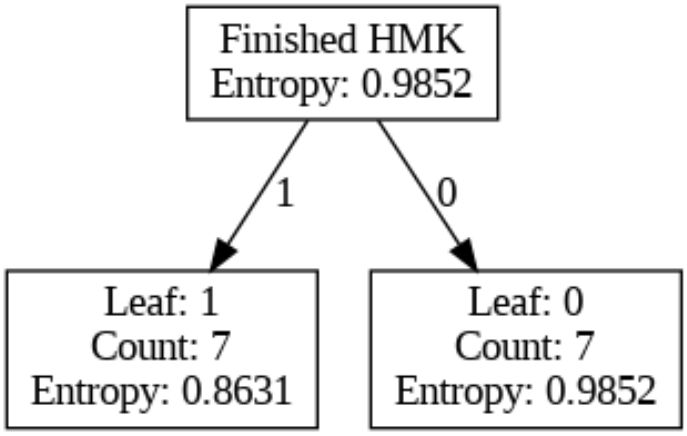

Figure 2: Decision Tree with Depth 1

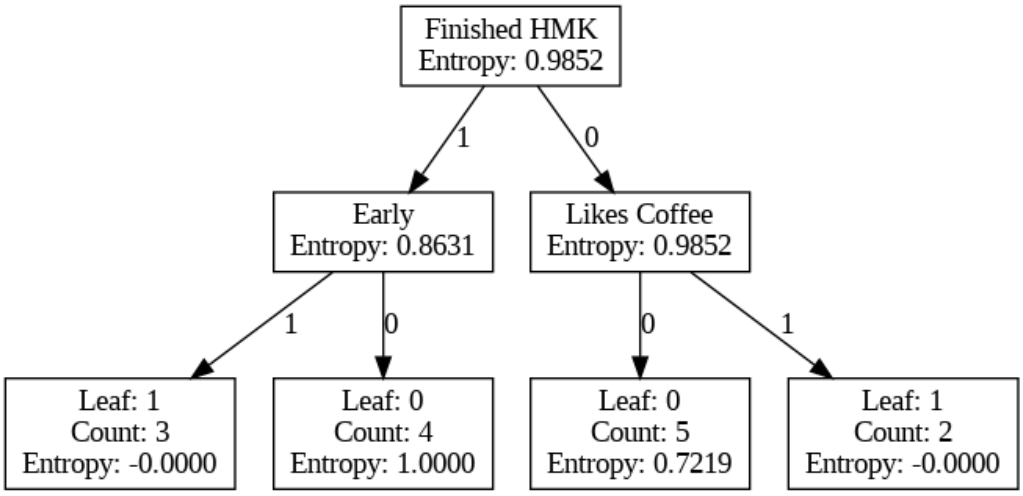

Figure 3: Decision Tree with Depth 2

4. [5 points] Make one more decision tree. Use the same procedure as in (3), but make it depth 3. Now, given these three trees, which would you prefer if you wanted to predict the grades of 10 new students who are not included in this data-set? Justify your choice.

Ans:

The decision tree for depth 3 computed by the same Python code used in (3) is visualized in the following figure:

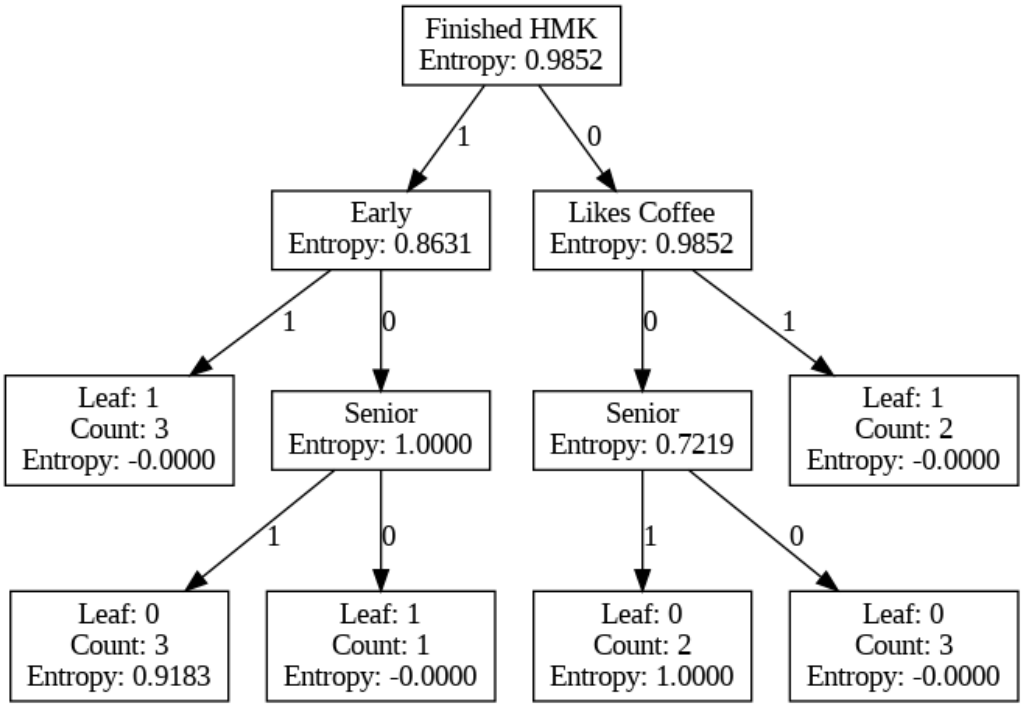

Figure 4: Decision Tree with Depth 3

Given these three trees, if we wanted to predict the grades of 10 new students who are not included in this data-set,

I would prefer the decision tree with depth 2. The reason for this choice is that the decision tree with depth 2 is the overall model among the three trees.

