FADML: Module II

Weekly Summary

Name: Niranjan Herwadkar **Dates:** 27-Feb-25 to 06-Mar-25 (Week 1)

Introduction: The Learning Problem

Machine learning involves learning a function F: X → Y, where X is the input space (features) and Y is the output space (target values). If F is known, an algorithm can be directly designed; if unknown, machine learning techniques are used to approximate it.

Algorithmic Approach	Machine Learning Approach
X_{new} → [F] → Y_{new}	Training: (X, Y) → [L] → g ≈ F Inference: X_{new} → [g] → Y_{new}
F is a function designed by us	• L represents a learning algorithm that learns from data • g is the learned function, which approximates F

Where to apply Machine Learning?

1. **Movie Recommendation System** – Using machine learning, the system estimates the likelihood that a user will like an unseen movie and recommends it accordingly.
2. **Classification Problem: Logistic Regression** – Predicts categorical outcomes like disease detection (yes/no) or email spam (spam/not spam).
3. **Regression Problem: Salary Prediction** – Uses linear regression or neural networks to estimate salaries based on experience, education, and skills.
4. **Spam Filtering Using Naive Bayes** – Classifies emails as spam or not using probability-based text analysis.

How to apply Machine Learning?

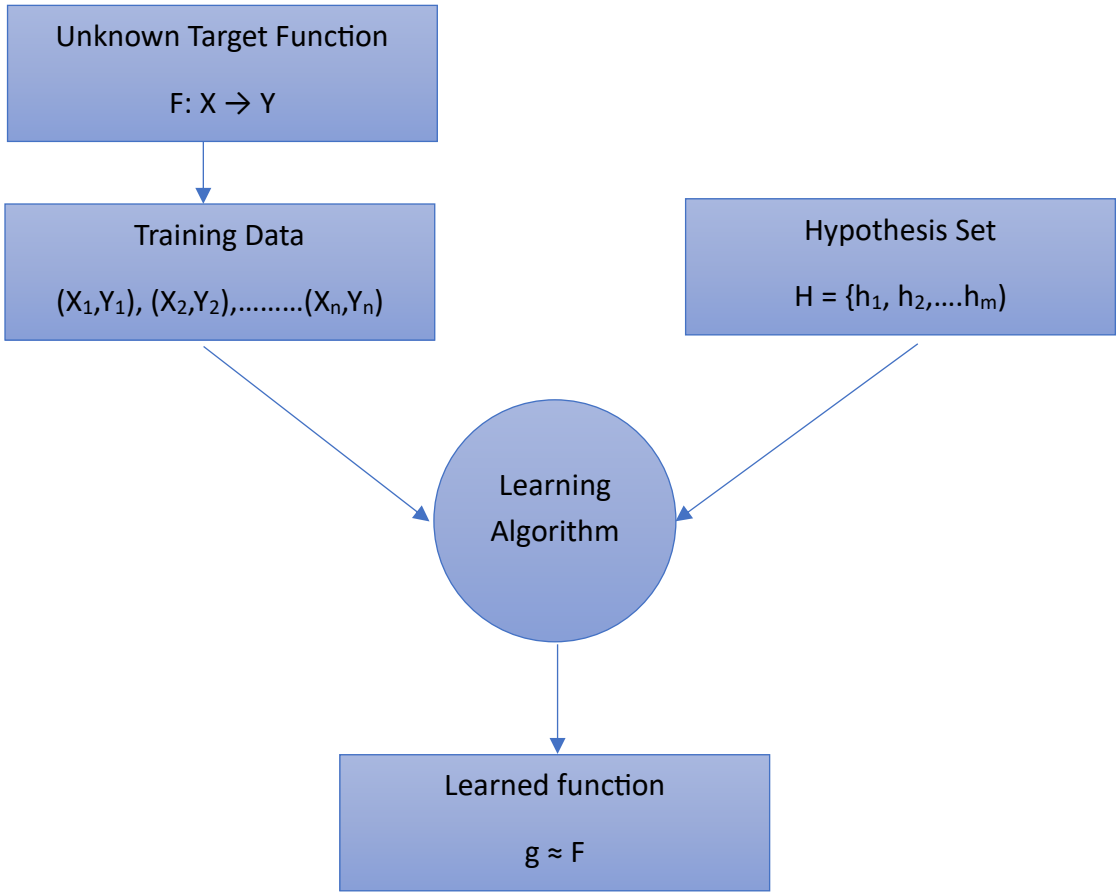
Machine learning is applied by training a model on data to recognize patterns and make predictions. In classification problems, a **decision boundary** is learned to separate different classes.

