Universidad de Costa Rica
Escuela de Matemáticas
Herramientas de Ciencias de Datos II
CA-0305

Reporte - Proyecto Individual

Análisis y probabilidades en juegos de bingo 5x5

Profesor
Luis Alberto Juárez Potoy

Estudiante
Luis Fernando Amey Apuy

I - 2024

Introducción

La curiosidad se fomenta en los espacios de aburrimiento, al menos así he probado ciertos elementos a lo largo de varios eventos. Este proyecto se inspiró en un juego normal de bingo, pero en especial en lograr optimizar y hallar esperanzas claves para el bingo. Por ejemplo, dado un caso general, que se estén jugando por completar filas y columnas, la esperanza de bolitas que tengan que salir antes de que alguien cante bingo es importante para saber la duración media del juego, que va a ser diferente a un tablero completo.

Variaciones existen en el bingo para sus combinaciones en especial, como hacer una X, diagonales, las cuales se combinan normalmente con filas y columnas, una O, o todas las orillas, entre otros. Usualmente se basaría en observar los diferentes métodos y combinaciones para lograr captar estas diferencias, y así poder sacar la mayor parte de información. También se basó este proyecto ya que hay muchas herramientas para otros juegos, por ejemplo póquer, o la lotería, que calculan probabilidades específicas muy detalladamente pero para el bingo parece un espacio ambiguo. Aunque complejo, se pueden hacer simulaciones para poder encontrar puntos clave.

Conceptos principales

Hay muchas variaciones de los tableros como tal del bingo. Existen los tableros 3x9, donde ya vienen casillas tapadas, pero el más famoso es el 5x5, donde se tapa la casilla inicial o se otorga al jugador y se parte desde ahí. Esta casilla "otorgada" es la más importante, no solo a nivel del jugador, pero a nivel de probabilidad. En efecto, es la que complica el cálculo explícito de la misma, para todos los casos. El tablero 5x5 ha servido mucho de inspiración para hacer dinámicas que impliquen el mismo tablero pero no con números aleatorios, pero con la competencia de eventos, por ejemplo, establecer cierto tipo de retos en cada casilla y tratar de completar una fila/columna antes que el rival.

Dejando eso de lado, podemos hablar un poco sobre el tablero 5x5. Usualmente se intentan hacer "combinaciones" cuando han salido números aleatorios y coinciden con un patrón dado. Por ejemplo, en la Figura 1, podemos presenciar visualmente que han salido varios números y han realizado combinaciones horizontales (en la primer fila), verticales (en la tercer columna) y en una diagonal (la diagonal izquierda inferior - derecha superior), pero eso no implica que hayan ya salido varios números en el juego de bingo. Como máximo, han podido ya salir 62 bolitas, y este no sea un tablero ganador en los muchos que existan.

Tablero #1				
B	I	N	G	O
6	23	42	50	72
8	16	35	59	70
5	22	X	51	65
7	29	41	56	61
4	26	31	48	73

Figura 1: Tablero de ejemplo

Por otro lado, podemos ver que los números en el tablero están primeramente divididos entre las letras B-I-N-G-O, cada una con un rango de 15 números normalmente. Hay juegos de bingo que se realizan con 80 números, otros con 50, pero en promedio y se usa más el rango de 75 números en total, 15 por columna. Esto resulta interesante puesto que al aumentar el rango de los números, aumentamos también el porcentaje de "fallo" de que un tablero contenga un número salido. Por ejemplo, si tuviéramos un rango de 25, los únicos números que podrían fallar son los de la columna N, puesto ahí se encuentra la casilla premarcada. Esta es otra de las razones por la cual el cálculo de la probabilidad se vuelve extenso; esta casilla premarcada divide la probabilidad en dos y lo vuelve un problema combinatorio.

También es importante definir el número de tableros. Usualmente los juegos se vuelven más extensos conforme a más tableros hayan, puesto que aumenta la probabilidad de que al menos un tablero consiga una combinación en juego con un número más reducido de bolitas que hayan salido. Por eso mismo también es lógico que entre más tableros hayan, más aburrido o abrumador se siente el juego, porque esa probabilidad de ganar se va disminuyendo y repartiendo entre todos los tableros, y al mismo tiempo la esperanza de bolitas va disminuyendo porque estos nuevos tableros aumentan las coincidencias. Por eso, para fines de no lucro, se intenta realizar juegos locales y pequeños, para que todos puedan tener un premio.