The decision tree with depth 1 is too simple and may not capture the underlying patterns in the data, while the decision tree with depth 3 is too complex and may overfit the training data.

The decision tree with depth 2 strikes a good balance between simplicity and complexity, making it more likely to generalize well to new, unseen data.

Furthermore, we can also compare the average entropy of the leaf nodes for each tree. A lower average entropy at the leaves implies a more confident prediction at each leaf. This would make it preferable if accuracy is the primary goal.

Based on these calculated decision trees, the average entropy of the leaf nodes for each tree is as follows:

- Depth 1 Tree: 0.9242
- Depth 2 Tree: 0.4305
- Depth 3 Tree: 0.3197

We can see that the average entropy of the leaf nodes decreases as the depth of the tree increases. This implies that the decision tree with depth 3 is more confident in its predictions compared to the other trees. However, there is not significant difference between depth 2 value and depth 3 value and there is still the chance that a decision tree with depth 3 might overfit the data, making it less generalizable to new students.

5. [5 points] Consider a new definition of a "realizable" case: "For some fixed concept class C, such as decision trees, a realizable case is one where the algorithm gets a sample consistent with some concept $c \in C$. In other words, for decision trees, a case is realizable if there is some tree that perfectly classifies the data-set.

   If the number of attributes A is sufficiently large, under what condition would a dataset not be realizable for decision trees of no fixed depth? Prove that the dataset is unrealizable if and only if that condition is true.

Ans:

For a dataset to not be realizable for decision trees of depth n, there should be at least one combination of input attributes and labels in the dataset that cannot be represented or separated by a decision tree of depth n, regardless of the size of the attribute space A.

For instance, a decision tree of depth n can split the data into at most 2^n leaf nodes, and each leaf node corresponds to one subset of the input space uniquely.

So, if the dataset has more unique combinations of input features that need to be classified differently than can be represented by 2^n leaf nodes, then the dataset cannot be realizable for a decision tree of depth n.

Formally, for a dataset with m distinct examples, a depth with n decision tree can perfectly classify the dataset only if $m \leq 2^n$.

It is also important to note that the size of the attribute space A does not affect the realizability of the dataset. A larger A provides more potential splits for the tree. However, the depth of the tree, not the attribute count, determines the maximum number of partitions 2^n that the tree can form.

Therefore, **if $m > 2^n$, the dataset is not realizable for decision trees of n, regardless of the size of the attribute space A.**

2 Application of Decision Tree on Real-Word Data-set [25 points]

1. [10 points] Train a decision tree classifier using the data file. Vary the cut-off depth from 2 to 10 and report the training accuracy for each cut-off depth k. Based on your results, select an optimal k.

   Ans:
   Please check the source code and outputs included in the appendix named as
   CAP_5610_Assignment_1_Solution_Arman_Sayan.ipynb
   for the solution.

2. [8 points] Using the trained classifier with optimal cut-off depth k, classify the 99,762 instances from the test file and report the testing accuracy (the portion of testing instances classified correctly).

   Ans:
   Please check the source code and outputs included in the appendix named as
   CAP_5610_Assignment_1_Solution_Arman_Sayan.ipynb
   for the solution.

3. [7 points] Do you see any over-fitting issues for this experiment? Report your observations.

   Ans:
   Please check the source code and outputs included in the appendix named as
   CAP_5610_Assignment_1_Solution_Arman_Sayan.ipynb
   for the solution.

3 Independent Events and Bayes Theorem [20 points]

1. [5 points] For events A, B prove:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\neg A)P(\neg A)}$$

($\neg A$ denotes the event that A does not occur.)

Ans:

We need to prove Bayes' theorem in the expanded form. We know from Bayes' rule that the definition of conditional probability is

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

By a simple induction, we can write $P(A \cap B)$ as

$$P(A \cap B) = P(B|A)P(A)$$

Furthermore, using the law of total probability, we can express $P(B)$ as

$$P(B) = P(A \cap B) + P(\neg A \cap B)$$

where by using the above induction again, we reveal that

$$P(\neg A \cap B) = P(B|\neg A)P(\neg A)$$

Using these definitions and substituting them in the original theorem, we can prove that

$$\boldsymbol{P(A|B) = \frac{P(A \cap B)}{P(B)}}$$

$$\boldsymbol{P(A|B) = \frac{P(B|A)P(A)}{P(A \cap B) + P(\neg A \cap B)}}$$

$$\boldsymbol{P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\neg A)P(\neg A)}}$$

2. Let X, Y , and Z be random variables taking values in $0, 1$. The following table lists the probability of each possible assignment of 0 and 1 to the variables X, Y, and Z:

	$Z = 0$		$Z = 1$	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
$Y = 0$	0.1	0.05	0.1	0.1
$Y = 1$	0.2	0.1	0.175	0.175

i. [5 points] Is X independent of Y? Why or why not?