$$wx_1 + wx2 + w3 = 0$$

When dealing with misclassified data points, we adjust the **decision boundary** in two ways:

1. **Shifting the Decision Boundary** – Modifying $\mathbf{w_3}$ in the equation moves the boundary without changing its orientation.
2. **Rotating the Decision Boundary** – Changing $\mathbf{w_1}$ and $\mathbf{w_2}$ alters the slope, rotating the boundary to better separate the data.

Learning Process Diagram



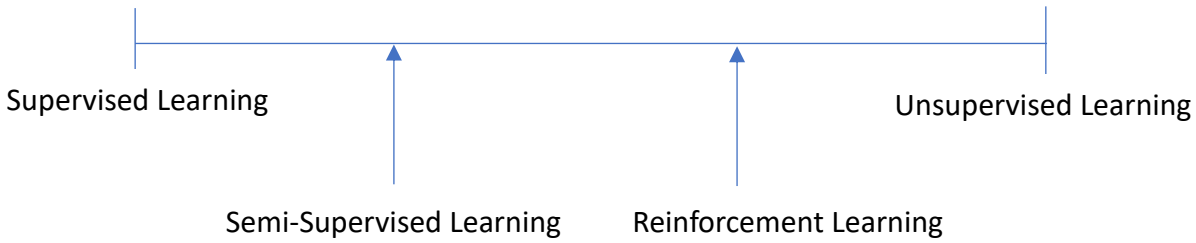
Supervised machine learning aims to approximate an unknown target function using a learning process. Key components include:

- **Unknown Target Function (F)**– Represents the true relationship between inputs X (features) and outputs Y(labels), which the model tries to approximate.
- **Training Data** – Labeled examples used to train the model.
- **Hypothesis Set (H)** – A collection of possible functions/models that could approximate F.
- **Learning Algorithm** – Selects the best hypothesis $h \in H$ by minimizing a loss function
- **Learned Function (g)** – The final output of the algorithm that approximates F and makes predictions on unseen data.

Types of learning

Machine learning is categorized based on how a model learns from data:

- **Supervised Learning** – Trains on labeled data to map inputs to outputs.
- **Unsupervised Learning** – Identifies patterns and structures in unlabeled data.
- **Semi-Supervised Learning** – Combines labeled and unlabeled data for learning.
- **Reinforcement Learning** – Learns through trial and error using rewards and penalties.



Feasibility of Learning

A key challenge in supervised learning is determining whether a function can be learned from a limited set of training examples while still generalizing well to unseen data. The **Probably Approximately Correct (PAC) learning framework** addresses this by establishing probabilistic bounds on how closely the chosen hypothesis approximates the true target function.

This is mathematically analyzed using **Hoeffding's Inequality**, which sets an upper bound on the likelihood that the empirical error (evaluated on training data) significantly deviates from the true error (calculated over the entire input distribution).

Hoeffding's Inequality

Hoeffding's inequality is a key result in probability theory that offers a statistical guarantee on how closely the training error E_{in} approximates the true generalization error E_{out}. It is formulated as:

$$P(|E_{in} - E_{out}| > \epsilon) \leq 2Me^{-2\epsilon^2 N}$$

where:

- **E_{in}** – Error measured on the training dataset (observed error).
- **E_{out}** – Error on the entire population or unseen data (true error).
- **ϵ** – Tolerance threshold, representing the allowed deviation between E_{in} and E_{out}.
- **N** – Number of samples in the training dataset.
- **M** – M is the number of hypotheses in the hypothesis set.

As we increase the number of training examples (N), the chance of a big gap between training error (E_{in}) and true error (E_{out}) becomes smaller and smaller. This means that with enough data, a machine learning model can learn a function that works well on unseen data.

