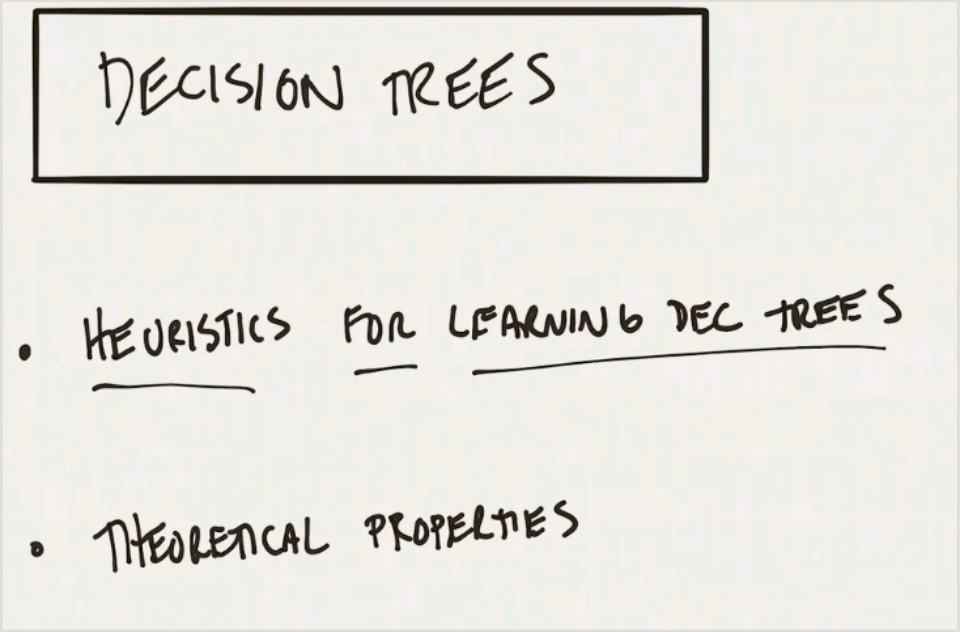
1.2 Decision Trees



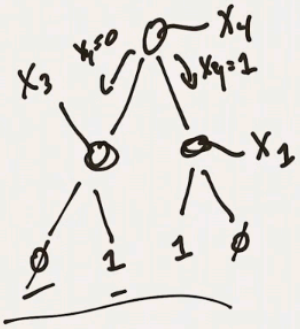
Heuristics : enabling someone to discover or learn something for themselves.

DECISION TREE IS A BOOLEAN FUNCTION

(OUTPUTS +1 or -1
0 or 1)

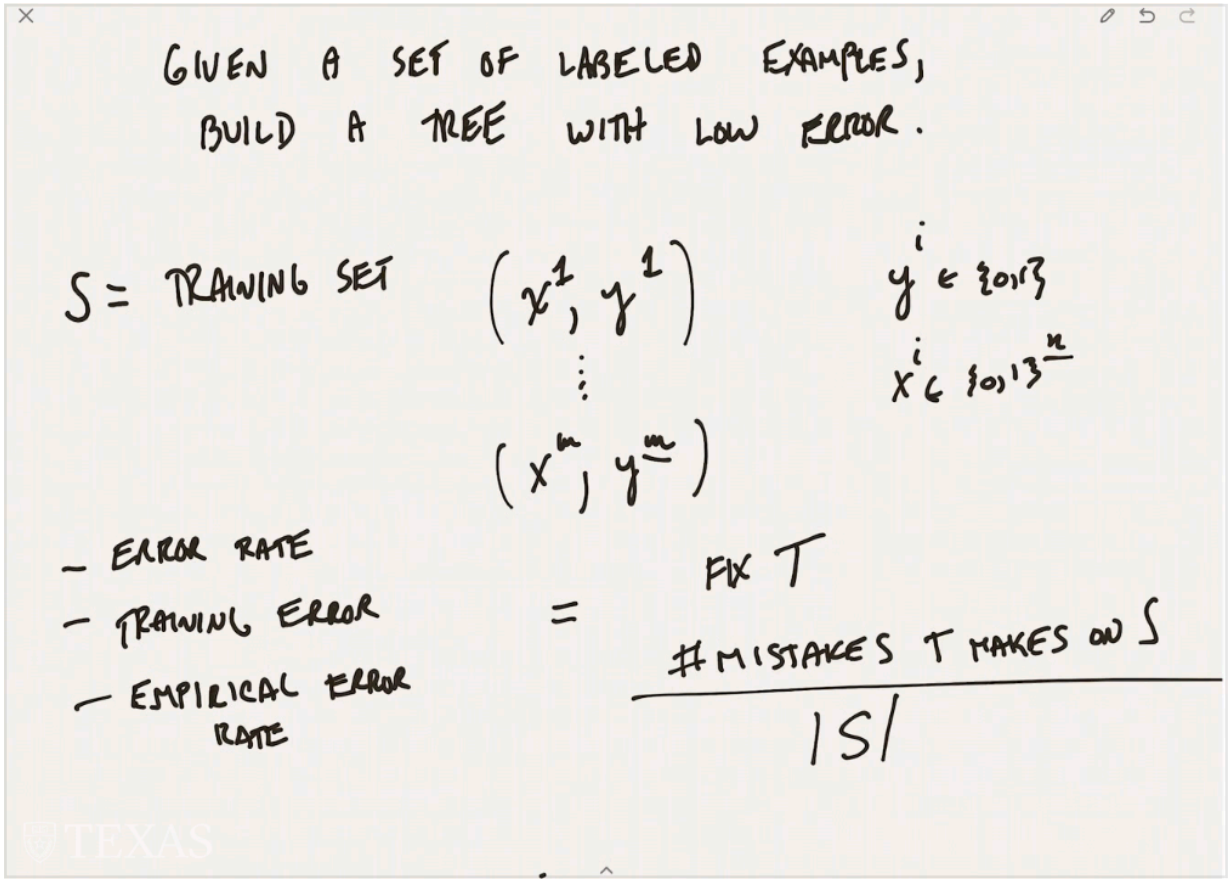
INPUT:

$x \in \{0,1\}^n$ $\quad f(x) \to \{0,1\}$



SIZE OF DECISION TREE = # OF NODES IN THE TREE

DEPTH OF TREE = LENGTH OF LONGEST PATH FROM ROOT TO A LEAF.



The notation $|S|$ represents the **size (cardinality) of the training set** S—meaning the total number of training examples in the dataset.

Breaking it Down:

- S is the training set:

$$S = \{(x^1, y^1), (x^2, y^2), ..., (x^m, y^m)\}$$

- $|S|$ simply counts how many such examples exist, i.e., $|S| = m$.