Ans:
To determine if X and Y are independent, we need to check whether the joint probability $P(X = x, Y = y)$ equals the product of the marginal probabilities $P(X = x) \cdot P(Y = y)$ for all possible values of x and y. If this holds true for all combinations, X and Y are independent. Otherwise, they are dependent.
As the first step, we need to calculate the marginal probabilities. The formula for marginal probability $P(X = x, Y = y)$ is given as

$P(X = x, Y = y) = P(X = x, Y = y, Z = 0) + P(X = x, Y = y, Z = 1)$
For each combination of X and Y, we can calculate the marginal probabilities as follows:

- $P(X = 0, Y = 0) = P(X = 0, Y = 0, Z = 0) + P(X = 0, Y = 0, Z = 1) = 0.1 + 0.1 = 0.2$
- $P(X = 1, Y = 0) = P(X = 1, Y = 0, Z = 0) + P(X = 1, Y = 0, Z = 1) = 0.05 + 0.1 = 0.15$
- $P(X = 0, Y = 1) = P(X = 0, Y = 1, Z = 0) + P(X = 0, Y = 1, Z = 1) = 0.2 + 0.175 = 0.375$
- $P(X = 1, Y = 1) = P(X = 1, Y = 1, Z = 0) + P(X = 1, Y = 1, Z = 1) = 0.1 + 0.175 = 0.275$

Now, we can calculate the marginal probabilities $P(X)$ and $P(Y)$ as follows:

- $P(X = 0) = P(X = 0, Y = 0) + P(X = 0, Y = 1) = 0.2 + 0.375 = 0.575$
- $P(X = 1) = P(X = 1, Y = 0) + P(X = 1, Y = 1) = 0.15 + 0.275 = 0.425$

- $P(Y=0) = P(X=0,Y=0) + P(X=1,Y=0) = 0.2 + 0.15 = 0.35$
- $P(Y=1) = P(X=0,Y=1) + P(X=1,Y=1) = 0.375 + 0.275 = 0.65$

The second step is to check the independce, which means for X and Y to be independent, the following must hold:

$$P(X=x, Y=y) = P(X=x) \cdot P(Y=y)$$

Check $P(X=0, Y=0)$:

$$P(X=0,Y=0) = 0.2$$
$$P(X=0) \cdot P(Y=0) = 0.575 \cdot 0.35 = 0.20125$$
$$\boldsymbol{P(X=0, Y=0) \neq P(X=0) \cdot P(Y=0)}$$

Check $P(X=1, Y=0)$:

$$P(X=1,Y=0) = 0.15$$
$$P(X=1) \cdot P(Y=0) = 0.425 \cdot 0.35 = 0.14875$$
$$\boldsymbol{P(X=1, Y=0) \neq P(X=1) \cdot P(Y=0)}$$

Check $P(X=0, Y=1)$:

$$P(X=0,Y=1) = 0.375$$
$$P(X=0) \cdot P(Y=1) = 0.575 \cdot 0.65 = 0.37375$$
$$\boldsymbol{P(X=0, Y=1) \neq P(X=0) \cdot P(Y=1)}$$

Check $P(X=1, Y=1)$:

$$P(X=1,Y=1) = 0.275$$
$$P(X=1) \cdot P(Y=1) = 0.425 \cdot 0.65 = 0.27625$$
$$\boldsymbol{P(X=1, Y=1) \neq P(X=1) \cdot P(Y=1)}$$

The values $P(X=x, Y=y)$ and $P(X=x)P(Y=y)$ are not exactly equal for all combinations. Therefore, ***X* and Y are not independent**.

ii. [5 points] Is X conditionally independent of Y given Z? Why or why not?

Ans:
To determine whether X is conditionally independent of Y given Z, we need to verify if the following holds for all values of X, Y, and Z:

$$P(X|Y,Z) = P(X|Z)$$

The formula for conditional probability $P(X|Y,Z)$ is given as

$$P(X|Y,Z) = \frac{P(X,Y,Z)}{P(Y,Z)}$$

The first step is to calculate the joint probabilities $P(X,Y,Z)$ for all combinations of X, Y, and Z. We can use the values from the table as it is.
The second step is to calculate the marginal probabilities $P(Y,Z)$ for all combinations of Y and Z.

- $P(Y=0,Z=0) = P(X=0,Y=0,Z=0) + P(X=1,Y=0,Z=0) = 0.1 + 0.05 = 0.15$
- $P(Y=1,Z=0) = P(X=0,Y=1,Z=0) + P(X=1,Y=1,Z=0) = 0.2 + 0.1 = 0.3$
- $P(Y=0,Z=1) = P(X=0,Y=0,Z=1) + P(X=1,Y=0,Z=1) = 0.1 + 0.1 = 0.2$
- $P(Y=1,Z=1) = P(X=0,Y=1,Z=1) + P(X=1,Y=1,Z=1) = 0.175 + 0.175 = 0.35$

The third step is to calculate the conditional probabilities $P(X|Y,Z)$ and $P(X|Z)$. We will have 2 cases to check.
The first case is when $Z=0$:
First, we compute the probabilities $P(Y,Z=0)$ which are obtained by summing over all X:

- $P(Y=0,Z=0) = P(X=0,Y=0,Z=0) + P(X=1,Y=0,Z=0) = 0.1 + 0.05 = 0.15$
- $P(Y=1,Z=0) = P(X=0,Y=1,Z=0) + P(X=1,Y=1,Z=0) = 0.2 + 0.1 = 0.3$

$$P(Z=0) = P(Y=0,Z=0) + P(Y=1,Z=0) = 0.15 + 0.3 = 0.45$$

Second, we compute the probabilities $P(X,Z=0)$ which are obtained by summing over all Y:

- $P(X = 0, Z = 0) = P(X = 0, Y = 0, Z = 0) + P(X = 0, Y = 1, Z = 0) = 0.1 + 0.2 = 0.3$
- $P(X = 1, Z = 0) = P(X = 1, Y = 0, Z = 0) + P(X = 1, Y = 1, Z = 0) = 0.05 + 0.1 = 0.15$

Third, we compute the probability $P(X|Z = 0)$:

- $P(X = 0|Z = 0) = \frac{P(X = 0, Z = 0)}{P(Z = 0)} = \frac{0.3}{0.45} = 0.666$
- $P(X = 1|Z = 0) = \frac{P(X = 1, Z = 0)}{P(Z = 0)} = \frac{0.15}{0.45} = 0.333$