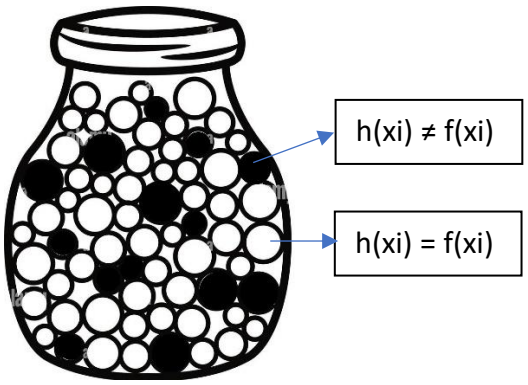
In simple terms, **if our model performs well on the training data, it is more likely to perform well on new data too—provided we have enough examples**. Hoeffding's Inequality mathematically supports this idea, showing that with more data, our training results become a reliable reflection of real-world performance.

Application of Hoeffding's Inequality: Marble selection problem

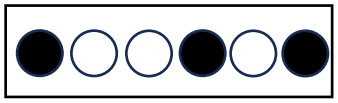
Objective: To understand the relationship between training error (E_{in}) and true error (E_{out}) using Hoeffding's Inequality. The key question is: *How well does a model's performance on training data approximate its performance on unseen data?*

Experimental Setup:

- A bin contains two types of marbles:
  - **Shaded marbles:** Incorrect predictions (misclassifications).
  - **Unshaded marbles:** Correct predictions.



- A learner does not know the true proportion of shaded marbles but can draw a random sample to estimate it.


N Samples

- The analogy:
  - Bin → Overall data distribution.
  - Marbles → Data points.
  - Picking marbles → Selecting a training dataset.
  - Fraction of shaded marbles in sample → Observed training error (E_{in}).
  - Fraction of shaded marbles in bin → True generalization error (E_{out}).

The question: *How closely does the fraction of shaded marbles in the sample estimate the true fraction in the bin?*

Application of Hoeffding's Inequality:

Hoeffding's Inequality helps us understand how closely the observed training error (E_{in}) matches the true generalization error (E_{out}).

The **larger the sample size**, the smaller the gap between E_{in} and E_{out}.

Conclusion:

- With a sufficiently large sample size, the observed training error (E_{in}) becomes a reliable estimate of the true error (E_{out}) with high probability.
- There exists a trade-off between accuracy, confidence, and required sample size:
    1. Higher accuracy (lower ϵ) requires more samples.
    2. Higher confidence (lower δ) increases sample size.
- This analysis supports the foundation of supervised learning: with enough data, we can generalize well from training to unseen data.

<u>Understanding dilation of bounds: Coin Flip Experiment</u>

Concept: Dilation of bounds refers to the widening of confidence intervals when multiple hypotheses are tested. This helps explain why testing many models increases the risk of false positives or overfitting.

Case 1: Single Coin Flip Experiment

- A fair coin is flipped 10 times.
- Probability of getting all heads = 0.00098 (very low).

Case 2: Multiple Coin Flip Experiment

1. 1,000 coins are flipped, each 10 times.
2. Probability that at least one coin gets all heads = 62.5% (much higher).

Key Takeaways

- With one hypothesis (single coin), extreme events are rare.
- With many hypotheses (1,000 coins), extreme events are more likely to happen by chance.
- In machine learning, testing too many models increases the risk of overfitting, as at least one model may fit the data just by luck rather than true patterns.

Decision Tree Learning

Decision tree learning is a supervised machine learning method that splits data into branches based on feature values, forming a tree-like structure to make predictions. It recursively partitions the dataset using criteria like Gini impurity or entropy to maximise information gain.

Cricket Match Example:

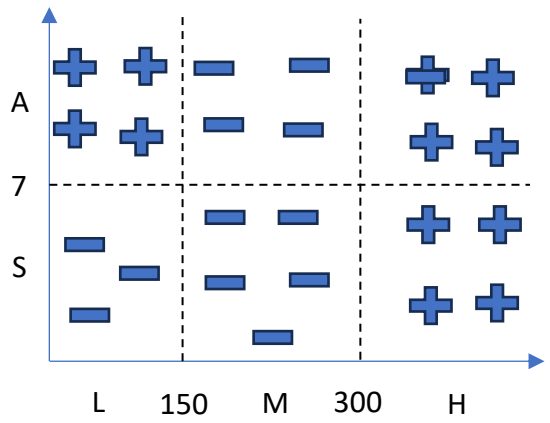
We wish to analyze how a decision tree can predict match outcomes using historical data from the Indian cricket team's matches. The dataset includes two main features:

- **Runs (R)** categorized as Low, Medium, or High.
- **Wickets (W)** categorized as Some (≤7 wickets lost) or Most (>7 wickets lost).

