Instituto Politécnico Nacional

Escuela Superior de Cómputo.

“Información sobre códigos

Cadenas de Markov

Random walk ”

Programa Académico: Licenciatura en Ciencia de Datos.

Unidad de aprendizaje: Procesos Estocasticos.

Cadenas de Markov

Código 1

Definimos una matriz de probabilidades de transición de una cadena de markov con 3 estados , donde los elementos corresponden a las probabilidades de transición de un estado a otro.

Luego, obtenemos la transpuesta de dicha matriz con el fin de poder crear un diagrama de la cadena de markov

Mandamos a llamar la librería (diagram) para crear un diagrama visual de la matriz de probabilidad de transición inversa que explica:

- El estado 2 se puede alcanzar directamente desde el estado 1
- Los estados 2 y 3 son accesibles entre si
- El estado 1 es accesibles desde el estado dos a través del estado 3
- La cadena tiene bucles por tanto se puede acceder a los estados en un paso
- Por lo tanto, los tres estados se comunican, y así la cadena tiene una sola clase y es irreducible.

Se calcula la matriz de probabilidad de transición de tres pasos (tm3) elevando la matriz original a la potencia 3.

Después en “init.p” definimos la probabilidad de comenzar en cualquiera de los 3 estados

La siguiente instrucción nos realiza el proceo de calculo de la distribución estacionaria de probabilidad después de 3 pasos

Las siguientes líneas el procesamiento de lo anteriormente mencionando usando funciones para llegar al valor de acceso de la línea anterior

Las siguientes peticiones nos piden las recurrentes de las clases , la transitorios de la clases,

La absorción de los estados, el periodo de la clase y por ultimo el resultado del cálculo de la distribución estacionaria de probabilidad después de 3 pasos

Código 2

Definimos una matriz de probabilidad de transición de una cadena de markov de 6 estados, donde los elementos corresponden a las probabilidades de transición de un estado a otro.

Luego, obtenemos la transpuesta de dicha matriz con el fin de poder crear un diagrama de la cadena de markov

Creamos un diagrama visual de la matriz de probabilidad de transición inversa que explica:

- Los estados 3 y 4 son reflejantes, por tanto transitorios.
- La cadena transitara fuera de los estados 3 y 4 y entrara a los estados recurrentes {1, 2} o {5, 6}.
- Los estados 3 y 4 tienen períodos infinitos
- Las clases recurrentes son aperiódicas debido a la existencia de los bucles.

Definimos un objeto de clase “markovchain” con la matriz de tansicion definida por tm, y el nombre de los estados

Solicitamos cuales son los estados recurrentes y los transitorios.

Posteriormente calculamos el perido de cada subcadena irreducible (las clases recurrentes).

Y por último el resultado del cálculo de la distribución estacionaria de probabilidad

Código 3

Simulación 1.1 step 1

Definimos una matriz de probabilidades de transición de una cadena de markov con 3 estados , donde los elementos corresponden a las probabilidades de transición de un estado a otro.

Definimos un objeto de clase “markovchain” con la matriz de tansicion definida por tm, y el nombre de los estados

Difinimos el numero total de pasos a simular

Después en “p0” definimos la probabilidad de comenzar en cualquiera de los 3 estados

En la variable “MC.states” se crea una matriox para guardar las trayectorias simuladas

Después generamos un semilla que generara números aleatorios

Se utiliza un bucle for para simular dos trayectorias de la cadena de Markov. Para cada trayectoria, se selecciona un estado inicial aleatoriamente según la distribución de probabilidad inicial p0 y luego se simula la trayectoria utilizando la función rmarkovchain() del paquete markovchain. La simulación de la trayectoria comienza desde el estado inicial t0 y se incluye en la matriz de estados MC.states usando la columna i.

Y luego ploteamos los resultados obtenidos de cada una de ellas

Simulación 1.1 step 2

El código define una función que simula una trayectoria de una cadena de Markov a partir de una matriz de transición y una probabilidad inicial.

Se define una función llamada MC que simula una trayectoria de una cadena de Markov con matriz de transición tm, vector de probabilidades iniciales p0, y número de pasos nsteps.

Se crea un vector states de longitud nsteps para almacenar los estados de la cadena de Markov en cada paso.

Se selecciona el estado inicial de la cadena de Markov de forma aleatoria según las probabilidades iniciales p0.

Se selecciona la fila de la matriz de transición correspondiente al estado anterior de la cadena de Markov, lo que representa las probabilidades de transición desde ese estado a los demás estados.

Se devuelve el vector states que contiene los estados de la cadena de Markov en cada paso.