Fourth, we compute the probability $P(X|Y, Z = 0)$:

- For $Y = 0$:

$$P(X = 0|Y = 0, Z = 0) = \frac{P(X = 0, Y = 0, Z = 0)}{P(Y = 0, Z = 0)} = \frac{0.1}{0.15} = 0.666$$
$$P(X = 1|Y = 0, Z = 0) = \frac{P(X = 1, Y = 0, Z = 0)}{P(Y = 0, Z = 0)} = \frac{0.05}{0.15} = 0.333$$

- For $Y = 1$:

$$P(X = 0|Y = 1, Z = 0) = \frac{P(X = 0, Y = 1, Z = 0)}{P(Y = 1, Z = 0)} = \frac{0.2}{0.3} = 0.666$$
$$P(X = 1|Y = 1, Z = 0) = \frac{P(X = 1, Y = 1, Z = 0)}{P(Y = 1, Z = 0)} = \frac{0.1}{0.3} = 0.333$$

Since $P(X|Y, Z = 0) = P(X|Z = 0)$ for all values of X, the condition holds for $Z = 0$.
The second case is when $Z = 1$:
First, we compute the probabilities $P(Y, Z = 1)$ which are obtained by summing over all X:

- $P(Y = 0, Z = 1) = P(X = 0, Y = 0, Z = 1) + P(X = 1, Y = 0, Z = 1) = 0.1 + 0.1 = 0.2$
- $P(Y = 1, Z = 1) = P(X = 0, Y = 1, Z = 1) + P(X = 1, Y = 1, Z = 1) = 0.175 + 0.175 = 0.35$

$$P(Z=1) = P(Y=0,Z=1) + P(Y=1,Z=1) = 0.2 + 0.35 = 0.55$$

Second, we compute the probabilities $P(X, Z=1)$ which are obtained by summing over all Y:

- $P(X=0,Z=1) = P(X=0,Y=0,Z=1) + P(X=0,Y=1,Z=1) = 0.1 + 0.175 = 0.275$
- $P(X=1,Z=1) = P(X=1,Y=0,Z=1) + P(X=1,Y=1,Z=1) = 0.1 + 0.175 = 0.275$

Third, we compute the probability $P(X|Z=1)$:

- $P(X=0|Z=1) = \dfrac{P(X=0,Z=1)}{P(Z=1)} = \dfrac{0.275}{0.55} = 0.5$
- $P(X=1|Z=1) = \dfrac{P(X=1,Z=1)}{P(Z=1)} = \dfrac{0.275}{0.55} = 0.5$

Fourth, we compute the probability $P(X|Y,Z=1)$:

- For $Y=0$:

$$P(X=0|Y=0,Z=1) = \frac{P(X=0,Y=0,Z=1)}{P(Y=0,Z=1)} = \frac{0.1}{0.2} = 0.5$$
$$P(X=1|Y=0,Z=1) = \frac{P(X=1,Y=0,Z=1)}{P(Y=0,Z=1)} = \frac{0.1}{0.2} = 0.5$$

- For $Y=1$:

$$P(X=0|Y=1,Z=1) = \frac{P(X=0,Y=1,Z=1)}{P(Y=1,Z=1)} = \frac{0.175}{0.35} = 0.5$$
$$P(X=1|Y=1,Z=1) = \frac{P(X=1,Y=1,Z=1)}{P(Y=1,Z=1)} = \frac{0.175}{0.35} = 0.5$$

Like in the first case, Since $P(X|Y,Z=1) = P(X|Z=1)$ for all values of X, the condition holds for $Z=1$.
For all values of Z, we proved that

$$P(X|Y,Z) = P(X|Z)$$

Thus, **X is conditionally independent of Y given Z**.

iii. [5 points] Calculate $P(X \neq Y|Z = 0)$.

Ans:
To compute $P(X \neq Y|Z = 0)$, we need to calculate the probability of events where $X \neq Y$, such as $X = 0$ and $Y = 1$, or $X = 1$ and $Y = 0$, conditioned on $Z = 0$.
We can calculate the probability as follows:

$$P(X \neq Y|Z = 0) = \frac{P(X \neq Y, Z = 0)}{P(Z = 0)}$$

Furthermore, we can calculate $P(X \neq Y, Z = 0)$ as
$P(X \neq Y, Z = 0) = P(X = 0, Y = 1, Z = 0) + P(X = 1, Y = 0, Z = 0)$
From the table, we can find the values as

- $P(X = 0, Y = 1, Z = 0) = 0.2$
- $P(X = 1, Y = 0, Z = 0) = 0.05$

$$P(X \neq Y, Z = 0) = 0.2 + 0.05 = 0.25$$

In addition, we need to calculate $P(Z = 0)$ by summing all joint probabilities where $Z = 0$:

$$\begin{aligned} P(Z = 0) &= P(X = 0, Y = 0, Z = 0) + P(X = 1, Y = 0, Z = 0) \\ &+P(X = 0, Y = 1, Z = 0) + P(X = 1, Y = 1, Z = 0) \\ &= 0.1 + 0.05 + 0.2 + 0.1 = 0.45 \end{aligned}$$

