Trabajo Práctico 2

Unidad 3 - Conocimiento
Unidad 4 - Conocimiento incierto

Problema 1 - Caballeros

Objetivo

Escribe un programa para resolver los acertijos de lógica planteados.

Introducción

En 1978, el lógico Raymond Smullyan publicó "¿Cómo se llama este libro?", un libro de acertijos lógicos. Entre los acertijos del libro había una clase de acertijos que Smullyan llamó acertijos de "Caballeros y bribones".

En un acertijo de Caballeros y bribones, se proporciona la siguiente información: Cada personaje es un caballero o un bribón. Un caballero siempre dirá la verdad: si el caballero dice una sentencia, entonces esa sentencia es verdadera. Por el contrario, un bribón siempre mentirá: si un bribón dice una frase, entonces esa frase es falsa.

El objetivo del acertijo es, dado un conjunto de frases pronunciadas por cada uno de los personajes, determinar, para cada personaje, si ese personaje es un caballero o un bribón.

Por ejemplo, considere un acertijo simple con un solo personaje llamado A. A dice "Soy a la vez un caballero y un bribón".

Lógicamente, podríamos razonar que si A fuera un caballero, entonces esa sentencia tendría que ser verdadera. Pero sabemos que la frase no puede ser verdadera, porque A no puede ser al mismo tiempo caballero y bribón; sabemos que cada personaje es caballero o bribón, pero no ambos. Entonces, podríamos concluir que A debe ser un bribón.

Ese acertijo era más simple. ¡Con más personajes y más frases, los acertijos pueden volverse más complicados! Su tarea en este problema es determinar cómo representar estos acertijos utilizando lógica proposicional, de modo que una IA que ejecute un algoritmo de verificación de modelos pueda resolver estos acertijos por nosotros.

Descargar el código: `problema1-caballeros.zip`.

Explicación del problema

Eche un vistazo a logic.py, que quizás recuerde de las clases. No es necesario comprender todo lo que contiene este archivo, pero observe que este archivo define varias clases para diferentes tipos de conectivos lógicos. Estas clases se pueden componer entre sí, por lo que

una expresión como And(Not(A), Or(B, C))representa la sentencia lógica que establece que el símbolo Ano es verdadero y que el símbolo Bo símbolo Ces verdadero (donde .º.ªquí se refiere a inclusivo, no exclusivo, o).

Recuerde que logic.py también contiene una función model_check. model_check requiere una base de conocimientos y una consulta. La base de conocimiento es una única sentencia lógica: si se conocen varias sentencias lógicas, se pueden unir en una Andexpresión. model_check considera recursivamente todos los modelos posibles y devuelve True si la base de conocimientos implica la consulta y devuelve Falseen caso contrario.

Ahora, eche un vistazo a puzzle.py. En la parte superior, hemos definido seis símbolos proposicionales. AKnight, por ejemplo, representa la sentencia que .ª es un caballero", mientras que AKnave representa la sentencia que .ª es un bribón". También hemos definido símbolos proposicionales de manera similar para los caracteres B y C.

Lo que sigue son cuatro bases de conocimiento diferentes, knowledge0, knowledge1, knowledge2y knowledge3, que contendrán el conocimiento necesario para deducir las soluciones de los próximos acertijos 0, 1, 2 y 3, respectivamente. Observe que, por ahora, cada una de estas bases de conocimiento está vacía. ¡Ahí es donde entras tú!

La función main de esto puzzle.py recorre todos los acertijos y utiliza la verificación de modelos para calcular, dado el conocimiento de ese acertijo, si cada personaje es un caballero o un bribón, imprimiendo cualquier conclusión que el algoritmo de verificación de modelos pueda sacar.

Especificación

Agregue conocimientos a las bases de conocimientos **knowledge0**, **knowledge1**, **knowledge2** y **knowledge3** resuelva los siguientes acertijos.

1. El acertijo 0 es el acertijo del fondo. Contiene un solo carácter, A.
   - A dice: "Soy a la vez un caballero y un bribón".
2. El acertijo 1 tiene dos personajes: A y B.
   - A dice: .ªmbos somos bribones".
   - B no dice nada.
3. El acertijo 2 tiene dos personajes: A y B.
   - A dice "Somos del mismo tipo".
   - B dice: "Somos de diferentes tipos".
4. El acertijo 3 tiene tres personajes: A, B y C.