Definir una probabilidad para partir sería excelente. Note que en la Figura 1, solo se pueden marcar 24 casillas, y hay 75 bolitas disponibles. Desde los casos posibles entre casos totales, podemos armar una probabilidad básica pero potente, puesto que va a aparecer en todos lados; 24/75.

Desarrollo del código

La idea del código es lograr correr simulaciones para así obtener ciertas esperanzas que a nivel matemático sería bastante difícil. Se puede empezar por crear el objeto de Tablero(). Las librerías usadas para hacer este proyecto es *numpy*, aunque no hay ninguna operación matricial y se pudo hacer perfectamente con matrices normales y *random*, para hacer los tableros y escogencias aleatorias.

Clase Tablero()

Requisitos del constructor (atributos):

- **rango : entero**
  El rango de números del tablero.

Atributos:

- **tablero : matriz**
  Matriz que tiene los números del tablero. Vacía inicialmente.
- **marcado : matriz**
  Matriz booleana que indica si un número está marcado.
- **anterior : booleano**
  Un indicador para saber si el número reciente coincidió.

Métodos:

- **generar()**, genera nuevos números de un tablero.
- **marcar(num)**, marca un número *num* en el tablero. Si lo encuentra, pone el atributo anterior como **True**, viceversa.
- **revisar(tipo)**, donde los tipos pueden ser Horizontal, Vertical, Diagonal, I, H, O, X, Z, N, T o Full. Cada uno de estos revisa la combinación respectiva.

La idea de un tablero es que pueda generar() números a través de un rango deseado, sin limitar las veces en que pueda generar un tablero. Usualmente se esquivó tener un método específico para poner un tablero físico, puesto que con la función set se puede hacer normalmente, mientras sea un *numpy matrix*.

La idea detrás de la doble matriz es, primero, conservar la estructura de matrices, porque perfectamente se podrían agarrar arreglos y puede que sea más eficiente (pero no para revisar combinaciones), y segundo para poder rastrear cada marcado minuciosamente. Al menos sin booleanos lo único que se me ocurre es ir eliminando los números de la matriz para marcar tableros, aunque se complicaría mucho el lado de revisar combinaciones. Entonces para marcar(), solo nos fijamos en la casilla alterna a la matriz. Esta función de revisar() puede revisar varias cosas, mientras se corra varias veces con el diferente tipo.

Después de crear cada tablero por separado, los podemos juntar en un juego ficticio, con el objeto Bingo(). Se pueden adjuntar la cantidad de tableros deseados, siendo realista, y se define de acá el rango de los tableros y el tipo de combinaciones que vamos a hacer en el juego, o como lo decimos, revisiones, porque revisamos cada tablero cada vez que se marca un número.

Clase Bingo()

Requisitos del constructor (atributos):

- **num : entero**
  El número de tableros para generar.
- **rango : entero**
  El rango de números de los tableros.
- **revisiones : lista**
  Los tipos de revisiones de la función revisar()

Atributos:

- **tableros : lista**
  Lista que va a contener a todos los tableros
- **salidos : lista**
  Lista que va a contener a los números que han salido

Métodos:

- **construir()**, construye y genera todos los tableros.
- **reset()**, olvida las marcas de los tableros y los números salidos.
- **revisar(imprimir)**, revisa todos los tableros según el atributo revisiones.
- **num(numero)**, marca en todos los tableros el número dado.
- **num_random(imprimir)**, marca un número aleatorio dentro del rango que no haya salido aún.
- **simular_juego(imprimir)**, simula un juego de bingo hasta que un tablero gane con las condiciones en el atributo revisiones.

Si el usuario desea imprimir información, entonces coloca imprimir con **True**

La idea de no construir() automáticamente es para poder estar seguros antes de construir el juego, puesto que si son bastantes tableros, entonces es mejor asegurar la construcción. Se pudo hacer en el constructor del objeto pero lo hallé mejor tener así los objetos para trabajar ordenadamente. Además, si se desea hacer un juego con el mismo set de tableros se puede hacer un reset() de los marcados. Por otro lado, acá podemos revisar() todos los tableros a la vez, para estar revisando constantemente cuando marquemos un número específico con num() o uno aleatorio con num_random(), mientras que ninguno haya ya salido, porque nos interesa los que quedan en la tómbola.