Finally, by substituting the values, we can calculate $P(X \neq Y|Z = 0)$ as

$$\boldsymbol{P(X \neq Y|Z = 0) = \frac{0.25}{0.45} \approx 0.5556}$$

4 Implementing Naive Bayes [25 points]

You will now learn how to use Naive Bayes Algorithm to solve a real-world problem: text cat- egorization. Text categorization (also referred to as text classification) i s t he t ask o f assigning documents to one or more topics. For our homework, we will use a benchmark dataset that is fre- quently used in text categorization problems. This dataset, Reuters-21578, consists of documents that appeared in Reuters newswire in 1987. Each document was then manually categorized into a topic among over 100 topics. In this homework, we are only interested in earn and acquisition (acq) topics, so we will use a shortened version of the dataset (documents assigned to topics other than "earn" or "acq" are not in the dataset provided for the homework). As features, we will use the frequency (counts) of each word that occurred in the document. This model is known as the bag-of-words model and it is frequently used in text categorization. You can download Assignment 2 data from the Canvas. In this folder, you will find:

- **train.csv:** Training data. Each row represents a document, and each column separated by commas represents features (word counts). There are 4527 documents and 5180 words.
- **train labels.txt:** labels for the training data
- **test.csv:** Test data, 1806 documents and 5180 words

Implement Naive Bayes Algorithm. Train your classifier on the training set that is given and report training accuracy, testing accuracy, and the amount of time spent training the classifier.

Ans:

Please check the source code and outputs included in the appendix named as

CAP_5610_Assignment_1_Solution_Arman_Sayan.ipynb

for the solution.

A Appendix

CAP_5610_Assignment_1_Solution_Arman_Sayan

January 26, 2025

CAP 5610 Assignment #1: Decision Tree and Naive Bayes Classifier

This source code is written by Arman Sayan.

Last Edit: January 26, 2024

1 Q1 - Decision Tree Basics

1.1 Part (3):

[1]:
```
import pandas as pd
import numpy as np
from math import log2
from collections import Counter
from graphviz import Digraph

# Data setup
data = [
    [1, 1, 0, 0, 1, 1],
    [1, 1, 1, 0, 1, 1],
    [0, 0, 1, 0, 0, 0],
    [0, 1, 1, 0, 1, 0],
    [0, 1, 1, 0, 0, 1],
    [0, 0, 1, 1, 1, 1],
    [1, 0, 0, 0, 1, 0],
    [0, 1, 0, 1, 1, 1],
    [0, 0, 1, 0, 1, 1],
    [1, 0, 0, 0, 0, 0],
    [1, 1, 1, 0, 0, 1],
    [0, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 1, 0],
    [1, 0, 0, 1, 0, 1],
]
columns = ["Early", "Finished HMK", "Senior", "Likes Coffee", "Liked The Last␣
 ↪Jedi", "A"]
df = pd.DataFrame(data, columns=columns)

# Entropy calculation
```

```
def entropy(labels):
    total = len(labels)
    counts = Counter(labels)
    return -sum((count / total) * log2(count / total) for count in counts.
 ↪values() if count > 0)

# Information gain calculation
def information_gain(df, attribute, target="A"):
    total_entropy = entropy(df[target])
    values = df[attribute].unique()
    weighted_entropy = sum(
        (len(subset) / len(df)) * entropy(subset[target])
        for value in values
        if (subset := df[df[attribute] == value]) is not None
    )
    return total_entropy - weighted_entropy

# ID3 algorithm
def id3(df, attributes, target="A", depth=1):
    node_entropy = entropy(df[target])
    if depth == 0 or node_entropy == 0 or len(attributes) == 0:
        most_common_label = Counter(df[target]).most_common(1)[0][0]
        return {"label": most_common_label, "count": len(df), "entropy":␣
 ↪node_entropy}

    best_attribute = max(attributes, key=lambda attr: information_gain(df,␣
 ↪attr, target))
    tree = {"attribute": best_attribute, "entropy": node_entropy, "children":␣
 ↪{}}

    for value in df[best_attribute].unique():
        subset = df[df[best_attribute] == value]
        if len(subset[target].unique()) == 1:
            label = subset[target].iloc[0]
            tree["children"][value] = {"label": label, "count": len(subset),␣
 ↪"entropy": entropy(subset[target])}
        else:
            remaining_attributes = [attr for attr in attributes if attr !=␣
 ↪best_attribute]
            tree["children"][value] = id3(subset, remaining_attributes, target,␣
 ↪depth - 1)

    return tree

# Visualize the decision tree
def visualize_tree(tree, graph=None, parent=None, edge_label=None):
```

```
    if graph is None:
        graph = Digraph(format="png")
        graph.attr("node", shape="box")

    if "label" in tree:
        node_label = f"Leaf: {tree['label']}\nCount: {tree['count']}\nEntropy:␣
↪{tree['entropy']:.4f}"
        node_id = str(id(tree))
        graph.node(node_id, label=node_label)
        if parent:
            graph.edge(parent, node_id, label=edge_label)
    else:
        node_label = f"{tree['attribute']}\nEntropy: {tree['entropy']:.4f}"
        node_id = str(id(tree))
        graph.node(node_id, label=node_label)
        if parent:
            graph.edge(parent, node_id, label=edge_label)

        for value, child in tree["children"].items():
            visualize_tree(child, graph, parent=node_id, edge_label=str(value))

    return graph

# Build trees
depth_1_tree = id3(df, columns[:-1], depth=1)
depth_2_tree = id3(df, columns[:-1], depth=2)

# Save visualizations
visualize_tree(depth_1_tree).render("depth_1_tree", cleanup=True)
visualize_tree(depth_2_tree).render("depth_2_tree", cleanup=True)
```

[1]: 'depth_2_tree.png'

1.2 Part (4):

[2]:
```
# Build a depth-3 tree
depth_3_tree = id3(df, columns[:-1], depth=3)

# Save visualization
visualize_tree(depth_3_tree).render("depth_3_tree", cleanup=True)
```