- A dice "Soy un caballero.º "Soy un bribón", pero no sabes cuál.
- B dice .ª dijo 'Soy un bribón'".
- B luego dice Ç es un bribón".
- C dice .ª es un caballero".

En cada uno de los acertijos anteriores, cada personaje es un caballero o un bribón. Cada frase pronunciada por un caballero es verdadera y cada frase pronunciada por un bribón es falsa.

Una vez que haya completado la base de conocimientos para un problema, debería poder ejecutar python puzzle.py para ver la solución al acertijo.

- Para cada base de conocimiento, es probable que desee codificar dos tipos diferentes de información: (1) información sobre la estructura del problema en sí (es decir, información proporcionada en la definición de un acertijo de Caballero y Sota), e (2) información sobre lo que realmente dijeron los personajes.
- Considere lo que significa si un personaje pronuncia una sentencia. ¿En qué condiciones es cierta esa frase? ¿En qué condiciones esa frase es falsa? ¿Cómo puedes expresar eso como una sentencia lógica?
- Hay múltiples bases de conocimiento posibles para cada acertijo que calcularán el resultado correcto. Debe intentar elegir una base de conocimientos que ofrezca la traducción más directa de la información del acertijo, en lugar de realizar un razonamiento lógico por su cuenta. También debes considerar cuál sería la representación más concisa de la información del acertijo.
  - Por ejemplo, para el acertijo 0, la configuración knowledge0 = AKnavedaría como resultado un resultado correcto, ya que a través de nuestro propio razonamiento sabemos que A debe ser un bribón. Pero hacerlo iría en contra del espíritu de este problema: el objetivo es que su IA razone por usted.
- No debería necesitar (ni debería) modificar logic.py nada para completar este problema.

Problema 2 - Buscaminas

Objetivo

Escribe una IA para jugar al Buscaminas

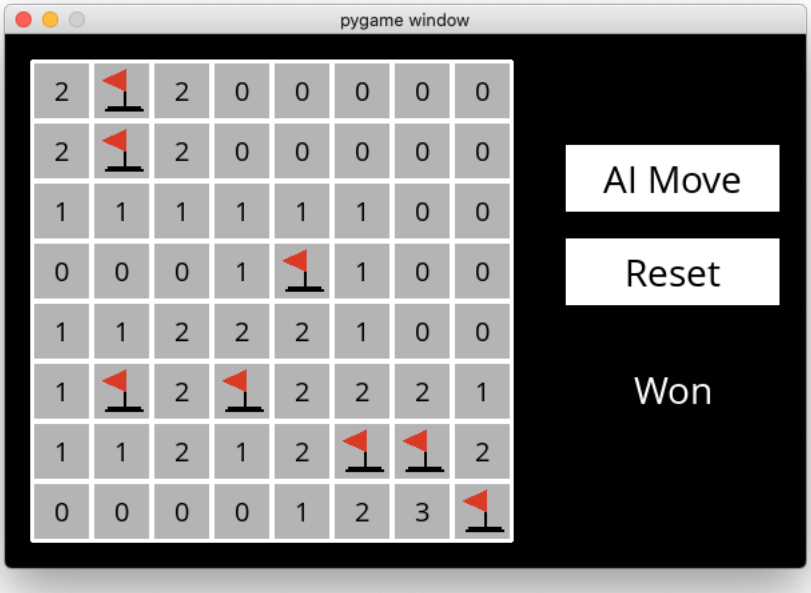

Figura 1

Introducción

Buscaminas es un juego que consta de una cuadrícula de celdas, donde algunas de las celdas contienen "minas.ºcultas. Al hacer clic en una celda que contiene una mina, la mina detona y hace que el usuario pierda el juego. Al hacer clic en una celda "segura" (es decir, una celda que no contiene una mina) se revela un número que indica cuántas celdas vecinas (donde un vecino es una celda que está un cuadrado a la izquierda, derecha, arriba, abajo o diagonal desde la celda dada: contiene una mina.

En este juego Buscaminas 3x3, por ejemplo, los tres 1valores indican que cada una de esas celdas tiene una celda vecina que es una mina. Los cuatro valores 0 indican que cada una de esas celdas no tiene una mina vecina.