Where It's Used in the Formula:

The **training error** (or **empirical error rate**) is calculated as:

$$\frac{\#\text{ of mistakes } T \text{ makes on } S}{|S|}$$

This formula measures how many errors the decision tree T makes on the training set, normalized by the total number of training samples. This gives the **proportion of misclassified examples**, which is the empirical error.

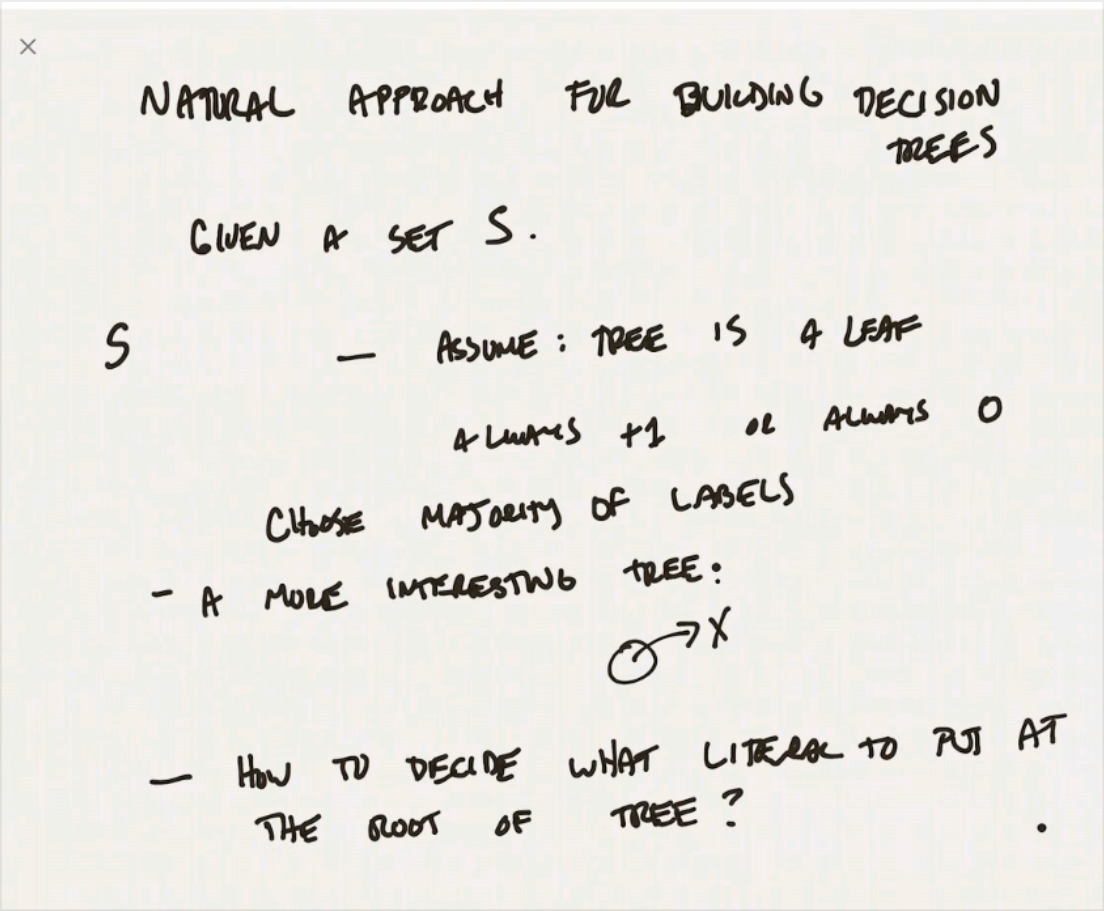
Example:

Suppose you have a training set with $|S| = 100$ examples.

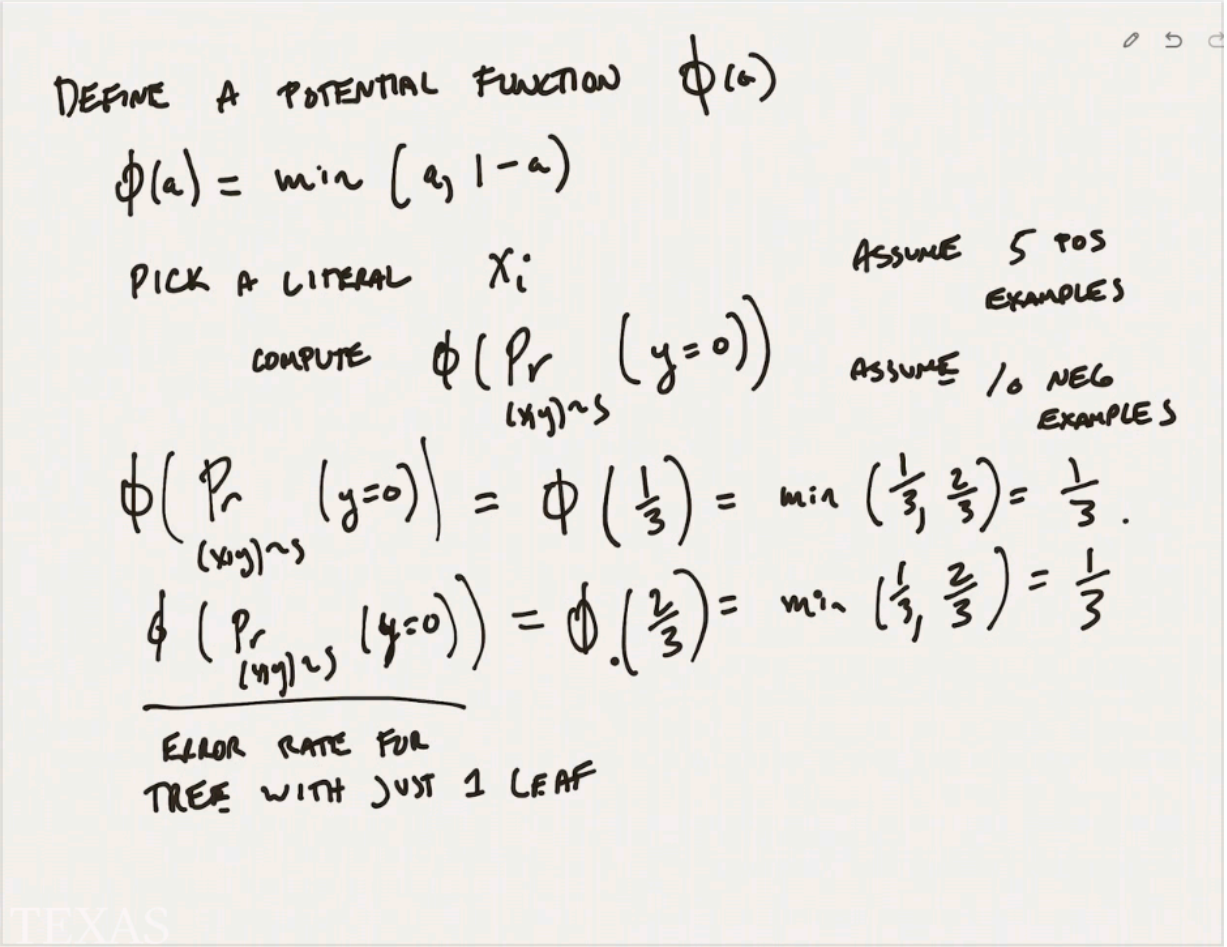
- If your decision tree makes **12 mistakes** on this dataset, the empirical error rate would be:

$$\frac{12}{100} = 0.12 \quad (12\% \text{ error rate})$$

Would you like a further breakdown on how this is used in decision trees? 🚀



Where a is set of examples which fulfill the certain criteria
1-a is excluding what left which didn't fulfil the criteria

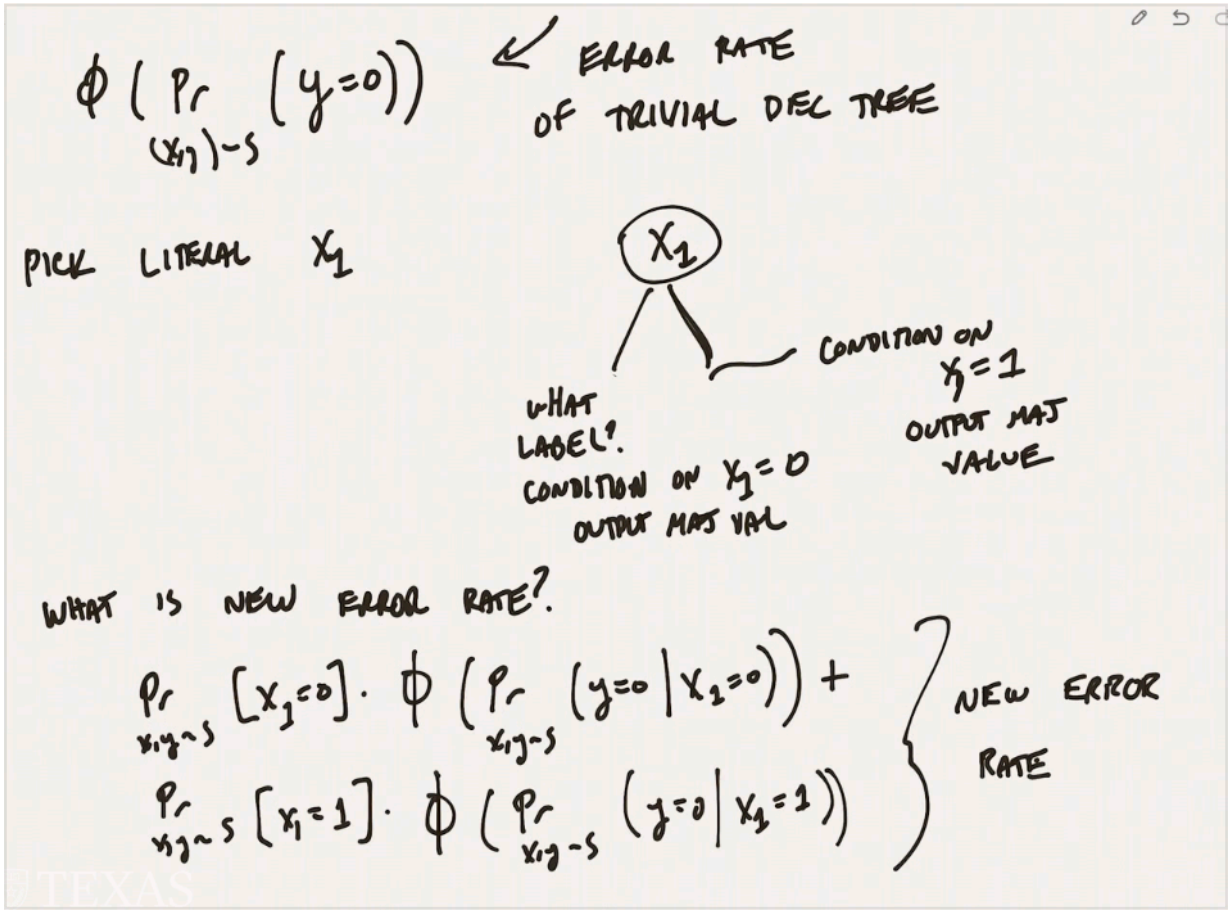


Step 1: Understanding $\Phi_1(a) = \min(a, 1 - a)$

This represents the **error rate of a simple decision tree split** where we assign labels based on the majority class in each subset.

- Suppose we split data into two groups using a feature.
- If **a** is the proportion of one class (e.g., spam emails) in the left node, then **1 - a** is the proportion of the other class (not spam).
- The best possible prediction is to **assign the majority class** as the output.
- So, the **error in each node** is the minority class proportion, which is **min(a, 1 - a)**.

💡 **Intuition:** If a split gives 80% spam (a=0.8), the best we can do is classify everything as spam. The error will be **20% (1 - 0.8)**, which is the smaller value between (a, 1 - a).



Imagine you are a **teacher** in a classroom with **10 students**. You have to guess whether each student **likes chocolate** 🍫 **or not**.

Step 1: Before Splitting (One Big Group)

Right now, you don't have any information about the students, so you guess the **most common answer** (majority).

Let's say:

- **6 students like chocolate (Yes)** ✅
- **4 students don't like chocolate (No)** ❌

Since most students like chocolate, you guess **"Yes" for everyone**.

But oops! You made mistakes for the **4 students who actually don't like chocolate**. So, your **error rate is**:

$$\frac{4}{10} = 40\%$$

Step 2: Splitting the Students Based on a Rule (Feature X_1)

Now, you decide to split students based on **a new rule**:
"Do they like ice cream 🍦?" (Let's call this rule X_1)

- **5 students like ice cream (Group 1)** 🍦
  - In this group, 4 like chocolate ✅
  - 1 does not ❌
- **5 students don't like ice cream (Group 2)** 🚫🍦
  - In this group, 2 like chocolate ✅
  - 3 do not ❌

Now, in each group, you again guess the **majority**:

- **Group 1**: Most students like chocolate → You guess "Yes" ✅
  - Mistake? **Only 1 mistake!** (1 student who actually doesn't like chocolate)
- **Group 2**: Most students don't like chocolate → You guess "No" ❌
  - Mistake? **Only 2 mistakes!** (2 students who actually like chocolate)

Now, instead of **one big group making 4 mistakes**, we have **two smaller groups making fewer mistakes**.

