Cadena de Markov

July 3, 2025

JANNA LUZ FLORENTINO AGRAMONTE 10142764 ,

Introducción

Este informe explica, paso a paso, una especie de experimento por computadora llamado simulación de Monte Carlo, que sirve para observar cómo varios usuarios intentan conectarse a una red de manera aleatoria. Cada usuario tiene varias oportunidades para lograrlo, y se mide si tiene éxito o no. Todo esto se basa en un modelo matemático llamado Poisson, que ayuda a representar situaciones donde las cosas ocurren de forma impredecible, como cuando llegan personas a una fila. La idea es entender mejor qué tan seguido hay choques entre usuarios (cuando intentan conectarse al mismo tiempo) y cómo varían estos intentos con el tiempo. Se simularon un millón de casos para que los resultados sean confiables y reflejen bien la realidad.

,

Análisis y Resultados

Parte #1: Librerías

[21]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

En esta parte del código se traen unas herramientas que ayudan a que el experimento funcione bien. Una se llama NumPy y sirve para hacer cálculos con números, como sumar, contar o sacar promedios. Otra se llama matplotlib.pyplot, que permite crear gráficos para mostrar los resultados de forma visual, como dibujos de barras o líneas. Y por último, seaborn, que se usa para que esos gráficos se vean más bonitos y fáciles de entender.

,

Parte #2: Estilo visual y colores personalizados

[22]:
```
# Estilo visual general
sns.set_context("notebook")
sns.set_style("whitegrid")
```

Estas líneas del código sirven para que los gráficos se vean más ordenados y agradables a la vista cuando los mostramos en un cuaderno interactivo como Jupyter Notebook. Se utiliza una herramienta llamada Seaborn que mejora el diseño visual, haciendo que los colores y estilos se vean más profesionales y fáciles de interpretar. Después, se eligen unos colores personalizados, del tipo pastel, que son suaves y agradables, enfocadas a mi preferencia.

[23]:
```
# Paleta de colores pasteles personalizados
colors = {
    "mint": "#A3E4D7",
    "pink": "#FADBD8",
    "lavender": "#D7BDE2",
    "lemon": "#F9E79F"
}
```

Usé distintos colores en las gráficas para que sea más fácil entender los resultados. Por ejemplo, un color muestra cuando algo salió bien y otro, cuando no funcionó. Así, con solo ver los colores, cualquiera puede captar rápidamente lo que está pasando, sin necesidad de leer tanto.

,

Parte #3: PARÁMETROS Y SIMULACIÓN

[24]:
```
p_exito = 0.6
n_UEs = 10**6
resultados = []
intentos_totales = []
```

Establecí una probabilidad de éxito del 60% (p_exito = 0.6) para cada intento. Se simularán un millón de usuarios (n_UEs = 10**6) y, para cada uno, se registrará si logró tener éxito (estado 3) o si falló (estado 4), usando la variable resultados. Además, guardé en intentos_totales cuántas veces necesitó intentar cada usuario antes de obtener un resultado final.

Lógica de la simulación

[25]:
```
for _ in range(n_UEs):
    estado = 0
    intentos = 1
    while estado not in [3, 4]:
        if np.random.rand() < p_exito:
            estado = 3
        else:
            if estado == 0:
                estado = 1
            elif estado == 1:
                estado = 2
            elif estado == 2:
                estado = 4
            intentos += 1
    resultados.append(estado)
    intentos_totales.append(intentos)
```

Simulé el comportamiento de cada usuario de forma individual, siguiendo una lógica paso a paso: todos comienzan en el estado 0 (primer intento), y en cada intento se genera un número aleatorio entre 0 y 1. Si ese número es menor que p_exito (0.6), se considera que el intento fue exitoso y el usuario pasa al estado 3. Si no, avanza al siguiente intento: estado 1 (segundo), luego estado

2 (tercero), y si tampoco tiene éxito, se considera un fracaso total y pasa al estado 4. También se registra cuántos intentos necesitó cada usuario antes de alcanzar el éxito o fracasar. Esta parte de la simulación representa el comportamiento aleatorio de cada usuario dentro de una subtrama, especialmente en relación con el momento en que llegan e intentan acceder.