[2]: 'depth_3_tree.png'

[3]:
```
def calculate_average_leaf_entropy(tree):
    def collect_leaf_entropies(node):
        if "label" in node:  # Leaf node
```

```
            return [node["entropy"]]
        # Recursive collection of entropies from children
        entropies = []
        for child in node["children"].values():
            entropies.extend(collect_leaf_entropies(child))
        return entropies

    leaf_entropies = collect_leaf_entropies(tree)
    return sum(leaf_entropies) / len(leaf_entropies) if leaf_entropies else 0
```

[4]:
```
avg_entropy_depth_1 = calculate_average_leaf_entropy(depth_1_tree)
avg_entropy_depth_2 = calculate_average_leaf_entropy(depth_2_tree)

print(f"Average leaf entropy for depth-1 tree: {avg_entropy_depth_1:.4f}")
print(f"Average leaf entropy for depth-2 tree: {avg_entropy_depth_2:.4f}")
```

```
Average leaf entropy for depth-1 tree: 0.9242
Average leaf entropy for depth-2 tree: 0.4305
```

[5]:
```
# Build and visualize depth-3 tree
depth_3_tree = id3(df, columns[:-1], depth=3)
visualize_tree(depth_3_tree).render("depth_3_tree", cleanup=True)

# Calculate average leaf entropy for depth-3 tree
avg_entropy_depth_3 = calculate_average_leaf_entropy(depth_3_tree)
print(f"Average leaf entropy for depth-3 tree: {avg_entropy_depth_3:.4f}")
```

```
Average leaf entropy for depth-3 tree: 0.3197
```

2 Q2 - Application of Decision Tree on Real-Word Data-set

2.1 Check Statistics for the Census-Income Data Set:

[6]:
```
import pandas as pd

# Load the dataset

column_names = [
"AAGE",
"ACLSWKR",
"ADTIND",
"ADTOCC",
"AHGA",
"AHRSPAY",
"AHSCOL",
"AMARITL",
"AMJIND",
```

```
"AMJOCC",
"ARACE",
"AREORGN",
"ASEX",
"AUNMEM",
"AUNTYPE",
"AWKSTAT",
"CAPGAIN",
"CAPLOSS",
"DIVVAL",
"FILESTAT",
"GRINREG",
"GRINST",
"HHDFMX",
"HHDREL",
"MARSUPWT",
"MIGMTR1",
"MIGMTR3",
"MIGMTR4",
"MIGSAME",
"MIGSUN",
"NOEMP",
"PARENT",
"PEFNTVTY",
"PEMNTVTY",
"PENATVTY",
"PRCITSHP",
"SEOTR",
"VETQVA",
"VETYN",
"WKSWORK",
"YEAR",
"INCCLS"]

data = pd.read_csv("census-income.data", header=None, names=column_names)
test = pd.read_csv("census-income.test", header=None, names=column_names)
```

[7]:
```
data.head(5)
```

[7]:
```
   AAGE                        ACLSWKR  ADTIND  ADTOCC  \
0    73                Not in universe       0       0
1    58  Self-employed-not incorporated       4      34
2    18                Not in universe       0       0
3     9                Not in universe       0       0
4    10                Not in universe       0       0

                          AHGA  AHRSPAY            AHSCOL         AMARITL  \
```

```
0         High school graduate        0   Not in universe         Widowed
1   Some college but no degree        0   Not in universe        Divorced
2                   10th grade        0       High school   Never married
3                     Children        0   Not in universe   Never married
4                     Children        0   Not in universe   Never married

                         AMJIND                                AMJOCC  …  \
0   Not in universe or children                        Not in universe  …
1                 Construction   Precision production craft & repair  …
2   Not in universe or children                        Not in universe  …
3   Not in universe or children                        Not in universe  …
4   Not in universe or children                        Not in universe  …

         PEFNTVTY        PEMNTVTY        PENATVTY  \
0   United-States   United-States   United-States
1   United-States   United-States   United-States
2         Vietnam         Vietnam         Vietnam
3   United-States   United-States   United-States
4   United-States   United-States   United-States

                               PRCITSHP SEOTR            VETQVA  VETYN  \
0     Native- Born in the United States     0   Not in universe      2
1     Native- Born in the United States     0   Not in universe      2
2   Foreign born- Not a citizen of U S      0   Not in universe      2
3     Native- Born in the United States     0   Not in universe      0
4     Native- Born in the United States     0   Not in universe      0

   WKSWORK  YEAR     INCCLS
0        0    95   - 50000.
1       52    94   - 50000.
2        0    95   - 50000.
3        0    94   - 50000.
4        0    94   - 50000.

[5 rows x 42 columns]
```

[8]:
```
test.head(5)
```

[8]:
```
    AAGE                         ACLSWKR  ADTIND  ADTOCC  \
0     38                         Private       6      36
1     44   Self-employed-not incorporated      37      12
2      2                 Not in universe       0       0
3     35                         Private      29       3
4     49                         Private       4      34

                                  AHGA  AHRSPAY            AHSCOL  \
0             1st 2nd 3rd or 4th grade        0   Not in universe
```

```
1   Associates degree-occup /vocational        0   Not in universe
2                              Children        0   Not in universe
3                  High school graduate        0   Not in universe
4                  High school graduate        0   Not in universe

                            AMARITL                          AMJIND  \
0   Married-civilian spouse present    Manufacturing-durable goods
1   Married-civilian spouse present   Business and repair services
2                     Never married    Not in universe or children
3                          Divorced                 Transportation
4                          Divorced                   Construction

                                  AMJOCC  …        PEFNTVTY  \
0   Machine operators assmblrs & inspctrs  …          Mexico
1                 Professional specialty  …   United-States
2                        Not in universe  …   United-States
3         Executive admin and managerial  …   United-States
4     Precision production craft & repair  …   United-States

         PEMNTVTY        PENATVTY                              PRCITSHP SEOTR  \
0          Mexico          Mexico   Foreign born- Not a citizen of U S      0
1   United-States   United-States     Native- Born in the United States     0
2   United-States   United-States     Native- Born in the United States     0
3   United-States   United-States     Native- Born in the United States     2
4   United-States   United-States     Native- Born in the United States     0

             VETQVA  VETYN  WKSWORK  YEAR     INCCLS
0   Not in universe      2       12    95   - 50000.
1   Not in universe      2       26    95   - 50000.
2   Not in universe      0        0    95   - 50000.
3   Not in universe      2       52    94   - 50000.
4   Not in universe      2       50    95   - 50000.

[5 rows x 42 columns]
```