Se establece una semilla para que los resultados sean reproducibles.

Se crea una matriz MC.states2 de nsteps filas y 2 columnas para almacenar dos trayectorias de la cadena de Markov simuladas.

Se inicia un bucle que simula dos trayectorias de la cadena de Markov y las almacena en la matriz MC.states2. En cada iteración se selecciona un estado inicial de forma aleatoria según las probabilidades iniciales p0, y se llama a la función MC para simular la trayectoria de la cadena de Markov. El resultado se almacena en la columna j de la matriz MC.states2.

Y luego ploteamos los resultados obtenidos de cada una de ellas

Simulación 1.1 step 3

Ahora calculamos iterativamente los vectores de probabilidad y los representamos gráficamente en función del número de pasos.

Podemos ver en el grafico por medio de pasos el resultado del cálculo de la distribución estacionaria de probabilidad y como se mantiene

Codigo 4

Simulación 1.2 step 1

Definimos una matriz de probabilidad de transición de una cadena de markov de 6 estados, donde los elementos corresponden a las probabilidades de transición de un estado a otro.

Difinimos el numero total de pasos a simular

Después en “p0” definimos la probabilidad de comenzar en cualquiera de los 6 estados

Definimos un objeto de clase “markovchain” con la matriz de tansicion definida por tm, y el nombre de los estados

Se utiliza un bucle for para simular dos trayectorias de la cadena de Markov. Para cada trayectoria, se selecciona un estado inicial aleatoriamente según la distribución de probabilidad inicial p0 y luego se simula la trayectoria utilizando la función rmarkovchain() del paquete markovchain. La simulación de la trayectoria comienza desde el estado inicial t0 y se incluye en la matriz de estados MC.states usando la columna i.

Y luego ploteamos los resultados obtenidos de cada una de ellas

Simulación 1.2 step 2

El programa sirve para presentarnos la trayectoria dependiendo del punto inicial del estado , en este caso haciendo referencia a la probabilidad del camino de trayectoria por cada uno de los estados.

Código 5

Aplication 1.1 step 1

Resultado del cálculo de la distribución estacionaria de probabilidad para cuando aparezcan consonantes y vocales

Aplication 1.1 step 2

El código está diseñado para realizar un análisis de texto en el archivo "Onegin.txt". Primero, el texto se lee en el programa con la función "read_file" y luego se limpia el texto de caracteres innecesarios usando la función "gsub". El texto se convierte en minúsculas, se eliminan los espacios en blanco, los saltos de línea, la puntuación y los caracteres de suavidad en el idioma ruso.

Luego, se crean dos listas de vocales y consonantes en el idioma ruso. El programa itera a través de cada carácter en el texto y verifica si es una vocal o una consonante. También se verifica si hay combinaciones de dos vocales, dos consonantes, una vocal y una consonante en el texto.

Finalmente, se calcula la suma del número de vocales, consonantes y combinaciones de dos vocales, dos consonantes, una vocal y una consonante.

Codigo 6

Especificacion general

El modelo mendeliano de herencia genética en humanos establece que un rasgo genético específico está determinado por un par de genes que pueden ser de tres tipos: AA, Aa o aa. Durante la reproducción, un descendiente hereda un gen del par de cada progenitor, y los genes se seleccionan al azar e independientemente entre sí. Supongamos que el gen a es un gen mutante o que una persona portadora de este gen tiene una enfermedad. Consideremos el genotipo de los descendientes en sucesivas generaciones si el segundo progenitor siempre tiene el genotipo AA. Esto se puede presentar como una cadena de Markov con un espacio de estados S = {AA, Aa, aa} y una matriz de probabilidad de transición. Si los padres tienen los genes (AA, AA), entonces su descendencia tendrá genes AA con probabilidad 1. Si los padres tienen los genes (Aa, AA), es igualmente probable que su descendencia tenga genes AA o Aa. Finalmente, si los padres tienen los genes (aa, AA), su descendencia con certeza tendrá genes Aa

Aplication 1.3 step 1

Definimos una matriz de probabilidades de transición de una cadena de markov con 3 estados , donde los elementos corresponden a las probabilidades de transición de un estado a otro.