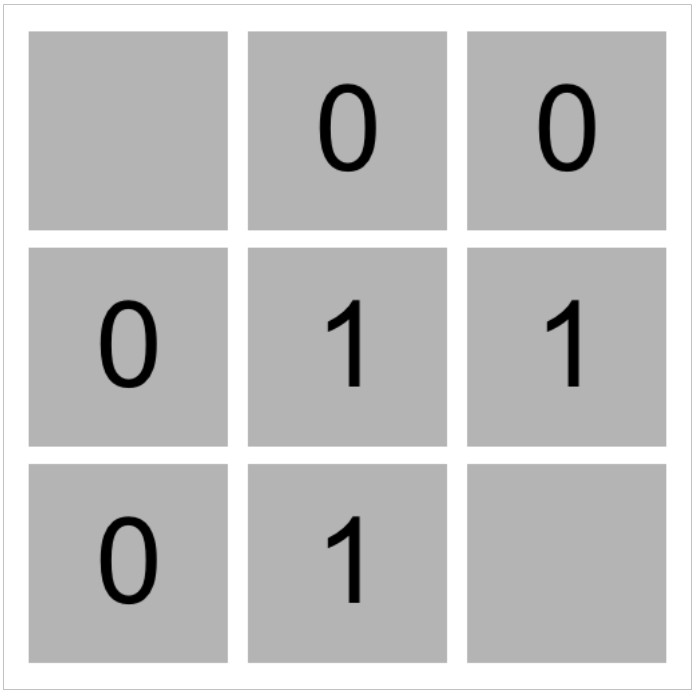

Figura 2

Dada esta información, un jugador lógico podría concluir que debe haber una mina en la celda inferior derecha y que no hay ninguna mina en la celda superior izquierda, porque sólo en ese caso las etiquetas numéricas en cada una de las otras celdas serían preciso.

El objetivo del juego es marcar (es decir, identificar) cada una de las minas. En muchas implementaciones del juego, incluida la de este proyecto, el jugador puede marcar una mina haciendo clic derecho en una celda (o haciendo clic con dos dedos, según la computadora).

Lógica proposicional

El objetivo es construir una IA que pueda jugar al buscaminas. Recordar que los agentes basados en el conocimiento toman decisiones basadas en su base de conocimiento y haciendo inferencias basadas en este conocimiento.

Una forma de representar el conocimiento de una IA sobre un juego Buscaminas es haciendo de cada celda una variable proposicional que sea verdadera si la celda contiene una mina y falsa en caso contrario.

¿A qué información tiene acceso la IA? Bueno, la IA sabría cada vez que se hace clic en una celda segura y podría ver el número de esa celda. Considere el siguiente tablero del Buscaminas, donde se reveló la celda del medio y las otras celdas se etiquetaron con una letra de identificación para facilitar la discusión.

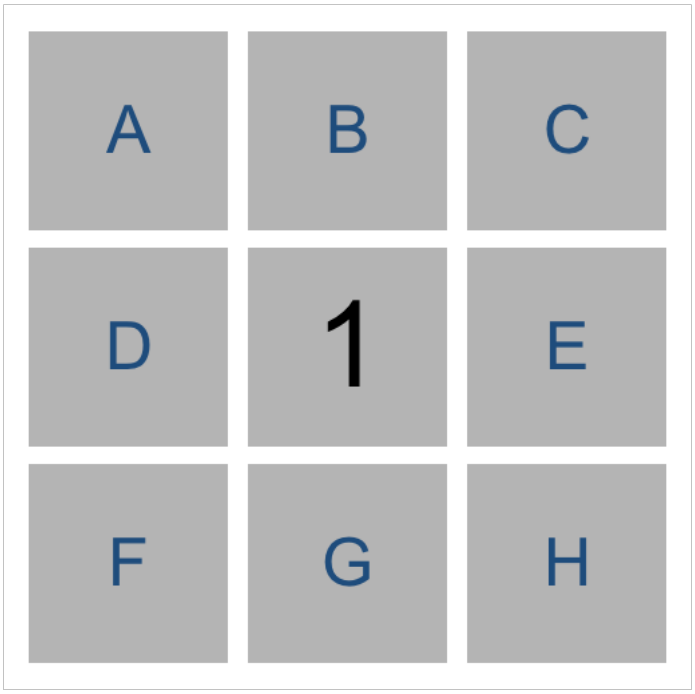

Figura 3

¿Qué información tenemos ahora? Parece que ahora sabemos que una de las ocho celdas vecinas es una mina. Por lo tanto, podríamos escribir una expresión lógica como la siguiente para indicar que una de las celdas vecinas es una mina.