[9]:
```python
# Print the number of instances
num_instances_data = data.shape[0]
print(f"Number of instances in data: {num_instances_data}")

num_instances_test = test.shape[0]
print(f"Number of instances in test: {num_instances_test}")
```

```
Number of instances in data: 199523
Number of instances in test: 99762
```

[10]:
```python
# Calculate the probability distribution of the 'income' column
income_probabilities = test["INCCLS"].value_counts(normalize=True)
```

```
# Print the probabilities
print("Class probabilities for income-projected.test file:")
print(income_probabilities)
```

```
Class probabilities for income-projected.test file:
INCCLS
- 50000.    0.937992
50000+.     0.062008
Name: proportion, dtype: float64
```

[11]:
```
print("Information about .data file:")
# Calculate the number of distinct values for each column
distinct_values_count = data.nunique()

# Display the results
print("Number of distinct values for each column:")
print(distinct_values_count)
```

```
Information about .data file:
Number of distinct values for each column:
AAGE           91
ACLSWKR         9
ADTIND         52
ADTOCC         47
AHGA           17
AHRSPAY      1240
AHSCOL          3
AMARITL         7
AMJIND         24
AMJOCC         15
ARACE           5
AREORGN        10
ASEX            2
AUNMEM          3
AUNTYPE         6
AWKSTAT         8
CAPGAIN       132
CAPLOSS       113
DIVVAL       1478
FILESTAT        6
GRINREG         6
GRINST         51
HHDFMX         38
HHDREL          8
MARSUPWT    99800
MIGMTR1        10
MIGMTR3         9
```

```
MIGMTR4        10
MIGSAME         3
MIGSUN          4
NOEMP           7
PARENT          5
PEFNTVTY       43
PEMNTVTY       43
PENATVTY       43
PRCITSHP        5
SEOTR           3
VETQVA          3
VETYN           3
WKSWORK        53
YEAR            2
INCCLS          2
dtype: int64
```

2.2 Task (a):

Train a decision tree classifier using the data file. Vary the cut-off depth from 2 to 10 and report the training accuracy for each cut-off depth k. Based on your results, select an optimal k.

[12]:
```python
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Preprocess the data
# Encode categorical variables
label_encoders = {}
for column in data.select_dtypes(include=["object"]).columns:
    label_encoders[column] = LabelEncoder()
    data[column] = label_encoders[column].fit_transform(data[column])
```

[13]:
```python
# Separate features and target variable
X = data.drop(["INCCLS", "MARSUPWT"], axis=1)
y = data["INCCLS"]
```

[14]:
```python
# Train decision trees with varying depths
training_accuracies = []
depths = range(2, 11)

for depth in depths:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X, y)

    # Compute training accuracy
```

```
    y_train_pred = clf.predict(X)
    train_acc = accuracy_score(y, y_train_pred)
    training_accuracies.append(train_acc)
```

[15]:
```
# Report results
for depth, train_acc in zip(depths, training_accuracies):
    print(f"Depth: {depth}, Training Accuracy: {train_acc:.4f}")

# Select the optimal depth based on testing accuracy
optimal_depth = depths[np.argmax(training_accuracies)]
print(f"\nOptimal Depth: {optimal_depth}")
```

```
Depth: 2, Training Accuracy: 0.9445
Depth: 3, Training Accuracy: 0.9449
Depth: 4, Training Accuracy: 0.9455
Depth: 5, Training Accuracy: 0.9492
Depth: 6, Training Accuracy: 0.9498
Depth: 7, Training Accuracy: 0.9520
Depth: 8, Training Accuracy: 0.9525
Depth: 9, Training Accuracy: 0.9535
Depth: 10, Training Accuracy: 0.9544

Optimal Depth: 10
```

2.3 Task (b)

Using the trained classifier with optimal cut-off depth k, classify the 99,762 instances from the test file and report the testing accuracy (the portion of testing instances classified correctly).

[16]:
```
# Preprocess the test data
# Apply the same LabelEncoders used for training
label_encoders_test = {}
for column in test.select_dtypes(include=["object"]).columns:
    label_encoders_test[column] = LabelEncoder()
    test[column] = label_encoders_test[column].fit_transform(test[column])
```

[17]:
```
# Separate features and target variable in the test data
X_test = test.drop(["INCCLS", "MARSUPWT"], axis=1)
y_test = test["INCCLS"]
```

[18]:
```
# Train a classifier with the optimal depth
clf_optimal = DecisionTreeClassifier(max_depth=optimal_depth, random_state=42)
clf_optimal.fit(X, y)

# Classify the test instances
y_pred_final = clf_optimal.predict(X_test)

# Compute testing accuracy
```

```
final_accuracy = accuracy_score(y_test, y_pred_final)

# Report the results
print(f"Testing Accuracy on the test file: {final_accuracy:.4f}")
```

```
Testing Accuracy on the test file: 0.9509
```

2.4 Task (c)

Do you see any over-fitting issues for this experiment? Report your observations.