Luego, obtenemos la transpuesta de dicha matriz con el fin de poder crear un diagrama de la cadena de markov

Creamos un diagrama visual de la matriz de probabilidad de transición inversa que explica:

- Se observa que el estado Aa y aa son estados transitorios
- AA es el estado absorvente
- A largo plaxo el gen a desaparecerá de la población , lo justificamo calculando la distribuvion estacionaria

Calculamos de la distribución estacionaria de probabilidad para Aa,AA,aa

Después generacmos un código para calcular las combinaciones cuando el gen a esta presente el 1% , dejandonos con la pAA del 99% y pAa del 0% y paa =1%, como valores de condición inicial

Notemos que apartir de la generación 2 el gen aa se pierde , generando un hibrido Aa que es el que permanece hasta la 10 generacion

Código 7

El programa analiza un conjunto de datos meteorológicos de Los Ángeles y categoriza los tipos de clima en "clear", "clouds", "fog" y "rain". Luego, utiliza la función "Lag" del paquete Hmisc para crear tres nuevas columnas que registran el tipo de clima en los tres días anteriores. A continuación, el programa calcula la probabilidad de que el clima del tercer día sea "clear", "clouds", "fog" o "rain" en función del clima en los tres días anteriores. Para ello, se utilizan una serie de condicionales y se cuentan los casos en que se cumple cada combinación de climas en los tres días anteriores.

Random Walk

Codigo 1 Simulacion 2.1

El código simula y grafica tres trayectorias de un caminante aleatorio simple con probabilidad de moverse a la derecha igual a 0.6 en cada paso. Primero, se definen las variables ntraj, p y nsteps. Luego, se genera una semilla aleatoria para garantizar la reproducibilidad de los resultados. A continuación, se define la matriz "walk" para almacenar los pasos del caminante aleatorio. Después, se utilizan dos bucles for para simular las trayectorias del caminante aleatorio. Finalmente, se grafican las tres trayectorias junto con una cuadrícula en el fondo de la figura.

Composición de bucle for:

Se realiza un bucle "for" para simular las trayectorias. Primero se establece que la posición inicial para cada trayectoria es 0 (en la primera fila de la matriz "walk"). Luego se utiliza otro bucle "for" para generar los pasos de la trayectoria. Para cada paso i en la trayectoria j, se simula un número aleatorio entre 0 y 1 con "runif(1)", y se compara con la probabilidad "p". Si el número aleatorio es menor que "p", se avanza una unidad hacia la derecha desde la posición anterior (fila i-1, misma columna j) sumando 1 a la posición anterior. Si el número aleatorio es mayor o igual que "p", se avanza una unidad hacia la izquierda restando 1 a la posición anterior.

Codigo 2 Simulacion 2.2

Este código simula una caminata aleatoria en dos dimensiones en la que se parte de un punto (0,0) y en cada paso se mueve una unidad en una dirección aleatoria: arriba, abajo, izquierda o derecha. El número de pasos que se dan en la caminata está determinado por la variable nsteps. Al final se grafica la trayectoria de la caminata, comenzando con un punto verde en el origen y terminando con un punto rojo en el último punto de la caminata.

Simulación de trayectorias explicación :

Se simulan las trayectorias del caminante aleatorio utilizando un bucle for. En cada iteración, se actualiza la posición del caminante sumando una fila aleatoria de la matriz rstep a su posición actual, almacenando las nuevas coordenadas (x,y) en la matriz walk.

Codigo 3 Simulacion 2.3

Este código genera una simulación de un paseo aleatorio en tres dimensiones, es decir, una serie de movimientos al azar en un espacio tridimensional. El número de pasos y la semilla aleatoria se especifican al principio del código, y se definen las coordenadas del punto de partida en el origen

(0,0,0). Luego se definen los posibles pasos aleatorios en las tres dimensiones y se ejecuta un bucle para generar una trayectoria de paseo aleatorio. Finalmente, se muestra la trayectoria en un gráfico 3D con puntos que indican el punto de partida y el punto final.

Codigo 4 Aplicaciones 2.1

Descripcion general:

El juego de azar es quizás la aplicación más antigua de las caminatas aleatorias, y el problema más famoso es el problema de la ruina del jugador. Una versión del problema de la ruina del jugador fue formulada tan temprano como en 1656, en la correspondencia entre Blaise Pascal y Pierre de Fermat.

Supongamos que un jugador comienza con una fortuna de $i y avanzará $1 con probabilidad p o retrocederá $1 con probabilidad q = 1 - p hasta que esté en bancarrota o alcance la fortuna de $N. ¿Cuál es la probabilidad de que quede en bancarrota?

Para responder a esta pregunta, calcularemos la probabilidad complementaria de alcanzar la fortuna de $N.