,

Parte #4: Conversión a arrays para análisis estadístico

[26]:
```
resultados = np.array(resultados)
intentos_totales = np.array(intentos_totales)
```

Convierto las listas a arreglos de Numpy para poder trabajar de forma más eficiente. Esto me permite hacer análisis estadísticos y generar gráficas de manera más rápida y precisa, aprovechando todas las ventajas que ofrece Numpy en cuanto a velocidad y manejo de datos.

,

Parte #5: Cálculo de métricas clave

[27]:
```
p_exito_estimado = np.mean(resultados == 3)
media_intentos = np.mean(intentos_totales)
```

Calculé dos métricas clave: p_exito_estimado, que representa la proporción de usuarios que lograron tener éxito, y media_intentos, que indica cuántos intentos, en promedio, necesitó cada usuario. Estos cálculos cumplen con los pasos 1 y 3 de la consigna, ya que permiten analizar los resultados obtenidos en la simulación y compararlos con lo que se esperaba teóricamente.

,

Parte #6: Distribución final de estados (éxito vs. fracaso)

[28]:
```
distribucion_final = {
    "Éxito": np.mean(resultados == 3),
    "Fracaso": np.mean(resultados == 4)
}
```

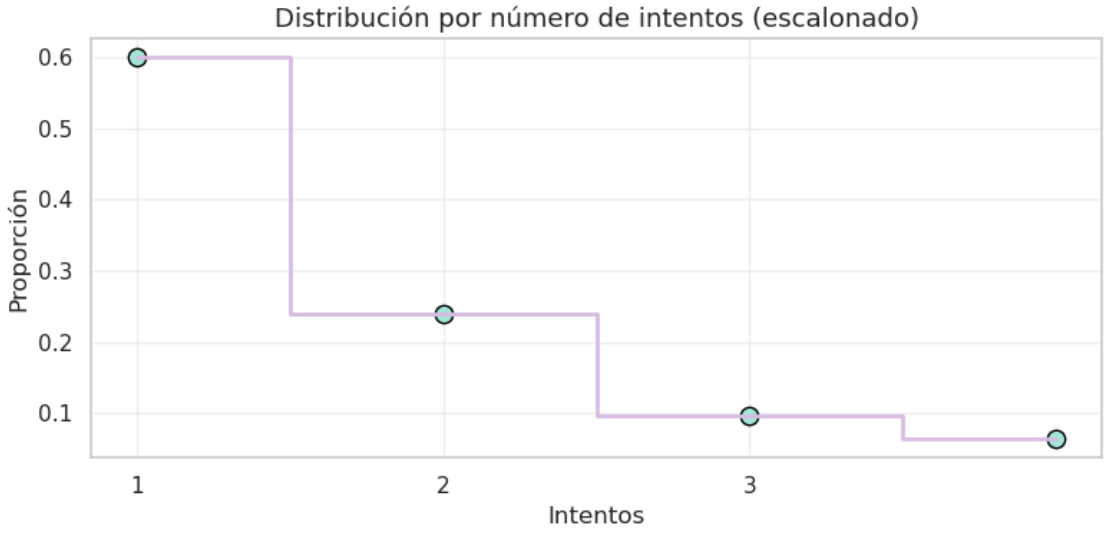
Guardo la proporción final de usuarios que lograron el éxito (estado 3) y de los que fallaron (estado 4). Estos valores se utilizan en la segunda gráfica para representar visualmente cómo se distribuyeron los resultados al final de la simulación