The **new error rate formula** is:

$$\text{New Error Rate} = P(X_1 = 0) \times \text{Error in Group 2} + P(X_1 = 1) \times \text{Error in Group 1}$$

Using our example:

$$\left(\frac{5}{10} \times \frac{2}{5}\right) + \left(\frac{5}{10} \times \frac{1}{5}\right)$$

$$= 0.5 \times 0.4 + 0.5 \times 0.2$$

= 0.2 + 0.1 = 0.3 \text{ (30% error)}

Step 4: Compare Error Rates

- **Before Split**: **40% errors**
- **After Split**: **30% errors (less mistakes!)** 🎉

Since the error **decreased**, this means our rule X_1 (ice cream preference) **helped us make better guesses!** 🏆

Real-World Example: Email Spam Detection 📧

Let's say you work at a company that wants to **classify emails as Spam (1) or Not Spam (0)** using a decision tree.

Step 1: Initial Dataset (Before Splitting)

Email	Contains "Free Money" (X_1)	Spam (Y)
A	1 (Yes)	1 (Spam)
B	0 (No)	0 (Not Spam)
C	1 (Yes)	1 (Spam)
D	0 (No)	0 (Not Spam)
E	1 (Yes)	1 (Spam)
F	0 (No)	0 (Not Spam)
G	1 (Yes)	1 (Spam)
H	1 (Yes)	1 (Spam)
I	0 (No)	0 (Not Spam)
J	0 (No)	0 (Not Spam)

- **Majority Class Before Splitting**:
  - 5 Spam (1)
  - 5 Not Spam (0)
  - **Tie!**

🔷 **Error Rate Before Splitting**: If we assume **Not Spam (0) as the majority class**, we misclassify all **5 spam emails**.

$$\frac{5}{10} = 50\%$$

Step 2: Splitting Based on “Contains 'Free Money'” (X_1)

We split emails based on whether they **contain the phrase "Free Money."**

Group 1: Emails that contain "Free Money" ($X_1 = 1$)

Email	X_1 (Contains "Free Money")	Spam (Y)
A	1 (Yes)	1 (Spam)
C	1 (Yes)	1 (Spam)
E	1 (Yes)	1 (Spam)
G	1 (Yes)	1 (Spam)
H	1 (Yes)	1 (Spam)

- **Majority Label**: All are **Spam (1)** ✅
- **Errors**: **0 mistakes** 🎉

Group 2: Emails that do NOT contain "Free Money" ($X_1 = 0$)

Email	X_1 (Contains "Free Money")	Spam (Y)
B	0 (No)	0 (Not Spam)
D	0 (No)	0 (Not Spam)
F	0 (No)	0 (Not Spam)
I	0 (No)	0 (Not Spam)
J	0 (No)	0 (Not Spam)

- **Majority Label**: All are **Not Spam (0)** ✅
- **Errors**: **0 mistakes** 🎉

Step 3: Compute New Error Rate

Using the formula:

$$\text{New Error Rate} = P(X_1 = 0) \times \text{Error in Group 2} + P(X_1 = 1) \times \text{Error in Group 1}$$

$$= \left(\frac{5}{10} \times 0\right) + \left(\frac{5}{10} \times 0\right) = 0\%$$

✅ **The error rate dropped from 50% to 0% after splitting!**

Why This Matters in Production?

- Decision trees **help classify spam emails** automatically.
- This example shows how adding **one good feature ("Free Money")** can reduce errors significantly.
- In real-world scenarios, more features like **sender reputation, number of links, and message length** help further improve accuracy.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Sample dataset
data = {
```

```
    'Contains_Free_Money': [1, 0, 1, 0, 1, 0, 1, 1, 0, 0],  # Feature X1
    'Spam': [1, 0, 1, 0, 1, 0, 1, 1, 0, 0]  # Target Y
}

df = pd×DataFrame(data)

# Split features (X) and target (y)
X = df['Contains_Free_Money']
y = df['Spam']

# Train a simple Decision Tree
clf = DecisionTreeClassifier(criterion='gini', max_depth=1)  # Single split (depth=1)
clf.fit(X, y)

y_pred = clf×predict(X)
accuracy = accuracy_score(y, y_pred)
error_rate = 1 - accuracy