¿Qué información tenemos ahora? Parece que ahora sabemos que una de las ocho celdas vecinas es una mina. Por lo tanto, podríamos escribir una expresión lógica como la siguiente para indicar que una de las celdas vecinas es una mina.

```
    Or(A, B, C, D, E, F, G, H)

```

Pero en realidad sabemos más de lo que dice esta expresión. La sentencia lógica anterior expresa la idea de que al menos una de esas ocho variables es verdadera. Pero podemos hacer una afirmación más contundente: sabemos que exactamente una de las ocho variables es verdadera. Esto nos da una sentencia de lógica proposicional como la siguiente.

```
  Or(
    And(A, Not(B), Not(C), Not(D), Not(E), Not(F), Not(G), Not(H)),
    And(Not(A), B, Not(C), Not(D), Not(E), Not(F), Not(G), Not(H)),
    And(Not(A), Not(B), C, Not(D), Not(E), Not(F), Not(G), Not(H)),
    And(Not(A), Not(B), Not(C), D, Not(E), Not(F), Not(G), Not(H)),
    And(Not(A), Not(B), Not(C), Not(D), E, Not(F), Not(G), Not(H)),
    And(Not(A), Not(B), Not(C), Not(D), Not(E), F, Not(G), Not(H)),
    And(Not(A), Not(B), Not(C), Not(D), Not(E), Not(F), G, Not(H)),
    And(Not(A), Not(B), Not(C), Not(D), Not(E), Not(F), Not(G), H)
  )

```

¡Esa es una expresión bastante complicada! Y eso es sólo para expresar lo que significa que una célula tenga un 1interior. Si una celda tiene 2o 3algún otro valor, la expresión podría ser incluso más larga.

Intentar realizar una verificación de modelos en este tipo de problema también rápidamente se volvería intratable: en una cuadrícula de 8x8, el tamaño que Microsoft usa para su nivel Principiante, tendríamos 64 variables y, por lo tanto, 2^{64} modelos posibles para verificar. demasiados para que una computadora pueda calcularlos en un período de tiempo razonable. Necesitamos una mejor representación del conocimiento para este problema.

Representacion del conocimiento

En cambio, representaremos cada sentencia del conocimiento de nuestra IA como se muestra a continuación.

```
{A, B, C, D, E, F, G, H} = 1

```

Cada sentencia lógica en esta representación tiene dos partes: un conjunto de **cells** en el tablero que están involucrados en la sentencia y un número **count**, que representa el recuento de cuántas de esas celdas son minas. La sentencia lógica anterior dice que de las celdas A, B, C, D, E, F, G y H, exactamente 1 de ellas es una mina.

¿Por qué es esta una representación útil? En parte, se presta bien a ciertos tipos de inferencias. Considere el juego a continuación.

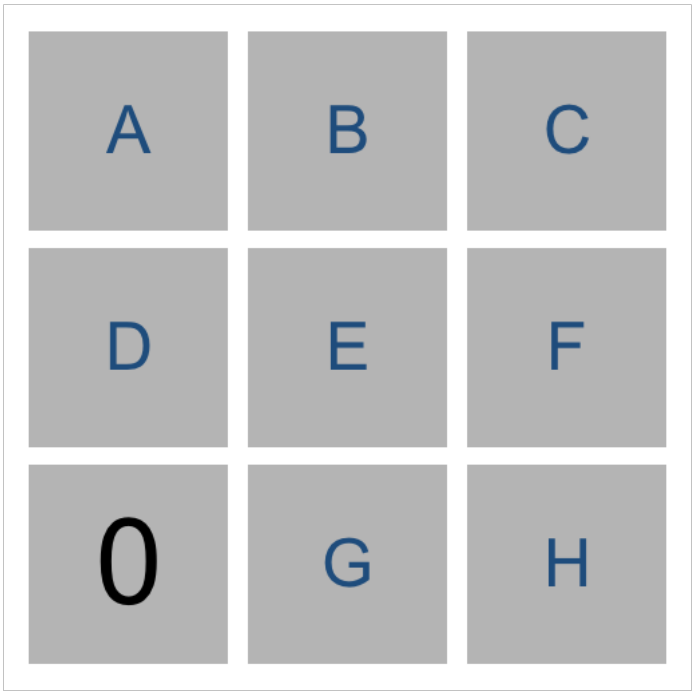

Figura 4

Usando el conocimiento del número inferior izquierdo, podríamos construir la sentencia D, E, G = 0para que signifique que de las celdas D, E y G, exactamente 0 de ellas son mías. Intuitivamente, podemos inferir de esa frase que todas las celdas deben estar a salvo. Por extensión, cada vez que tenemos una sentencia cuyo **count** es 0, sabemos que todas esas sentencias **cells** deben ser seguras.

