Lab 2 - Probabilidadesgod

February 20, 2025

1 Laboratorio 2 - Probabilidades

1.1 Adrian Arimany Zamora - 211063 & Daniel Sarmieto - 231105

1.1.1 Instrucciones: Escriba programas en Python para generar todos los posibles resultados de cada experimento aleatorio, filtrar según las condiciones de cada evento y contar para resolver los ejercicios.

[25]:
```
import itertools as it
from math import comb
```

1.2 Ejercicio 1.

Suponga que una caja contiene dos monedas de tipo A y una de tipo B. Cuando se lanza una moneda de tipo A, sale cara con probabilidad 1/4, mientras que cuando se lanza una moneda de tipo B, sale cara con probabilidad 3/4.

Un experimento aleatorio consiste en elegir al azar una moneda de la urna y lanzarla.

Si se sabe que el resultado fue cara, ¿cuál es la probabilidad de que sea una moneda de tipo A?

[ ]:
```
#Given a uniform probability for p and q, then p = Probability(Heads)
#Event is the condition that wants to be determined.
def flipMonedas(p, event):
    q = 1 - p
    # Definimos las monedas como tuplas (tipo, p = cara, q = cruz)
    monedas = [
        ("p", p),  # p = cara
        ("p", p),  # p = cara
        ("q", q)   # q = cruz
    ]
    # Probabilidad Total
    p_cara = sum((1/3) * p for (_, p) in monedas)

    # P(A|p)
    p_AyCara = sum((1/3) * p for (tipo, p) in monedas if tipo == event)

    # P(A | cara)
    p_A_dado_cara = p_AyCara / p_cara
```

```
    print("Ejercicio 1:")
    print(f"La probabilidad de que sea moneda A dado que salió cara es:␣
 ↪{p_A_dado_cara:.4f}")

flipMonedas(p= 1/4, event= "p")
```

```
Ejercicio 1:
La probabilidad de que sea moneda A dado que salió cara es: 0.4000
```

1.3 Ejercicio 2.

Suponga que se extrae un 10 rojo y un 6 rojo de un mazo estándar de cartas. Luego, un experimento aleatorio consiste en extraer una carta del mazo incompleto.

Sean E el evento de que la tarjeta extraída es un 10, F el evento en que la carta extraída es roja y G el evento de que la carta extraída sea un 10 o un 6.

Demuestre que E y F no son independientes, pero son condicionalmente independientes* condicionados a G.

Definición: Dos eventos, E y F, son *condicionalmente independientes* dados G si y solo si, la probabilidad de que ambos eventos ocurran al mismo tiempo, dado que G ya ocurrió, es igual al producto de la probabilidad de que cada evento ocurra individualmente, dado que G ya ocurrió. En símbolos:

$$P(E \cap F \mid G) = P(E \mid G)P(F \mid G)$$

[22]:
```
def deckPicker():
    # Construimos un mazo estándar con un tuple (rango, symbolo)
    # Symbolos: [Clubs = 'C', Diamonds = 'D', Hearts = 'H', Spades = 'S'], ␣
 ↪Rangos: [1..13]
    palos = ['C','D','H','S']  # C=Trebol (negro), D=Diamante (rojo), H=Corazón␣
 ↪(rojo), S=Picas (negro)
    rangos = range(1,14)       # 1=A, 2..10, 11=J, 12=Q, 13=K

    # Producir las combinaciones del mazo compelto
    mazo_completo = [(r, p) for r in rangos for p in palos]

    # Condicion dada
    carta1_removida = (10, 'H')
    carta2_removida = (6, 'H')

    # Mazo reducido
    mazo_reducido = [c for c in mazo_completo if c not in [carta1_removida,␣
 ↪carta2_removida]]

    # Definir funciones para eventos
    def es_10(carta):
        return carta[0] == 10
```

```python
def es_roja(carta):
    return carta[1] in ['H','D']

def es_10_o_6(carta):
    return carta[0] in [10, 6]

# Contamos las cartas que cumplen cada evento
n_total = len(mazo_reducido)  # debería ser 50

# E = "La carta es 10"
E_cartas = [c for c in mazo_reducido if es_10(c)]
pE = len(E_cartas) / n_total

# F = "La carta es roja"
F_cartas = [c for c in mazo_reducido if es_roja(c)]
pF = len(F_cartas) / n_total

# E union F
EF_cartas = [c for c in mazo_reducido if es_10(c) and es_roja(c)]
pEF = len(EF_cartas) / n_total

# Comprobación de independencia
print("Ejercicio 2:")
print(f"P(E)            = {pE:.4f}")
print(f"P(F)            = {pF:.4f}")
print(f"P(E union F)    = {pEF:.4f}")
print(f"P(E)*P(F)       = {pE*pF:.4f}")
print("¿Son E y F independientes?", "Sí" if pE*pF == pEF else "No")

# Evento G: "La carta es un 10 o un 6"
G_cartas = [c for c in mazo_reducido if es_10_o_6(c)]
pG = len(G_cartas) / n_total

# Probabilidades condicionadas a G
# P(E | G) = ( E union G ) / ( G )
EG_cartas = [c for c in G_cartas if es_10(c)]
pE_dado_G = len(EG_cartas) / len(G_cartas)

FG_cartas = [c for c in G_cartas if es_roja(c)]
pF_dado_G = len(FG_cartas) / len(G_cartas)

EFG_cartas = [c for c in G_cartas if es_10(c) and es_roja(c)]
pEF_dado_G = len(EFG_cartas) / len(G_cartas)

print(f"\nP(G)                        = {pG:.4f}")
print(f"P(E|G)                     = {pE_dado_G:.4f}")
print(f"P(F|G)                     = {pF_dado_G:.4f}")
```

```
    print(f"P(E F|G)                    = {pEF_dado_G:.4f}")
    print(f"P(E|G)*P(F|G)               = {pE_dado_G*pF_dado_G:.4f}")
    print("¿Son E y F cond. indep. dado G?",
          "Sí" if pEF_dado_G  == pE_dado_G*pF_dado_G  else "No")

deckPicker()
```

```
Ejercicio 2:
P(E)            = 0.0600
P(F)            = 0.4800
P(E union F)    = 0.0200
P(E)*P(F)       = 0.0288
¿Son E y F independientes? No

P(G)                      = 0.1200
P(E|G)                    = 0.5000
P(F|G)                    = 0.3333
P(E F|G)                 = 0.1667
P(E|G)*P(F|G)           = 0.1667
¿Son E y F cond. indep. dado G? Sí
```

1.4 Ejercicio 3.

Un experimento aleatorio consiste en extraer tres cartas con reemplazo* de un mazo estándar de 52 cartas.

Sean E el evento de que la carta 1 y la carta 2 tienen el mismo palo; F el evento que la carta 2 y la carta 3 tienen el mismo palo y G sea el evento de que la carta 1 y la carta 3 tienen el mismo palo.

Demuestre que estos eventos son independientes por pares, pero no independientes.

* Se extrae una carta, se toma nota y se regresa al mazo, se mezclan las cartas, se extrae la siguiente, se toma nota, se mezclan, y se extrae la tercera.

[ ]:
```
def cardPickerIndepency():
    suits = ['C', 'D', 'H', 'S']  # 4 suits
    ranks = range(1, 14)          # 13 ranks
    # Full deck
    deck = [(rank, suit) for suit in suits for rank in ranks]
    num_cards = len(deck)   # 52
    # Sample space for 3 cards with replacement:
    total_combinations = num_cards ** 3

    def same_suit(card1, card2):
        return card1[1] == card2[1]

    # Counters
    count_e = 0
    count_f = 0
```

```python
    count_g = 0
    count_ef = 0
    count_eg = 0
    count_fg = 0
    count_efg = 0

    for card1, card2, card3 in it.combinations(deck, 3):
        event_e = same_suit(card1, card2)
        event_f = same_suit(card2, card3)
        event_g = same_suit(card1, card3)

        count_e += event_e
        count_f += event_f
        count_g += event_g
        count_ef += event_e and event_f
        count_eg += event_e and event_g
        count_fg += event_f and event_g
        count_efg += event_e and event_f and event_g

    probability_e = count_e / total_combinations
    probability_f = count_f / total_combinations
    probability_g = count_g / total_combinations

    probability_ef = count_ef / total_combinations
    probability_eg = count_eg / total_combinations
    probability_fg = count_fg / total_combinations

    probability_efg = count_efg / total_combinations

    print("Exercise 3:")
    print("Pairwise independence")
    if (probability_ef == probability_e * probability_f and
            probability_eg == probability_e * probability_g and
            probability_fg == probability_f * probability_g):
        print("Events E with F, E with G, F with G are independent")
    else:
        print("Events E with F, E with G, F with G are not independent")


    print("Conditional independence")
    if probability_efg == probability_e * probability_f * probability_g:
        print("Events E, F, and G are independent")
    else:
        print("Events E, F, and G are not independent")
cardPickerIndepency()
```

```
Exercise 3:
```

```
- Pairwise independence (equalities hold).
Events E with F, E with G, F with G are not independent
Events E, F, and G are not independent
```

