**GOWRA SASIDHAR**

**2320030500**

**Definition: Probability of Addition (Addition Rule of Probability)**

The **Addition Rule of Probability** is used to find the probability that **at least one of two events** occurs (i.e., the union of two events).

There are **two main cases**:

**\*\*1. If Events A and B are mutually exclusive (they cannot happen at the same time):**

P(A∪B) = P(A)+P(B)

**\*\*2. If Events A and B are not mutually exclusive (they can happen at the same time):**

P(A∪B) = P(A)+P(B)−P(A∩B)

**Example 1: Mutually Exclusive Events**

Let’s say we roll a **single die**.

* Let **A** be the event “rolling a 2” → P(A)=1/6
* Let **B** be the event “rolling a 5” → P(B)=1/6

Since you can’t roll both a 2 and a 5 at the same time, they are **mutually exclusive**.

P(A∪B) = P(A)+P(B)

= 1/6+1/6=2/6

=1/3

**Example 2: Not Mutually Exclusive Events**

In a deck of 52 playing cards:

* Let **A** be the event “card is a heart” → P(A) = 13/52
* Let **B** be the event “card is a queen” → P(B) = 4/52​
* One card is both a **heart and a queen** (the Queen of Hearts) → P(A∩B)=1/52

So,

P(A∪B) = 13/52 + 4/52 – 1/52

= 16/52

= 4/13

Here's a simple **Java program** that demonstrates the **Addition Rule of Probability** for both **mutually exclusive** and **non-mutually exclusive** events.

public class ProbabilityAddition {

// Method for mutually exclusive events

public static double mutuallyExclusive(double pA, double pB) {

return pA + pB;

}

// Method for non-mutually exclusive events

public static double notMutuallyExclusive(double pA, double pB, double pAandB) {

return pA + pB - pAandB;

}

public static void main(String[] args) {

// Example 1: Mutually Exclusive Events

// Rolling a 2 or a 5 on a die

double pA = 1.0 / 6; // Probability of rolling 2

double pB = 1.0 / 6; // Probability of rolling 5

double result1 = mutuallyExclusive(pA, pB);

System.out.println("Probability of rolling 2 or 5 (mutually exclusive): " + result1);

// Example 2: Not Mutually Exclusive Events

// Drawing a heart or a queen from a deck of cards

double pHeart = 13.0 / 52;

double pQueen = 4.0 / 52;

double pHeartAndQueen = 1.0 / 52; // Queen of Hearts

double result2 = notMutuallyExclusive(pHeart, pQueen, pHeartAndQueen);

System.out.println("Probability of heart or queen (not mutually exclusive): " + result2);

}

}

**Definition: Probability of Multiplication (Multiplication Rule of Probability)**

The **Multiplication Rule** is used to find the probability that **two events A and B both happen** (i.e., the intersection A∩B).

There are **two cases**:

**1. Independent Events:**

If events A and B **do not affect each other**, then:

P(A∩B) = P(A)×P(B)

**2. Dependent Events:**

If the occurrence of one event **affects** the probability of the other, then:

P(A∩B) = P(A)×P(B∣A)

Where:

* P(B∣A) = Probability of B **given** A has occurred

**Example 1: Independent Events**

You toss a coin and roll a die.

* Let **A** be "getting heads" → P(A) = 1/2
* Let **B** be "rolling a 6" → P(B) = 1/6​

P(A∩B)=12×16 = 1/12

**Example 2: Dependent Events**

From a deck of 52 cards, draw **2 cards without replacement**:

* Probability of first card being an ace: P(A) = 4/52
* Probability second card is also an ace (1 less ace, 51 cards left):  
  P(B∣A) = 3/51

P(A∩B) = 452×351 = 12/2652 = 1/221

Here is a simple **Java program** that demonstrates the **Multiplication Rule of Probability** for both **independent** and **dependent** events:

public class ProbabilityMultiplication {

public static double independentEvents(double pA, double pB) {

return pA \* pB;

}

public static double dependentEvents(double pA, double pBGivenA) {

return pA \* pBGivenA;

}

public static void main(String[] args) {

// Example 1: Independent Events (Tossing a coin and rolling a die)

double pHead = 1.0 / 2;

double pSix = 1.0 / 6;

double result1 = independentEvents(pHead, pSix);

System.out.println("P(Head and 6) [independent]: " + result1);

// Example 2: Dependent Events (Drawing 2 aces without replacement)

double pAce1 = 4.0 / 52;

double pAce2GivenAce1 = 3.0 / 51;

double result2 = dependentEvents(pAce1, pAce2GivenAce1);

System.out.println("P(Ace and Ace) [dependent]: " + result2);

}

}

**Definition: Bayes' Theorem**

**Bayes' Theorem** is used to find the probability of an event **based on prior knowledge** of related conditions.

**Bayes' Theorem Formula:**

P(A∣B) = P(B∣A)⋅P(A) / P(B)

Where:

* P(A∣B) = Probability of A given B (posterior)
* P(B∣A) = Probability of B given A (likelihood)
* P(A) = Probability of A (prior)
* P(B) = Probability of B (evidence)

**Example: Medical Test (Common Use Case)**

Suppose:

* 1% of people have a disease → P(D)=0.01
* If a person **has** the disease, the test is **positive 99%** of the time → P(Pos∣D)=0.99
* If a person **does not have** the disease, the test is **positive 5%** of the time → P(Pos∣¬D)=0.05

What is the probability that a person **actually has the disease**, given they tested positive?

**Step-by-Step:**

We want to calculate P(D∣Pos):

P(D/Pos) = P(Pos/D)⋅P(D) / P(Pos)

But we need P(Pos) (total probability of a positive test):

P(Pos) = P(Pos∣D)⋅P(D)+P(Pos∣¬D)⋅P(¬D)

= (0.99) (0.01) + (0.05) (0.99)

= 0.0099 + 0.0495

= 0.0594

Now apply Bayes' theorem:

P(D∣Pos) = 0.99 × 0.01 / 0.0594 ≈ 0.0099 / 0.0594 ≈ 0.1667

So, even with a positive test, the chance the person **actually has the disease is only ~16.67%**.

​

Here's a simple **Java program** that demonstrates **Bayes' Theorem** using the medical test example:

public class BayesTheorem {

public static double calculateBayes(double pA, double pBgivenA, double pBgivenNotA) {

double pNotA = 1 - pA;

double pB = (pBgivenA \* pA) + (pBgivenNotA \* pNotA);

return (pBgivenA \* pA) / pB;

}

public static void main(String[] args) {

double pDisease = 0.01; // P(D)

double pPositiveGivenDisease = 0.99; // P(Pos | D)

double pPositiveGivenNoDisease = 0.05; // P(Pos | ¬D)

double result = calculateBayes(pDisease, pPositiveGivenDisease, pPositiveGivenNoDisease);

System.out.println("Probability of having disease given a positive test: " + result);

}

}