De manera similar, considere el juego a continuación.

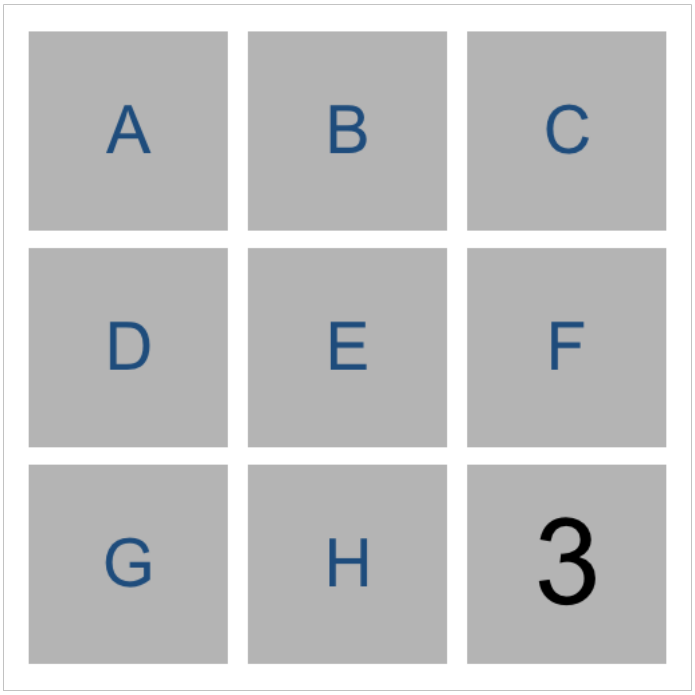

Figura 5

Nuestra IA construiría la frase E, F, H = 3. Intuitivamente, podemos inferir que todos E, F y H son míos. De manera más general, cada vez que el número de **cells** es igual a **count**, sabemos que todas esas sentencias **cells** deben ser mías.

En general, solo querremos que nuestras sentencias sean sobre objetos cellsque aún no se sabe que sean seguros o minas. Esto significa que, una vez que sabemos si una celda es mina o no, podemos actualizar nuestras sentencias para simplificarlas y potencialmente sacar nuevas conclusiones.

Por ejemplo, si nuestra IA conocía la frase A, B, C = 2, todavía no tenemos suficiente información para concluir nada. Pero si nos dijeran que C está a salvo, podríamos eliminarlo Cde la sentencia por completo, dejándonos con la sentencia A, B = 2(que, dicho sea de paso, nos permite sacar algunas conclusiones nuevas).

Del mismo modo, si nuestra IA conociera la sentencia A, B, C = 2 y nos dijeran que C es una mina, podríamos eliminar C de la sentencia y disminuir el valor de **count**(ya que C era

una mina que contribuyó a ese recuento), dándonos la sentencia A, B = 1. Esto es lógico: si dos de A, B y C son minas, y sabemos que C es una mina, entonces debe darse el caso de que de A y B, exactamente uno de ellos sea una mina.

Si somos aún más inteligentes, hay un último tipo de inferencia que podemos hacer.