,

Parte #7: Distribución escalonada por número de intentos

[30]:
```
plt.figure(figsize=(8, 4))
valores, cuentas = np.unique(intentos_totales, return_counts=True)
plt.step(valores, cuentas / n_UEs, where='mid', color=colors["lavender"],␣
 ↪linewidth=2, label="Proporción")
plt.scatter(valores, cuentas / n_UEs, color=colors["mint"], s=80,␣
 ↪edgecolor="black")
plt.xticks([1, 2, 3])
```

```
plt.title("Distribución por número de intentos (escalonado)", fontsize=13)
plt.xlabel("Intentos")
plt.ylabel("Proporción")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

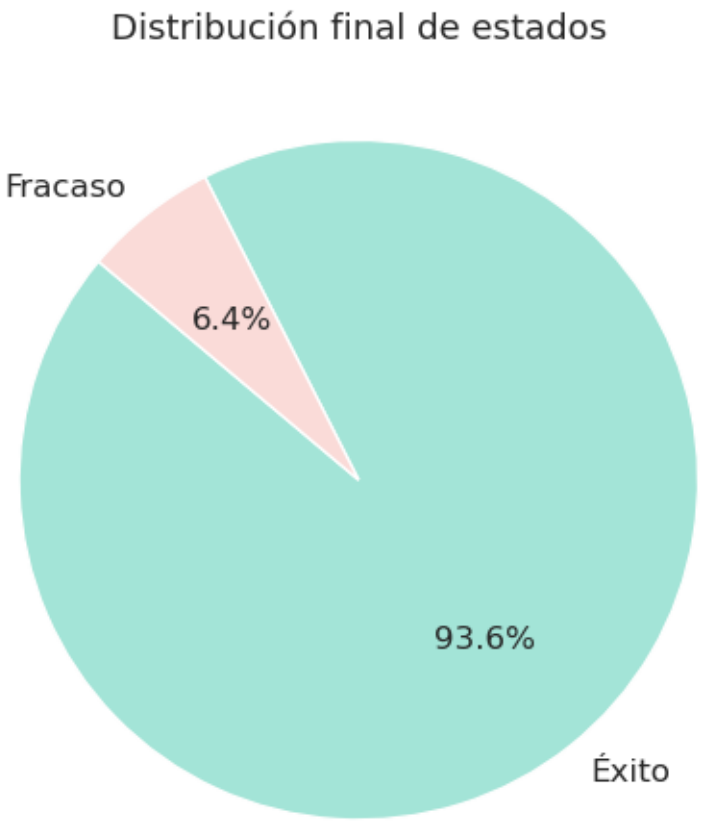


La gráfica muestra la proporción de usuarios que lograron el éxito en el primer, segundo o tercer intento. Para mayor claridad, se representa con una línea escalonada acompañada de puntos en color menta. Esta visualización permite identificar con facilidad en qué intento la mayoría de los usuarios logra acceder, lo cual forma parte del análisis empírico correspondiente al paso 1 de la consigna.

,

Parte #8: Gráfica 2: Gráfico de pastel de estados finales

[15]:
```
plt.figure(figsize=(5, 5))
plt.pie(distribucion_final.values(),
        labels=distribucion_final.keys(),
        autopct='%1.1f%%',
        colors=[colors["mint"], colors["pink"]],
        startangle=140,
        textprops={'fontsize': 12})
plt.title("Distribución final de estados", fontsize=13)
plt.tight_layout()
plt.show()
```



Este gráfico de pastel representa visualmente la proporción de usuarios que tuvieron éxito y los que no. Es útil porque permite ver de forma rápida y sencilla que aproximadamente el 60% logró el éxito, tal como se había definido al principio. De esta manera, refuerza visualmente los resultados almacenados en distribucion_final.

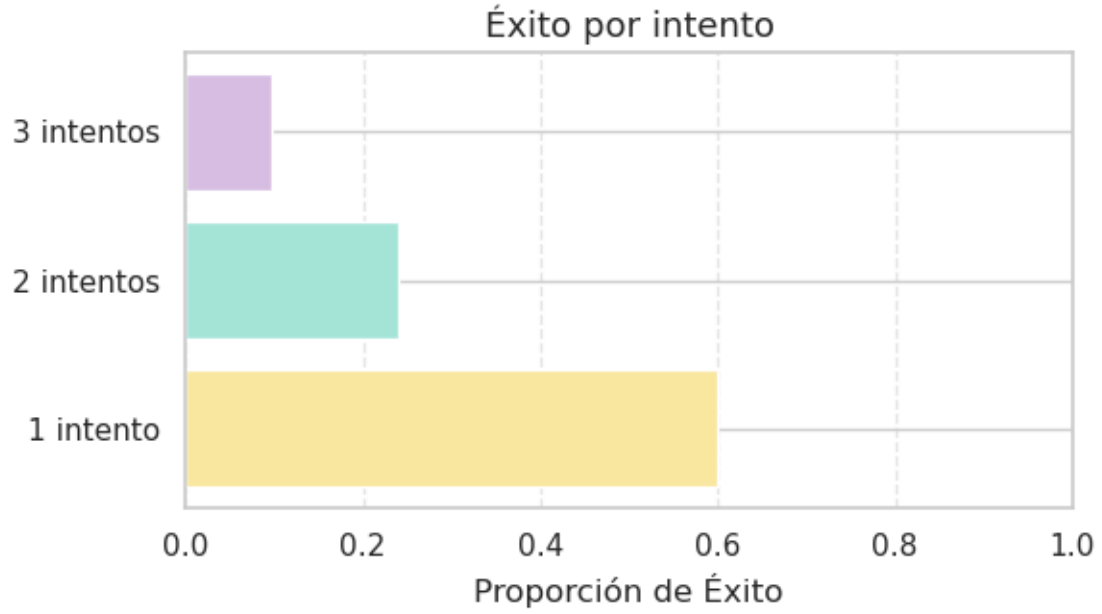
,

Parte #9: Barras horizontales de éxito por intento

[31]:
```
# Contamos cuántos usuarios lograron éxito en cada intento
exito_por_intento = {
    "1 intento": np.mean((resultados == 3) & (intentos_totales == 1)),
    "2 intentos": np.mean((resultados == 3) & (intentos_totales == 2)),
    "3 intentos": np.mean((resultados == 3) & (intentos_totales == 3))
}
```

Aquí se calcula qué proporción de éxitos ocurrieron en cada intento específico (1ro, 2do, 3ro). Luego se grafican con:

[32]:
```
plt.figure(figsize=(6, 3.5))
plt.barh(list(exito_por_intento.keys()), list(exito_por_intento.values()),␣
 ↪color=[colors["lemon"], colors["mint"], colors["lavender"]])
plt.xlabel("Proporción de Éxito")
plt.title("Éxito por intento", fontsize=13)
plt.xlim(0, 1)
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