1.5 Ejercicio 4.

Supongamos que la caja 1 contiene una pelota blanca y cuatro rojas, la caja 2 contiene dos pelotas blancas y tres rojas, y la caja 3 contiene tres pelotas blancas y dos rojas. En un experimento, se selecciona al azar una caja y, luego, se escogen tres pelotas.

a. Si se sabe que la caja 1 no ha sido seleccionada, ¿cuál es la probabilidad de escoger exactamente dos pelotas rojas?
b. Determine qué es más probable: salgan exactamente dos pelotas rojas o que salgan más pelotas blancas que rojas.
c. Si se sabe que han salido exactamente dos pelotas rojas, ¿cuál es la probabilidad de que se haya escogido la caja 3?

[26]:
```python
def urn():
    # Balls: 'W' = white, 'R' = red
    box1 = ['W', 'R', 'R', 'R', 'R']  # 1W, 4R
    box2 = ['W', 'W', 'R', 'R', 'R']  # 2W, 3R
    box3 = ['W', 'W', 'W', 'R', 'R']  # 3W, 2R
    # a) Probability of drawing exactly 2 red balls given box1 is not chosen
    def probability_two_reds(box):
        # Calculate P(2R) without replacement in 3 draws
        # = (# ways to choose 2R and 1W) / # ways to choose 3 out of 5
        num_reds = box.count('R')
        num_whites = box.count('W')
        ways_two_reds = comb(num_reds, 2) * comb(num_whites, 1)
        total_ways = comb(len(box), 3)
        return ways_two_reds / total_ways

    prob_two_reds_box2 = probability_two_reds(box2)
    prob_two_reds_box3 = probability_two_reds(box3)

    prob_two_reds_given_not_box1 = 0.5 * prob_two_reds_box2 + 0.5 *␣
↪prob_two_reds_box3

    print("Exercise 4:")
    print("a) Probability of drawing exactly 2 red balls given box1 is not␣
↪chosen:")
    print(f"   P(2R | not box1) = {prob_two_reds_given_not_box1:.4f}")

    # b) Compare P(2 reds) vs P(more whites than reds) (overall, with 1/3␣
↪probability for each box)
    def probability_of_event_in_box(box, condition):
        total_combinations = 0
        satisfying_combinations = 0
```

```
        for combo in it.combinations(box, 3):
            total_combinations += 1
            if condition(combo):
                satisfying_combinations += 1
        return satisfying_combinations / total_combinations

    def is_two_reds(triple):
        return triple.count('R') == 2

    def more_whites_than_reds(triple):
        return triple.count('W') > triple.count('R')

    # Probability of each "global" event = average of the 3 boxes (1/3 each)
    prob_two_reds_global = (probability_of_event_in_box(box1, is_two_reds) +
                            probability_of_event_in_box(box2, is_two_reds) +
                            probability_of_event_in_box(box3, is_two_reds)) / 3

    prob_more_whites_global = (probability_of_event_in_box(box1,␣
↪more_whites_than_reds) +
                               probability_of_event_in_box(box2,␣
↪more_whites_than_reds) +
                               probability_of_event_in_box(box3,␣
↪more_whites_than_reds)) / 3

    print("\nb) Comparison of global probabilities:")
    print(f"   P(2 reds)              = {prob_two_reds_global:.4f}")
    print(f"   P(more whites than reds)= {prob_more_whites_global:.4f}")
    if prob_two_reds_global > prob_more_whites_global:
        print("   It is more probable to draw exactly 2 reds.")
    else:
        print("   It is more probable to draw more whites than reds.")

    # c) P(box3 | 2R) = [P(2R|box3)*P(box3)] / P(2R)
    # Using previously calculated values:
    prob_box3 = 1 / 3
    prob_two_reds_box3 = probability_of_event_in_box(box3, is_two_reds)
    prob_box3_given_two_reds = (prob_two_reds_box3 * prob_box3) /␣
↪prob_two_reds_global

    print("\nc) Probability of having chosen box3 given that exactly 2 reds␣
↪were drawn:")
    print(f"   P(box3 | 2R) = {prob_box3_given_two_reds:.4f}")

urn()
```

```
Exercise 4:
a) Probability of drawing exactly 2 red balls given box1 is not chosen:
```

```
   P(2R | not box1) = 0.4500

b) Comparison of global probabilities:
   P(2 reds)              = 0.5000
   P(more whites than reds)= 0.3333
  It is more probable to draw exactly 2 reds.

c) Probability of having chosen box3 given that exactly 2 reds were drawn:
   P(box3 | 2R) = 0.2000
```