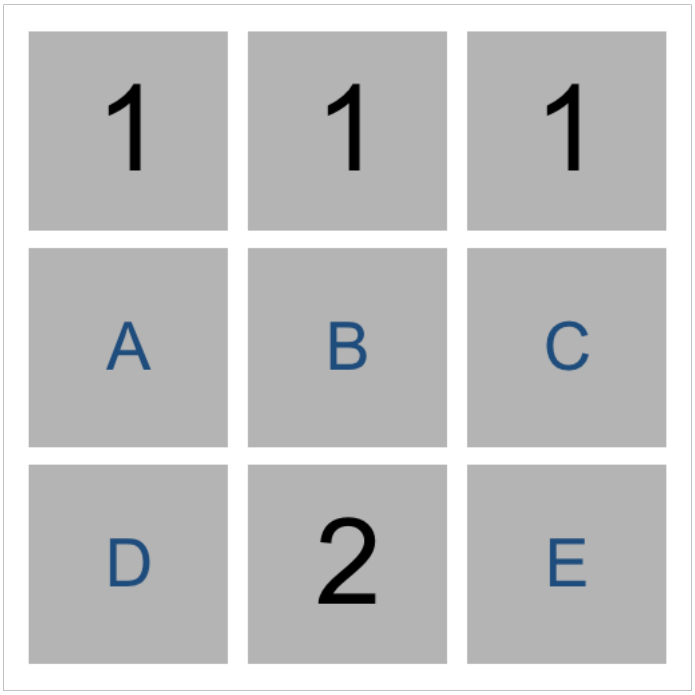

Figura 6

Considere solo las dos sentencias que nuestra IA conocería según la celda media superior y la celda media inferior. Desde la celda superior del medio, tenemos A, B, C = 1. Desde la celda central inferior, tenemos A, B, C, D, E = 2. Lógicamente, entonces podríamos inferir un nuevo conocimiento, que D, E = 1. Después de todo, si dos de A, B, C, D y E son míos, y sólo uno de A, B y C es mío, entonces es lógico que exactamente uno de D y E deba ser el otro mío.

De manera más general, cada vez que tengamos dos sentencias set1 = count1 y set2 = count2 donde set1 sea un subconjunto de set2, entonces podremos construir la nueva sentencia set2 - set1 = count2 - count1. Considere el ejemplo anterior para asegurarse de comprender por qué esto es cierto.

Entonces, usando este método de representación del conocimiento, podemos escribir un agente de IA que pueda recopilar conocimientos sobre el tablero Buscaminas y, con suerte, seleccionar celdas que sepa que son seguras.

Descargar el código: `problema2-buscaminas.zip`.
Una vez en el directorio del proyecto, ejecute `pip3 install -r requirements.txt` para instalar el paquete Python requerido (`pygame`) para este proyecto si aún no lo tiene instalado.

Explicación del problema

Hay dos archivos principales en este proyecto: `runner.py` y `minesweeper.py`. `minesweeper.py` contiene toda la lógica del juego en sí y para que la IA juegue. runner.pyse ha implementado para ti y contiene todo el código para ejecutar la interfaz gráfica del juego. Una vez que hayas completado todas las funciones requeridas en `minesweeper.py`, deberías poder correr python `runner.py` para jugar Buscaminas (o dejar que tu IA juegue por ti).

Abrámonos `minesweeper.py` para entender lo que se proporciona. Hay tres clases definidas en este archivo, Minesweeper que maneja el juego; Sentence, que representa una sentencia lógica que contiene tanto un conjunto de cells como un count; y MinesweeperAI, que se encarga de inferir qué movimientos realizar en función del conocimiento.

La clase Minesweeper ha sido completamente implementada para usted. Observe que cada celda es un par (i, j)donde iestá el número de fila (que va desde 0 hasta height - 1) y j el número de columna (que va desde 0 hasta width - 1).

La clase Sentence se utilizará para representar sentencias lógicas de la forma descrita en Antecedentes. Cada sentencia tiene un conjunto de cells dentro y un número count de cuántas de esas celdas son mías. La clase también contiene funciones known_mines y known_safes para determinar si se sabe que alguna de las celdas de la sentencia es mina o es segura. También contiene funciones mark_mine y mark_safe para poder actualizar una sentencia en respuesta a nueva información sobre una celda.

Finalmente, la clase MinesweeperAI implementará una IA que pueda jugar al Buscaminas. La clase de IA realiza un seguimiento de una serie de valores. self.moves_made contiene un conjunto de todas las celdas en las que ya se ha hecho clic, por lo que la IA sabe que no debe volver a seleccionarlas. self.mines contiene un conjunto de todas las celdas que se sabe que son minas. self.safes contiene un conjunto de todas las celdas que se sabe que son seguras. Y self.knowledge contiene una lista de todos los Sentence que la IA sabe que son ciertos.

La función mark_mine agrega una celda self.mines para que la IA sepa que es una mina. También recorre todas las sentencias en la IA **knowledge** e informa a cada sentencia que la celda es una mina, de modo que la sentencia pueda actualizarse en consecuencia si contiene información sobre esa mina. La función mark_safe hace lo mismo, pero para celdas seguras.

¡Las funciones restantes, add_knowledge, make_safe_move y make_random_move, quedan a tu elección!

Especificación

Complete las implementaciones de la clase Sentence y la clase MinesweeperAI en minesweeper.py.

En la clase **Sentence**, complete las implementaciones de known_mines, known_safes, mark_mine y mark_safe.