[19]:
```
# Train decision trees with varying depths
training_accuracies = []
testing_accuracies = []
depths = range(2, 11)

for depth in depths:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X, y)

    # Compute training accuracy
    y_train_pred = clf.predict(X)
    train_acc = accuracy_score(y, y_train_pred)
    training_accuracies.append(train_acc)

    # Compute testing accuracy
    y_test_pred = clf.predict(X_test)
    test_acc = accuracy_score(y_test, y_test_pred)
    testing_accuracies.append(test_acc)

# Report results
for depth, train_acc, test_acc in zip(depths, training_accuracies,␣
 ↪testing_accuracies):
    print(f"Depth: {depth}, Training Accuracy: {train_acc:.4f}, Testing␣
 ↪Accuracy: {test_acc:.4f}")

# Select the optimal depth based on testing accuracy
optimal_depth = depths[np.argmax(testing_accuracies)]
print(f"\nOptimal Depth: {optimal_depth}")
```

```
Depth: 2, Training Accuracy: 0.9445, Testing Accuracy: 0.9442
Depth: 3, Training Accuracy: 0.9449, Testing Accuracy: 0.9447
Depth: 4, Training Accuracy: 0.9455, Testing Accuracy: 0.9447
Depth: 5, Training Accuracy: 0.9492, Testing Accuracy: 0.9485
Depth: 6, Training Accuracy: 0.9498, Testing Accuracy: 0.9486
Depth: 7, Training Accuracy: 0.9520, Testing Accuracy: 0.9505
Depth: 8, Training Accuracy: 0.9525, Testing Accuracy: 0.9507
Depth: 9, Training Accuracy: 0.9535, Testing Accuracy: 0.9505
```

```
Depth: 10, Training Accuracy: 0.9544, Testing Accuracy: 0.9509

Optimal Depth: 10
```

From the data provided, there **does not appear to be a significant overfitting issue**, as both the **training accuracy** and **testing accuracy** are increasing or remaining stable as the tree depth increases.

2.4.1 Observations:

1. **Consistency between Training and Testing Accuracy**:
   - The testing accuracy does not decrease as the depth increases, which is a hallmark of overfitting. Instead, the testing accuracy either slightly increases or plateaus, indicating that the model is still generalizing well even at higher depths.
   - The gap between training and testing accuracy remains small (less than ~0.004), which is minimal.
2. **Optimal Depth**:
   - The testing accuracy peaks at **Depth = 10** (Testing Accuracy = 0.9509). However, the improvement from Depth = 8 to Depth = 10 is very marginal (0.9507 → 0.9509), and increasing the depth further may result in diminishing returns.
   - Depth = 8 or Depth = 10 can be considered the optimal depth based on the goal (e.g., achieving maximum accuracy or reducing computational complexity).
3. **Overfitting Behavior**:
   - While training accuracy increases more quickly with depth, this is expected for deeper decision trees as they capture more details in the data. However, the testing accuracy keeps up, suggesting the deeper trees are still generalizing well.
   - There is no clear overfitting in this experiment.

---

3 Q4 - Implementing Naive Bayes

Implement Naive Bayes Algorithm. Train your classifier on the training set that is given and report training accuracy, testing accuracy, and the amount of time spent training the classifier.

[20]:
```python
import pandas as pd
import numpy as np
import time
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Load data
def load_data(train_file, train_labels_file, test_file, test_labels_file):
    X_train = pd.read_csv(train_file, header=None).values
    y_train = pd.read_csv(train_labels_file, header=None).values.ravel()
    X_test = pd.read_csv(test_file, header=None).values
    y_test = pd.read_csv(test_labels_file, header=None).values.ravel()
    return X_train, y_train, X_test, y_test
```

```python
# Train and evaluate Naive Bayes classifier
def train_naive_bayes(X_train, y_train, X_test, y_test):
    print(f"Missing values in X_train: {np.isnan(X_train).sum()}")
    print(f"Missing values in X_test: {np.isnan(X_test).sum()}")

    start_time = time.time()

    # Initialize and train the classifier
    nb_classifier = MultinomialNB()
    nb_classifier.fit(X_train, y_train)

    # Measure training time
    training_time = time.time() - start_time

    # Predict on training and test sets
    y_train_pred = nb_classifier.predict(X_train)
    y_test_pred = nb_classifier.predict(X_test)

    # Calculate accuracies
    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    return train_accuracy, test_accuracy, training_time

# Main function to load data, train classifier, and report results
def main():
    # File paths (update these paths if necessary)
    train_file = "train.csv"
    train_labels_file = "train_labels.txt"
    test_file = "test.csv"
    test_labels_file = "test_labels.txt"

    # Load the data
    X_train, y_train, X_test, y_test = load_data(train_file, train_labels_file,␣
 ↪test_file, test_labels_file)

    # Train Naive Bayes and get metrics
    train_accuracy, test_accuracy, training_time = train_naive_bayes(X_train,␣
 ↪y_train, X_test, y_test)

    # Print results
    print(f"Training Accuracy: {train_accuracy:.4f}")
    print(f"Testing Accuracy: {test_accuracy:.4f}")
    print(f"Training Time: {training_time:.4f} seconds")

if __name__ == "__main__":
    main()
```

```
Missing values in X_train: 0
Missing values in X_test: 0
Training Accuracy: 0.9693
Testing Accuracy: 0.9823
Training Time: 0.0835 seconds
```