Step 1

El código simula el problema de ruina del jugador y estima la probabilidad de que un jugador que comienza con una fortuna de $i, que puede subir $1 con probabilidad p o bajar $1 con probabilidad q=1-p, llegue a una fortuna de $N o se arruine (alcance $0). Primero se especifican los parámetros p, i, N y ntraj, que controlan la probabilidad de subir y bajar, la fortuna inicial, la fortuna objetivo y el número de simulaciones. Luego se define un vector "walk" que almacenará las fortunas en cada paso. Se establecen contadores para el número de veces que se alcanza la fortuna objetivo, el número de veces que se arruina y el número total de juegos jugados. Después se establece una semilla para asegurar la reproducibilidad de los resultados y se realiza la simulación del juego en sí. En cada simulación, el jugador comienza con la fortuna inicial y se mueve hacia arriba o hacia abajo dependiendo de una probabilidad de éxito p. Se realiza esto hasta que el jugador llegue a la fortuna objetivo o se arruine. Si llega a la fortuna objetivo, se aumenta el contador de "nNs"; si se arruina, se aumenta el contador de "nzeros". El número total de juegos jugados también se incrementa en cada simulación. Finalmente, se calculan las probabilidades de alcanzar la fortuna objetivo o de arruinarse, y se estima el número promedio de juegos jugados por simulación

Prob ns – probabilidad de llegar a la fortuna

Prob nz – probabilidad de estar en banca rota

Pron mean games – probabilidad media de juegos para ganar o perder todo

Step 2

Este código calcula la probabilidad de ruina en una apuesta simple en la que el jugador comienza con una cantidad inicial de dinero "i" y apuesta una fracción "p" de su dinero en cada ronda. La apuesta se detiene cuando el jugador alcanza un cierto objetivo "N" o pierde todo su dinero.

El código utiliza la fórmula matemática para la probabilidad de ruina, que depende de la fracción de apuesta "p", el objetivo "N" y la cantidad inicial de dinero "i". Se calcula la probabilidad de que el jugador pierda todo su dinero antes de alcanzar el objetivo "N" y la probabilidad de que alcance el objetivo "N" antes de quedarse sin dinero. Estas probabilidades se almacenan en un vector "p.ruin".

Luego, el código grafica las probabilidades de ruina y de éxito (alcanzar el objetivo "N") en función de la fracción de apuesta "p". Se utiliza la función "plot" para trazar la curva de probabilidad de ruina en rojo y la curva de probabilidad de éxito en verde. También se agrega una leyenda para indicar qué línea representa cada probabilidad.

Forma simple :

El código calcula la probabilidad de ruina en una apuesta simple y la representa en un gráfico en función de la fracción de apuesta "p". Utiliza una fórmula matemática para calcular la probabilidad de perder todo el dinero antes de alcanzar un objetivo "N". Luego, traza la curva de probabilidad de ruina en rojo y la curva de probabilidad de éxito en verde en un gráfico. También agrega una leyenda para indicar qué línea representa cada probabilidad.

Step 3

Este código calcula el número esperado de juegos que un jugador necesitará para alcanzar una meta o perder todo su dinero en una apuesta simple. La apuesta se hace en fracciones de la cantidad total de dinero del jugador, y el objetivo es alcanzar una cantidad específica de dinero, N, antes de perder todo su dinero inicial, i.

La fórmula matemática utilizada para calcular el número esperado de juegos depende de la fracción de apuesta, p, la cantidad inicial de dinero, i, y la cantidad objetivo de dinero, N. Si p=0.5, lo que significa que el jugador está apostando la misma cantidad de dinero en cada ronda, la fórmula se simplifica a i*(N-i).

Si p no es igual a 0.5, la fórmula completa se utiliza para calcular el número esperado de juegos. Los resultados se almacenan en el vector "E.ngames".

Luego se grafica el número esperado de juegos en función de la fracción de apuesta "p". La función "plot" se utiliza para trazar la curva del número esperado de juegos en verde. También se agrega una leyenda para indicar qué línea representa el número esperado de juegos.

Forma simple

Este código calcula el número esperado de juegos en una apuesta simple, donde el jugador apuesta fracciones de su dinero para alcanzar un objetivo específico antes de perder todo su dinero inicial. La fórmula matemática depende de la fracción de apuesta, la cantidad inicial de dinero y el objetivo, y se almacenan los resultados en un vector. Luego se grafica la curva del número esperado de juegos en función de la fracción de apuesta utilizando la función "plot", con una leyenda para identificar la línea correspondiente.

Código 5 Aplicaciones 2.2

Forma simple

El código calcula el tiempo de espera esperado para que un usuario abandone un sitio web en particular utilizando una matriz de probabilidad de transición y la probabilidad de que el usuario esté en cada página después de un número determinado de iteraciones.