- La función ‘known_mines‘ debe devolver un conjunto de todas las celdas que se sabe que son minas en ‘self.cells‘.
- La función ‘known_safes‘ debe devolver un conjunto de todas las celdas que se sabe que son seguras en ‘self.cells‘.
- La función ‘mark_mine‘ primero debe verificar si ‘cell‘ es una de las celdas incluidas en la sentencia.
  - Si ‘cell‘ está en la sentencia, la función debe actualizar la sentencia para que ‘cell‘ ya no esté en la sentencia, pero aún represente una sentencia lógicamente correcta dado que se sabe que ‘cell‘ es una mina.
  - Si ‘cell‘ no está en la sentencia, entonces no es necesaria ninguna acción.
- La función ‘mark_safe‘ primero debe verificar si ‘cell‘ es una de las celdas incluidas en la sentencia.
  - Si ‘cell‘ está en la sentencia, la función debe actualizar la sentencia para que ‘cell‘ ya no esté en la sentencia, pero aún represente una sentencia lógicamente correcta dado que se sabe que ‘cell‘ es segura.
  - Si ‘cell‘ no está en la sentencia, entonces no es necesaria ninguna acción.

En la clase MinesweeperAI, complete las implementaciones de add_knowledge, make_safe_move y make_random_move.

- `add_knowledge` debe aceptar un `cell` (representado como una tupla) `(i, j)` y su correspondiente `count`, y actualizar `self.mines`, `self.safes`, `self.moves_made`, y con cualquier información nueva que la IA pueda inferir, dado que se sabe que es una celda segura con minas vecinas. `self.safes`, `self.moves_made`, `self.knowledge`, `cell`, `count`
  - La función debe marcar el `cell` como uno de los movimientos realizados en el juego.
  - La función debe marcar el `cell` como una celda segura, actualizando también las sentencias que lo contengan.
  - La función debería agregar una nueva sentencia a la base de conocimientos de la IA, basada en el valor de `cell`’s `count`, para indicar que `count` los vecinos de `cell` son minas. Asegúrese de incluir únicamente celdas cuyo estado aún esté indeterminado en la sentencia.
  - Si, según cualquiera de las sentencias en `self.knowledge`, las nuevas celdas se pueden marcar como seguras o como minas, entonces la función debería hacerlo.

- Si, basándose en cualquiera de las sentencias en `self.knowledge`, se pueden inferir nuevas sentencias (usando el método de subconjunto descrito en Antecedentes), entonces esas sentencias también deben agregarse a la base de conocimientos.
- Tenga en cuenta que cada vez que realice algún cambio en el conocimiento de su IA, es posible que se realicen nuevas inferencias que antes no eran posibles. Asegúrese de que esas nuevas inferencias se agreguen a la base de conocimientos, si es posible hacerlo.

- `make_safe_move` debe devolver un movimiento `(i, j)` que se sabe que es seguro.

  - Se debe saber que el movimiento devuelto es seguro y no un movimiento ya realizado.
  - Si no se puede garantizar un movimiento seguro, la función debería regresar `None`.
  - La función no debe modificar `self.moves_made`, `self.mines`, `self.safes`, o `self.knowledge`.

- `make_random_move` debería devolver un movimiento aleatorio `(i, j)`.

  - Esta función se llamará si no es posible realizar un movimiento seguro: si la IA no sabe dónde moverse, elegirá moverse aleatoriamente.
  - El movimiento no debe ser un movimiento que ya se haya realizado.
  - El movimiento no debe ser un movimiento que se sepa que es una mina.
  - Si tales movimientos no son posibles, la función debería devolver `None`.

Consejos

- Asegúrese de haber leído detenidamente la sección Antecedentes para comprender cómo se representa el conocimiento en esta IA y cómo la IA puede hacer inferencias.
- Si no se siente cómodo con la programación orientada a objetos, puede encontrar útil la documentación de clases de Python.
- Puede encontrar algunas operaciones comunes de conjuntos en la documentación de Python sobre conjuntos.
- Al implementar ‘known_mines‘ y ‘known_safes‘ en la clase ‘Sentence‘, considere: ¿bajo qué circunstancias sabe con seguridad que las celdas de una sentencia son seguras? ¿En qué circunstancias sabe con seguridad que las celdas de una frase son minas?
- ‘add_knowledge‘ hace bastante trabajo y probablemente será, con diferencia, la función más larga que escriba para este proyecto. Probablemente será útil implementar el comportamiento de esta función paso a paso.
- Puedes agregar nuevos métodos a cualquiera de las clases si lo deseas, pero no debes modificar ninguna de las definiciones o argumentos de las funciones existentes.