By examining these features, we aim to classify whether the team won or lost a match.

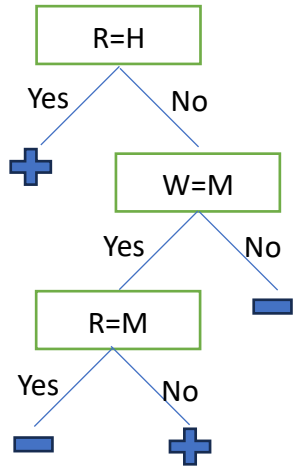
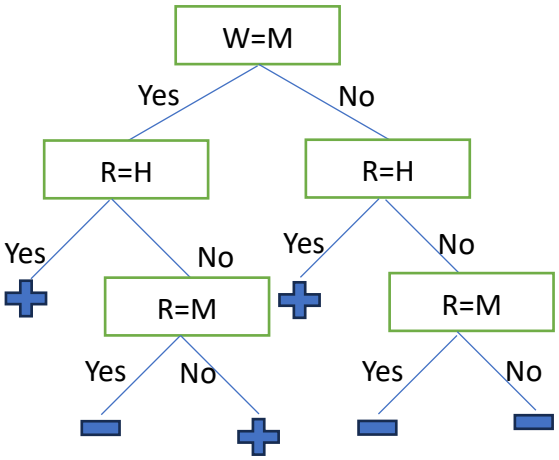
The scatter plot below illustrates the training data, where:

- **'+'** represents a win.
- **'–'** represents a loss.
- The **x-axis** categorizes Runs (R) into Low (L), Medium (M), and High (H).
- The **y-axis** categorizes Wickets (W) into Some (S) and Almost all (A).
- The **dashed lines** indicate the decision boundaries.



When building a decision tree, there are often multiple ways to split the data that lead to the same classification results. The main challenge is not only to create a tree that correctly classifies the data but also to identify the most optimal one among several valid options.

A decision tree classifies data by recursively dividing it based on feature values until all instances in a region belong to the same class. However, the sequence of these splits is not predetermined—there can be different ways to structure the tree while still achieving accurate classification.

Decision Tree 1	Decision Tree 2
	
Average comparisions $= \frac{1+3+3+2}{4} = \frac{9}{4}$	Average comparisions $= \frac{2+3+3+2+2}{5} = \frac{12}{5}$

The average comparisons in Decision Tree 1 are lesser than Decision Tree 2, indicating that it's the better option. However, we don't need to compute this every time to decide the best split choice. We can do so using the concepts of entropy, the Gini Index, information gain, and Impurity Reduction.

Measures of Impurity

1. **Entropy:**
   - Introduced by Claude Shannon in information theory. It measures the uncertainty in a probability distribution.
   - For a binary classification problem (k=2, where the probabilities are p_1 = p and p_2=1 - p), entropy is defined as:

$$E = -(p_+ \log_2(p_+) + p_- \log_2(p_-))$$

Behavior of Entropy:

- If all samples belong to one class (p=0 or p=1), then: E(S)=0, (no uncertainty).
- If classes are evenly split (p=0.5), then: E(S) = 1, (Max uncertainity)

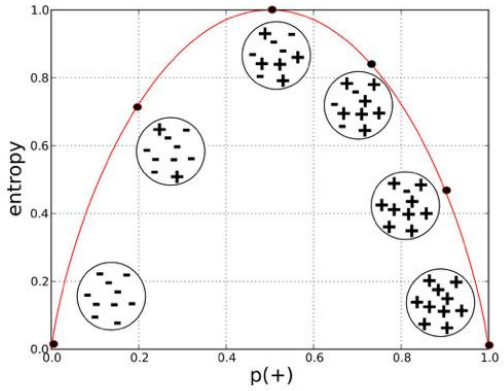
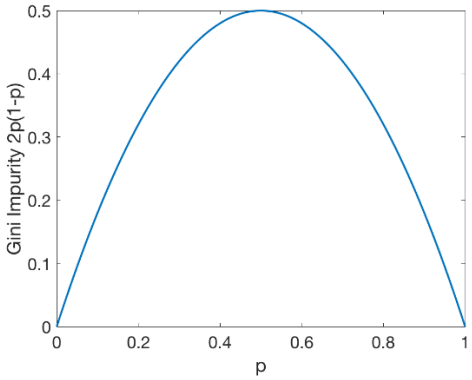
2. **Gini Index**
   - The Gini Index is a metric used in CART (Classification and Regression Trees) to measure impurity. It functions similarly to entropy in decision trees, as both are used to evaluate splits based on feature selection. However, their calculations differ significantly.
   - The Gini Impurity of a dataset after splitting can be determined using the following formula:

- In the case of binary classification (k = 2), where p is the probability of the positive class and 1−p represents the probability of the negative class, the formula simplifies to:
- Gini Index is defined as:

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

- This simplified formula shows that Gini Impurity reaches its maximum when classes are equally distributed (p=0.5) and is minimized when all samples belong to a single class.

Comparison between Entropy and Gini:

Entropy	Gini Index
	

Gini Impurity is often preferred for its **computational efficiency** over entropy. This is primarily because its calculation does not involve logarithmic functions, which require more processing power. As a result, Gini Impurity tends to be faster and is commonly used when selecting the best features for decision tree construction.

Information Gain and Impurity Reduction

1. Information Gain

Information Gain (IG) is a concept in decision trees for determining which feature best splits the data at each step. It measures how much uncertainty (entropy) is reduced by making a particular split.

Entropy of Set S:

$$H(S) = -(p_+ \log_2(p_+) + p_- \log_2(p_-))$$

Entropy after a split:

$$H(S,X) = \frac{|S_1|}{|S|}H(S_1) + \frac{|S_2|}{|S|}H(S_2)$$

Information gain (IG):

$$IG(S,X) = H(S) - H(S,X)$$

How to use Information gain in decision trees?
A **higher Information Gain** indicates a **more useful feature** for splitting. So at every node, we choose the feature with highest information gain.

2. Gini Impurity Reduction

Gini Impurity Reduction is another concept in decision trees for determining which feature best splits the data at each step. It uses **Gini Index** instead of entropy.

Gini Index of a set S:

$$G(S) = 1 - (p^2 + (1-p)^2) = 2p(1-p)$$

Gini Index after a split:

$$G(S,X) = \frac{|S_1|}{|S|}G(S_1) + \frac{|S_2|}{|S|}G(S_2)$$

Gini Impurity Reduction:

$$\Delta G(S,X) = G(S) - G(S,X)$$

How to use Gini Impurity Reduction in decision trees?
A **higher Gini Impurity Reduction** indicates a **more useful feature** for splitting. So, at every node, we choose the feature with highest impurity reduction.

Guiding Principle for model selection

1. Principle of Occam's Razor

Occam's Razor states that among competing explanations, the simplest one that fits the data is preferred. In the context of decision trees, this translates to favoring smaller and shallower trees that achieve high accuracy. Such trees minimize the risk of overfitting and improve generalization to unseen data.

2. Greedy Approach in Tree Construction

Decision trees are built using a **greedy algorithm** that selects the attribute with the **highest information gain** or highest impurity reduction at each step. This approach optimizes each decision locally, without revisiting previous splits.

3. Impact of Inductive Bias

Since decision trees do not backtrack once a split is made, they develop an **inductive bias toward locally optimal decisions**. While this often leads to effective models, it also means that **an early suboptimal split** cannot be corrected later in the process, potentially affecting overall performance.

Avoiding Overfitting in Decision Trees

Decision trees can overfit due to noise and missing data, making them overly complex and poor at generalization. **Reduced-Error Pruning (REP)** helps control this by simplifying the tree while maintaining accuracy.

Causes of Overfitting:

Noise in Data

- **Incorrect labels** – Misclassified data points
- **Outliers** – Extreme values
- **Random variations** – Inconsistent patterns

Missing Data

- Leads to **biased splits** and **unnecessary complexity**

Reduced-Error Pruning (REP):

REP removes unnecessary branches to prevent overfitting:

1. Train & Validate – Split data for validation
2. Check Leaf Nodes – Evaluate if removal simplifies the tree
3. Prune if Accuracy Holds – Replace branches with majority class if accuracy is unchanged
4. Repeat Until No More Gains

Bayesian Learning

In a typical classification problem, the goal is to approximate an unknown function g using a learned function f, where:

$$g \approx f : X \to Y$$

Here, X represents the input features, and Y represents the output classes (+1 or -1). Different machine learning approaches approximate this function differently.