# Print results
print("Predictions:", y_pred)
print("Accuracy:", accuracy)
print("Error Rate before split: 50% (assuming majority class) ")
print(f"Error Rate after split: {error_rate * 100:.2f}%")
```

The criterion='gini' in the **DecisionTreeClassifier** specifies the **splitting criterion** for the decision tree.

The `criterion='gini'` in the **DecisionTreeClassifier** specifies the **splitting criterion** for the decision tree.

What Does It Do?

- It tells the decision tree **how to measure impurity** (how mixed the labels are) when making a split.
- Lower impurity → **Better split** → More pure groups (all spam or all not spam).

Example: Classifying Emails as Spam or Not Spam

Imagine you have a dataset where emails are classified as **Spam (1)** or **Not Spam (0)**.

High Impurity (Mixed Labels)

Consider a node that contains 10 emails:

Email	Label
Email 1	Spam (1)
Email 2	Not Spam (0)
Email 3	Spam (1)
Email 4	Not Spam (0)
Email 5	Spam (1)
Email 6	Not Spam (0)
Email 7	Spam (1)
Email 8	Not Spam (0)
Email 9	Spam (1)
Email 10	Not Spam (0)

- Here, **5 emails are Spam (1) and 5 are Not Spam (0)**.
- The impurity is **high** because the classes are evenly mixed.

Lower Impurity (More Pure Groups)

Now, consider another node with the following labels:

Email	Label
Email 1	Spam (1)
Email 2	Spam (1)
Email 3	Spam (1)
Email 4	Spam (1)
Email 5	Spam (1)
Email 6	Spam (1)
Email 7	Not Spam (0)
Email 8	Not Spam (0)
Email 9	Not Spam (0)
Email 10	Not Spam (0)

- Here, **6 emails are Spam and 4 are Not Spam**.
- The impurity is **lower** because the classes are becoming more separated.

Zero Impurity (Perfectly Pure Nodes)

A node with **all Spam (1s) or all Not Spam (0s)** has **zero impurity**, meaning it is completely pure:

Email	Label
Email 1	Spam (1)
Email 2	Spam (1)
Email 3	Spam (1)
Email 4	Spam (1)
Email 5	Spam (1)
Email 6	Spam (1)
Email 7	Spam (1)
Email 8	Spam (1)
Email 9	Spam (1)
Email 10	Spam (1)

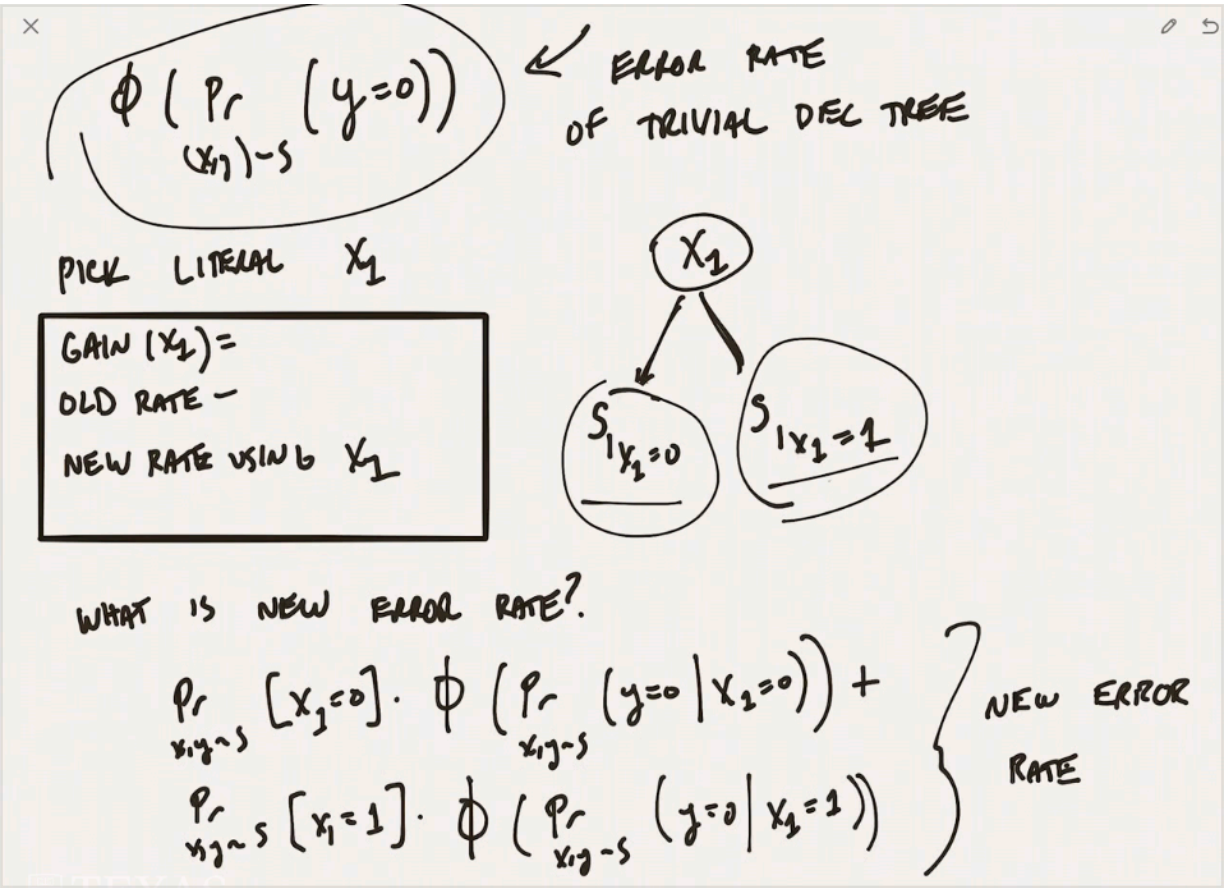
- **Impurity = 0** because all emails belong to the same class.

Types of Criteria in Decision Trees

1. **Gini Impurity (`criterion='gini'`)** - Default
   - Measures how often a randomly chosen element would be incorrectly classified.
   - Formula:
     $$Gini = 1 - \sum(p_i^2)$$
     where p_i is the probability of each class.
   - **Faster** than entropy and commonly used in decision trees.
2. **Entropy (`criterion='entropy'`)**
   - Uses **Information Gain** (from Information Theory).
   - Formula:
     $$Entropy = -\sum p_i \log_2(p_i)$$
   - Slower than Gini but **sometimes better** for deeper trees.

Which One to Use?

- **Gini** is computationally **faster** and works well in most cases.
- **Entropy** can sometimes create **more balanced splits** but is a bit slower.



Breaking It Down:

1. The **root node** is X_1, meaning we are splitting the dataset based on this feature.
2. The dataset is divided into two groups:
   - **Left branch** → $S|X_1 = 0$ (subset of data where $X_1 = 0$)
   - **Right branch** → $S|X_1 = 1$ (subset of data where $X_1 = 1$)

How This Relates to Decision Trees?

- We **choose a feature** X_1 that best separates the classes (e.g., spam vs. not spam).
- After the split, each subset will have its own **majority label** (either 0 or 1).
- This reduces **impurity (Gini or Entropy)** and helps classify data more accurately.

What's Next?

- Each of these subsets might be **further split** based on another feature (e.g., X_2).