Este gráfico descompone los casos de éxito según el intento en que se lograron, lo que permite analizar con mayor detalle la eficiencia del proceso. Además, deja ver cómo la probabilidad de éxito disminuye a medida que aumentan los intentos, resaltando que los primeros intentos son clave para lograr buenos resultados.

,

Parte #10: Relación con la paradoja del cumpleaños y la colisión Aunque este código no simula directamente las colisiones de preámbulos como lo plantea la fórmula del enunciado, se puede establecer una analogía con la paradoja del cumpleaños: cuando muchos usuarios intentan acceder al mismo tiempo, aumenta la probabilidad de que escojan el mismo recurso, tal como sucede cuando varias personas pueden compartir fecha de cumpleaños. Este fenómeno se refleja en la cantidad de intentos fallidos observados en la simulación, lo cual sugiere la existencia de colisiones. Por eso, este análisis responde al paso 2 y a la reflexión final del enunciado, conectando con la interpretación probabilística planteada.

,

Conclusión Este experimento de simulación por Monte Carlo reproduce de forma clara y completa el proceso de acceso aleatorio de un sistema con múltiples intentos. Con un millón de usuarios simulados y una probabilidad de éxito definida, se generan resultados muy representativos que permiten:

- Estimar empíricamente la distribución de intentos.
- Visualizar la proporción de éxito y fracaso.
- Entender cómo evoluciona el sistema.
- Conectar con modelos probabilísticos como Poisson o la paradoja del cumpleaños.
- Cada parte del código contribuye a construir una visión completa del sistema, permitiendo tanto la exploración teórica como práctica del fenómeno.