Probabilistic (Bayesian) Approach

The Bayesian approach relies on posterior probabilities, estimating the probability of each class given the input features using Bayes' Theorem:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Instead of directly assigning a class label, classification is based on the likelihood ratio:

$$\frac{P(y = 1|X_{1,\ldots\ldots,}X_n)}{P(y = 0|X_{1,\ldots\ldots,}X_n)}$$

- If the ratio > 1, the instance is classified as +1 (Class 1).
- If the ratio < 1, the instance is classified as -1 (Class 0).

This approach incorporates uncertainty, making it useful in cases with limited data or noisy environments.

Joint Probability Distribution

A **Joint Probability Distribution Table (JPDT)** represents the probabilities of different combinations of random variables occurring together. It is a structured way to describe the relationships between multiple random variables in a **probabilistic model**.

JPDT simplifies **probability calculations** by allowing direct lookup and conditional probability computation.

Example: JPDT for two Binary Variables (X, Y):

X	Y	P(X, Y)
0	0	0.2
0	1	0.3
1	0	0.1
1	1	0.4

To find P(Y = 1 | X = 1):

$$P(Y = 1|X = 1) = \frac{P(X = 1,\ Y = 1)}{P(X = 1)}$$

$$= \frac{0.4}{0.1 + \ 0.4} = 0.8$$

JPDT simplifies **probability calculations** by allowing direct lookup and conditional probability computation.

Issue of data sparsity

For n attributes, a Joint Probability Distribution Table (JPDT) requires 2^n +1 entries. This **grows exponentially**, making **storage and estimation impractical**.

Example: With 100 attributes, we need 2^{100} entries—far beyond feasible storage or data availability, leading to data sparsity.

Solutions to Address This Issue:

1. **Smart Estimation** – Efficiently estimate probabilities even when data is sparse.
2. **Smart Representation** – Store and structure the JPDT efficiently to reduce memory and computation costs.

These approaches help manage large-scale probabilistic models while ensuring accurate predictions.

Estimation

1. **Maximum Likelihood Estimation (MLE)**

   MLE is a technique used to estimate the parameter **θ** of a probability distribution by **maximizing the likelihood function** P(Data|θ). This means finding the parameter value that makes the observed data most probable.

   In **Bayesian learning**, MLE does **not** consider prior knowledge about θ; it relies only on the **observed data**, making it a purely data-driven approach.

$$\theta_{MLE} = \arg max_{\theta} P(Data|\theta)$$

   For example, MLE for coin toss problem can be found using the above formula to be:

$$\hat{\theta} = \frac{\alpha_H}{\alpha_H + \alpha_T}$$

   where:

   - α_H represents the number of observed heads
   - α_T represents the observed tails
   - θ represents the probability of getting heads in a single coin toss

2. **Maximum A Posteriori (MAP)**

   MAP estimation is a Bayesian approach for estimating a parameter θ by considering **both observed data and prior knowledge**. It finds the most likely value of θ by **maximizing the posterior probability**:

$$\theta_{MAP} = \arg max_{\theta} P(\theta|Data)$$

$$\theta_{MAP} = \arg max_{\theta} \left(\frac{P(Data|\theta)P(\theta)}{P(Data)}\right)$$

   Since P(Data) is constant for all θ,

$$\theta_{MAP} = \arg max_{\theta} P(Data|\theta)P(\theta)$$

Unlike Maximum Likelihood Estimation (MLE), which only maximizes P(Data|θ), MAP also incorporates the prior P(θ). This makes MAP **particularly effective** when **data is limited or noisy**, as it balances data-driven learning with prior information, reducing instability in small datasets.

For example, MAP for coin toss problem can be found using the above formula to be:

$$\hat{\theta} = \frac{\alpha_H + H}{\alpha_H + \alpha_T + H + T}$$

where:

- α_H represents the number of observed heads,
- α_T represents the observed tails
- θ represents the probability of getting heads in a single coin toss
- H represents the number of observed heads
- T represents the number of observed tails.