- This continues until reaching **pure subsets** (all values in a node are the same).

What is "Gain" in Decision Trees?

In the given image, **"Gain"** refers to the improvement in classification accuracy achieved by splitting the data using a feature (X_1). It quantifies how much **impurity is reduced** when a split is made.

Formula for Gain

$$\text{Gain}(X_1) = \text{Old Error Rate} - \text{New Error Rate (after splitting on } X_1)$$

Where:

- **Old Error Rate**: The error rate before making any split.
- **New Error Rate**: The weighted sum of error rates in the two child nodes after splitting on X_1.

Understanding the Breakdown

1. **Before Splitting (Old Error Rate)**:
   - If we classify all examples with the majority class (without splitting), the **error rate** is computed as:

$$\phi\left(\Pr(y=0)\right)$$

   - This is the probability of misclassifying a random sample using a trivial decision tree (no splits).
2. **After Splitting on X_1 (New Error Rate)**:
   - The dataset is divided into two groups based on X_1 values.
   - The new error rate is computed as a **weighted sum** of the errors in the two child nodes:

$$\Pr(x_1=0)\cdot\phi\left(\Pr(y=0|x_1=0)\right)+\Pr(x_1=1)\cdot\phi\left(\Pr(y=0|x_1=1)\right)$$

   - Each term represents the probability of a sample belonging to a given branch times the error rate in that branch.
3. **Gain Calculation**:
   - The difference between the **Old Error Rate** and the **New Error Rate** gives us the **Gain** from splitting on X_1.
   - A **higher gain** means that the split significantly **reduces misclassification**, making X_1 a good feature to use.

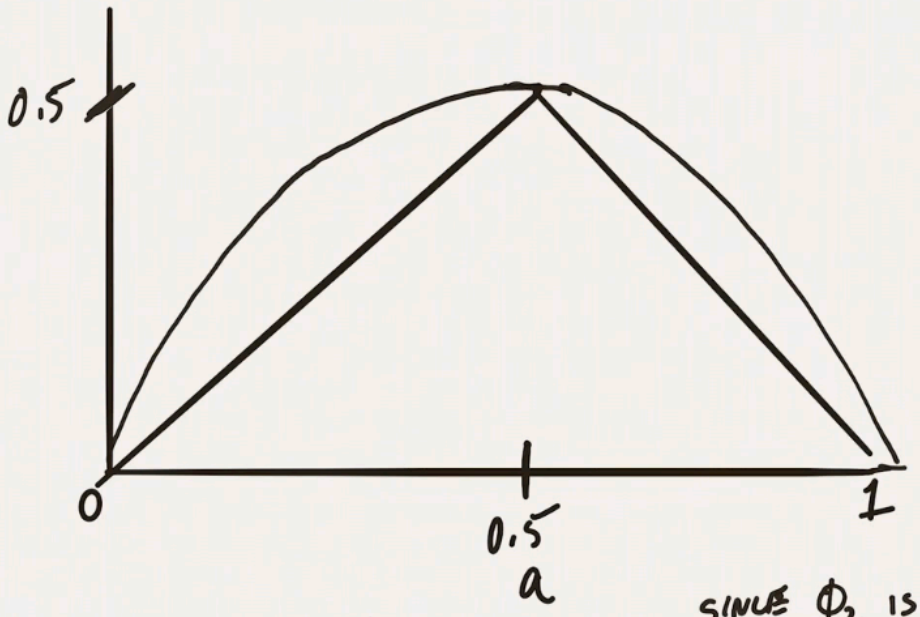
Key Takeaways

- **Gain** measures how much impurity (error) is **reduced** by splitting on a feature.
- **Higher gain → Better split → Feature is more useful** for classification.
- **If Gain = 0**, the feature does not help in reducing impurity and should not be used for splitting.

STRUCTURE OF TREE IS DETERMINED BY CHOICE OF ϕ

$\phi(a) = \min(a, 1-a)$ CORRESPONDED TO TRAINING ERROR

$\phi(a) = 2 \cdot a \cdot 1-a$ "GINI FUNCTION" OR "GINI INDEX"



$\phi_1(a) = \min(a, 1-a)$

$\phi_2(a) = 2 \cdot a \cdot (1-a)$

SINCE ϕ_2 IS AN UPPER BOUND ON ϕ_1

$\Rightarrow$ SMALL VALUES OF ϕ_2 (TRAINING ERROR

$\Rightarrow$ SMALL VALUES OF ϕ_1

This comes from the **Gini impurity formula** used in decision trees.

- **Gini impurity** measures how "mixed" the classes are in a node:

$$Gini = 1 - (p_1^2 + p_2^2)$$

  where **p_1 and p_2** are the probabilities of each class.

- If one class has proportion a and the other has $1 - a$, then:

$$Gini = 1 - (a^2 + (1 - a)^2)$$

- Expanding this:

$$Gini = 1 - (a^2 + 1 - 2a + a^2)$$

$$Gini = 1 - (1 - 2a + 2a^2)$$

$$Gini = 2a(1 - a)$$

Thus, **$\Phi_2(a) = 2 \times a \times (1 - a)$** represents the Gini impurity, which gives an **upper bound on the error rate (Φ_1)**.

💡 **Intuition:** If a split is perfectly balanced (a = 0.5), the Gini impurity is highest. If one class dominates (a → 0 or 1), the impurity drops to zero.

Why Is Φ_2 an Upper Bound for Φ_1?

- **$\Phi_1(a)$ (error rate) is always less than or equal to $\Phi_2(a)$ (impurity).**
- Gini impurity **overestimates the actual error** because it measures how mixed the data is **before** we make a decision.
- Once we assign the majority class as the label, the actual error (Φ_1) is always smaller.

📌 **Example:**

- If we have **80% spam (a=0.8)**, the Gini impurity is **$\Phi_2(0.8) = 2(0.8)(0.2) = 0.32$**.