Igual fuera el caso podemos cambiarle el tablero a alguno de este objeto para hacer una simulación en vivo con una cantidad aproximada de tableros en juego y el propio como el que se cambió, la mayor inspiración de este proyecto, el cual se ejecutará en la presentación del mismo, realizando un juego completo en vivo con tableros físicos generados por *numpy*. Poder tener una aplicación a mano no sólo facilita información, pero comodidad para no estar marcando manualmente y hacer proyecciones al mismo tiempo, consigo a las verificaciones del tablero más rápidas.

Clase Simulaciones()

Requisitos del constructor (atributos):

- **sim : entero**
  La cantidad de simulaciones a realizar
- **rango : entero**
  El rango de números de los tableros.
- **num : entero**
  El número de tableros para generar.

Métodos:

- **mean_salidos()**, simula la media de tableros que marcan.
- **mean_normal()**, simula la media de bolitas para que ganen según las combinaciones horizontal, vertical y diagonal.
- **prob_mas_1()**, simula la probabilidad de que ganen 2 o más tableros.
- **mean_bolita()**, simula la media de marcados por bolita.
- **prob_cond()**, simula la probabilidad condicional de marcados dado que ya salieron una cantidad de bolitas.
- **mean_especifico(imprimir)**, lo mismo que mean_normal(), solo que ahora el usuario puede escoger las combinaciones.
- **prob_win()**, aproxima la probabilidad de ganar al siguiente turno.

Poder realizar simulaciones para poder estimar la cantidad promedio de números/bolitas que deben de salir para que algún tablero se posicione como ganador es de gran ayuda, no sólo al dar la cantidad, pero al demostrar que muchas probabilidades y esperanzas complicadas, como lo pueden ser las discretas y las casuísticas (sea el caso de la casilla premarcada en el centro), pueden ser resueltas fácilmente con simulaciones estocásticas o aleatorias, como la dada.

Al menos se nota un gran vacío en encontrar fuentes de información para este tema, y saber que se puede calcular por medio de simulaciones, y tener el dato específico, que por ejemplo, los resultados más directos al usar mean_normal() es de 20 números y al usar mean_especifico() con *Full* es de 60 aproximadamente, se puede denotar un espacio entre casillas y números salidos más definido.

Conclusiones

Resultados

- La probabilidad inicial de que un tablero marque un número salido, 24/75
  Al menos pensé que no iba a salir en varios casos pero se mantuvo constante por muchos métodos e inclusive podría servir para el cálculo explícito.

- La esperanza de números salidos para que gane un juego normal, 20 aprox.
  El juego real de bingo que inspiró el proyecto observaba una media similar, así que es coherente.

- La esperanza de números salidos para que gane un juego *Full*, 60 aprox
  De igual forma sucedió esto, así que coincide con el dato simulado.

- La esperanza para que se ganen juegos de variaciones, 55 aprox
  Lo más interesante es que las demás combinaciones tenían un alto número de bolitas para que se terminaran los juegos. No siendo lo mismo al tablero lleno, pero diferenciándose de unas pocos números salidos.

- La independencia de las probabilidades
  Hasta el final del proyecto caí en cuenta que las probabilidades de los números marcados en el tablero no afecta que otros tableros lo marquen, por lo que cada bolita va a tener un número similar de tableros que coinciden.

- Las probabilidades condicionales
  También hasta el final corregí un error de las probabilidades condicionales, para donde pudimos ver todas las posibles combinaciones de números salidos y números marcados. Se pueden ver bastantes patrones y disminuciones notables.

Recomendaciones

- Al dejar las probabilidades condicionales se pueden sacar fácilmente las esperanzas condicionales y otros aspectos. Esta matriz es de mucha utilidad pues también sirve para sacar la probabilidad de ganar individualmente, considerando la cantidad de tableros.

- Hay varios problemas de optimización al cargar muchos objetos. Para mayores simulaciones recomendaría una gran disminución de reiteraciones y reciclar objetos a la hora de hacer pruebas.

- Hay datos que no saqué puesto toman mucho tiempo, como la probabilidad matemática real, aunque hay datos fáciles, como remarqué por todo el proyecto, esa casilla premarcada es la definitiva que arruina el cálculo matemático.