- The actual error (misclassified emails) is **$\Phi_1(0.8) = \min(0.8, 0.2) = 0.2$**.
- So, Φ_1 is always **less than or equal to** Φ_2.

- **Φ_1 (min(a, 1 - a))** → Actual error rate (how often we are wrong).
- **Φ_2 (2a(1 - a))** → Gini impurity (how mixed the labels are before decision).
- **Gini impurity is always an upper bound** because it assumes we haven't assigned labels yet.

1. **Φ_1(a) = min(a, 1 - a)**
    - This means **Φ_1** takes the smaller value between a and $1 - a$.
    - It stays low when a is close to **0 or 1**.
2. **Φ_2(a) = 2 × a × (1 - a)**
    - This is a mathematical function that **forms a curved shape** (parabola).
    - It reaches **its highest value at a = 0.5** and is **always above Φ_1**.

Why is Φ_2 an Upper Bound on Φ_1?

Think of **Φ_2** as a "ceiling" and **Φ_1** as a "floor":

- **Φ_2 is always bigger or equal** to Φ_1 for any value of a.
- It gives a **safe estimate** of how big Φ_1 can be.

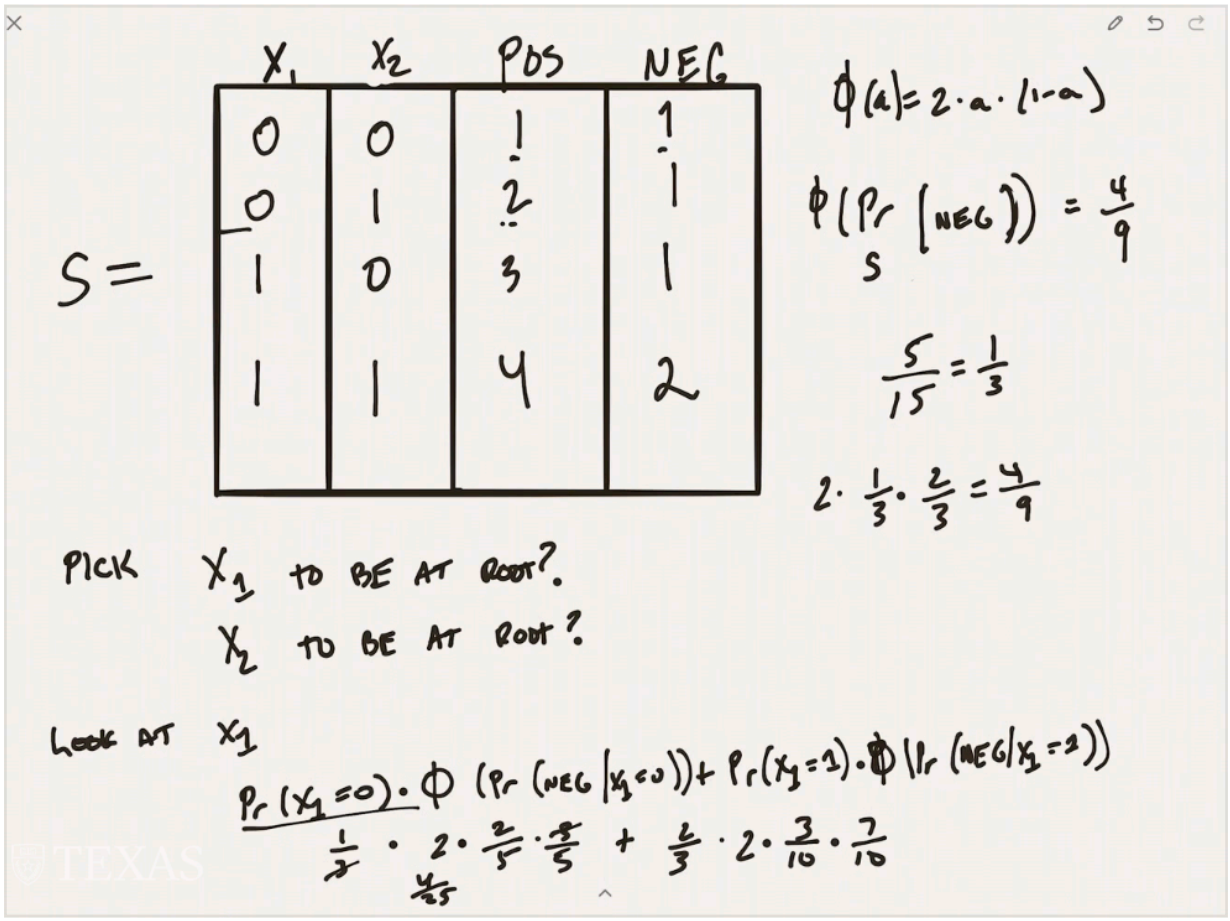
Real-World Analogy:

Imagine you're jumping on a trampoline:

- **Φ_1 is how high you actually jump** (your real height).
- **Φ_2 is the trampoline net** above you that **stops you from going too high** (a limit).
- No matter how you jump, **you will never go higher than the net!** 🚀

Why does this matter?

Since **Φ_2 is always higher**, if we make **Φ_2 small**, it **automatically makes Φ_1 small too**.
This is useful in **decision trees** because we want to **reduce error** (small Φ values = better model).



X1 = 0

2 × 2/5 × 3/5 × 5/15 = 0.16
X1 = 1
2 × 3/10 × 7/10 × 10/15 = 0.28

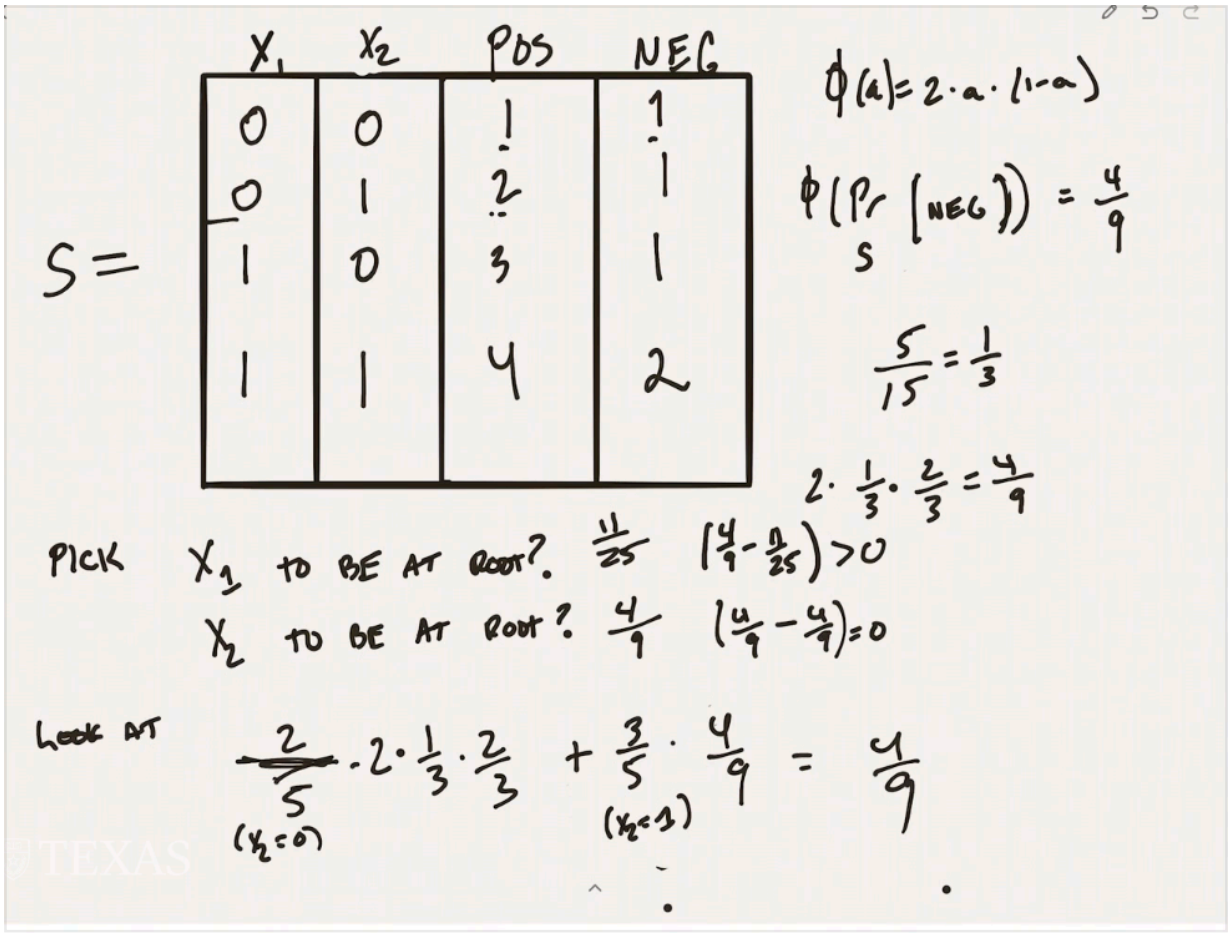
0.16+0.28 = 0.44

X2 = 0
2 × 2/6 × 4/6 × 6/15 = 0.178

X2 =1

2 × 3/9 × 6/9 × 9/15 = 0.267

0.178+0.267 = 0.445



Step 4: Choose the Root Feature

- If **Gain(X_1) > Gain(X_2)**, choose X_1.
- If **Gain(X_2) > Gain(X_1)**, choose X_2.

Since both features provide the **same Gini impurity reduction**, either feature could be used as the root. However, in cases where there's a small difference, we typically select the feature with the **higher gain**.

RANDOM FORESTS

- BUILD MANY "SMALL" DECISION TREES
- TAKE A MAJORITY OF RESULTING TREES

ALGORITHM FOR BUILDING MANY TREES:

TAKE TRAINING SET S

1. RANDOMLY SUBSAMPLE FROM S to CREATE S'
2. RANDOMLY PICK SOME FEATURES FROM $\{x_1, \ldots, x_n\}$ OF SIZE K.

BUILD A DECISION TREE USING S' AND THE K RANDOM FEATURES.

Example of Random Forest in Action

Let's consider a **spam email classification problem**, where we want to determine whether an email is **spam or not** using a dataset with different email features.

Step 1: Training Dataset (S)

We have a dataset with the following features:

- X_1 = Number of links in the email
- X_2 = Presence of "free money" keyword
- X_3 = Number of capitalized words
- X_4 = Sender's reputation score
- Y = Spam (1) or Not Spam (0)

Email	X_1 (Links)	X_2 ("Free Money")	X_3 (Caps)	X_4 (Reputation)	Y (Spam?)
1	3	Yes	5	Low	1 (Spam)
2	1	No	2	High	0 (Not)
3	4	Yes	7	Medium	1 (Spam)
4	0	No	1	High	0 (Not)
5	2	Yes	3	Low	1 (Spam)

Step 2: Creating Random Decision Trees

We will generate multiple decision trees by:

1. **Randomly Subsampling the Data**
    - Each tree gets a random subset of emails.
2. **Randomly Selecting Features**
    - Each tree only considers a random subset of features (e.g., one tree might use just X_1 and X_3, while another might use X_2 and X_4).

Step 3: Building Individual Trees

Each tree will independently learn patterns based on its own subset.

Example Decision Tree 1:

- If **X_2 = Yes** → Predict Spam
- Else If **$X_1 > 2$** → Predict Spam
- Otherwise → Not Spam

Example Decision Tree 2:

- If **$X_3 > 4$** → Predict Spam
- Else If **X_4 = High** → Predict Not Spam
- Otherwise → Spam

Each tree gives its own prediction.

Step 4: Making a Final Prediction (Majority Voting)

Let's say we have **5 trees**, and for a new email with:

- **$X_1 = 2$, X_2 = Yes, $X_3 = 4$, X_4 = Medium**

Predictions from trees:

- **Tree 1:** Spam
- **Tree 2:** Spam
- **Tree 3:** Not Spam
- **Tree 4:** Spam
- **Tree 5:** Not Spam

Final Decision:

- 3 out of 5 trees say **Spam → Final Prediction = Spam!**

Why Random Forest Works Well?

- **Avoids Overfitting**: A single decision tree might overfit, but averaging multiple trees reduces this risk.
- **Handles Noisy Data**: Different subsets ensure robustness.
- **Works with Large Datasets**: Efficient for high-dimensional data.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Sample dataset (X1: Links, X2: "Free Money" Keyword, X3: Caps, X4:
Reputation)
```

```
data = {
    "X1_Links": [3, 1, 4, 0, 2, 5, 1, 2, 3, 4],
    "X2_FreeMoney": [1, 0, 1, 0, 1, 1, 0, 0, 1, 1],  # 1 = Yes, 0 = No
    "X3_Caps": [5, 2, 7, 1, 3, 8, 2, 3, 6, 4],
    "X4_Reputation": [0, 1, 2, 1, 0, 0, 1, 2, 0, 1],  # 0 = Low, 1 = Medium, 2 = High
    "Spam_Y": [1, 0, 1, 0, 1, 1, 0, 0, 1, 1]  # 1 = Spam, 0 = Not Spam
}

# Convert to DataFrame
df = pd×DataFrame(data)

# Split into Features (X) and Target Variable (Y)
X = df.drop(columns=["Spam_Y"])
Y = df["Spam_Y"]

# Split Data into Training and Testing Sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
random_state=42)

# Create a Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=10, random_state=42)

# Train the Model
rf_model.fit(X_train, Y_train)

# Make Predictions
Y_pred = rf_model.predict(X_test)

# Calculate Accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print(f"Random Forest Model Accuracy: {accuracy * 100:.2f}%")

# Predict for a New Email (Example: 2 Links, "Free Money" Present, 4 Caps,
Medium Reputation)
new_email = np.array([[2, 1, 4, 1]])  # New email features
prediction = rf_model.predict(new_email)
print(f"New Email is Spam: {'Yes' if prediction[0] == 1 else 'No'}")
```

🔷 Explanation of the Code

1. **Data Preparation**:
   - Features: Number of links, "free money" keyword, number of capitalized words, and reputation score.
   - Target: Whether the email is spam (`1`) or not (`0`).
2. **Splitting Data**:
   - 70% for training, 30% for testing.
3. **Training the Random Forest Model**:
   - Using `n_estimators=10` (10 decision trees).
4. **Making Predictions**:
   - The model predicts spam/not spam for test data.
5. **Evaluating Performance**:
   - Accuracy is calculated using `accuracy_score()` .
6. **Testing on a New Email**:
   - Model predicts whether a new email (2 links, "Free Money" keyword present, 4 capitalized words, medium reputation) is spam.

🔥 Want to Improve the Model?

- Increase `n_estimators` (e.g., `100` instead of `10`).
- Tune hyperparameters (e.g., `max_depth` , `min_samples_split`).
- Use more real-world email features like sender domain, message length, etc.

Key Takeaways

✅ $n\_estimators = 10$ → Model creates **10 different decision trees.**
✅ **Each tree is trained on a random subset of data and features.**
✅ **Final prediction is based on majority voting.**
✅ **More trees generally improve accuracy and stability** (higher $n\_estimators$).