- Cuando ejecutas tu IA (como al hacer clic en .ªI Move"), ¡ten en cuenta que no siempre ganará! Habrá algunos casos en los que la IA deberá adivinar porque carece de información suficiente para realizar un movimiento seguro. Esto es de esperar. 'runner.py' imprimirá si la IA está haciendo un movimiento que cree que es seguro o si está haciendo un movimiento aleatorio.
- Tenga cuidado de no modificar un conjunto mientras lo itera. ¡Hacerlo puede provocar errores!

Problema 3 - Ranking de páginas

Objetivo

Escribe una IA para armar un ranking de páginas web por importancia.

```
$ python pagerank.py corpus0
PageRank Results from Sampling (n = 10000)
1.html: 0.2223
2.html: 0.4303
3.html: 0.2145
4.html: 0.1329
PageRank Results from Iteration
1.html: 0.2202
2.html: 0.4289
3.html: 0.2202
4.html: 0.1307
```

Introducción

Cuando los motores de búsqueda como Google muestran resultados de búsqueda, lo hacen colocando páginas más "importantes" y de mayor calidad en los resultados de búsqueda más arriba que las páginas menos importantes. Pero, ¿cómo sabe el motor de búsqueda qué páginas son más importantes que otras?

Una heurística podría ser que una página "importante" es aquella a la que muchas otras páginas enlazan, ya que es razonable imaginar que más sitios enlazarán a una página web de mayor calidad que a una página web de menor calidad. Por lo tanto, podríamos imaginar un sistema en el que a cada página se le asigna una clasificación según la cantidad de enlaces entrantes que tiene desde otras páginas, y clasificaciones más altas indicarían una mayor importancia.

Pero esta definición no es perfecta: si alguien quiere que su página parezca más importante, entonces, bajo este sistema, podría simplemente crear muchas otras páginas que enlacen a la página deseada para inflar artificialmente su clasificación.

Por esa razón, el algoritmo PageRank fue creado por los cofundadores de Google (incluido Larry Page, que dio nombre al algoritmo). En el algoritmo de PageRank, un sitio web es más importante si está vinculado a otros sitios web importantes, y los enlaces de sitios web menos importantes tienen menos peso. Esta definición parece un poco circular, pero resulta que existen múltiples estrategias para calcular estas clasificaciones.

Modelo de usuario navegante web aleatorio

Una forma de pensar en el PageRank es con el modelo de navegante aleatorio, que considera el comportamiento de un internauta hipotético en Internet que hace clic en enlaces al azar. Considere el corpus de páginas web a continuación, donde una flecha entre dos páginas indica un enlace de una página a otra.

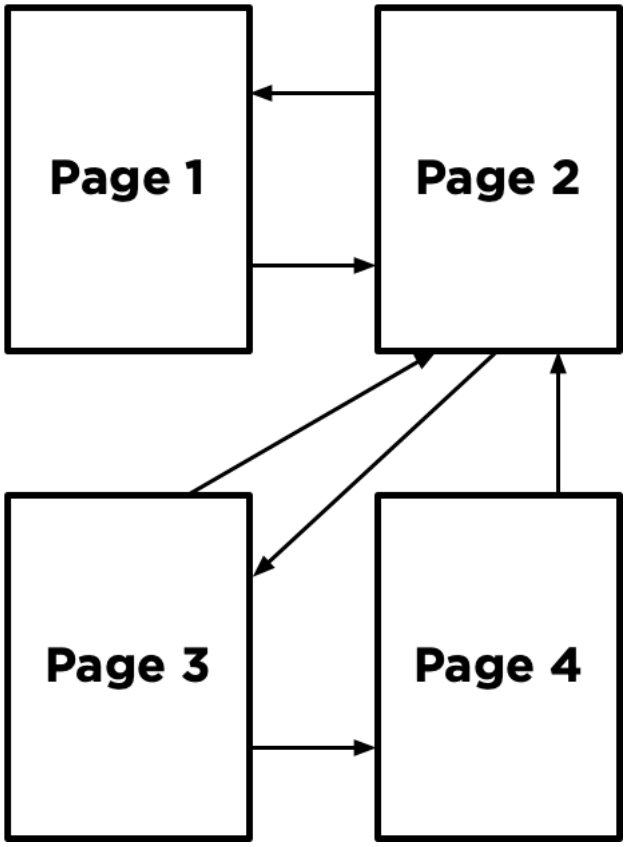

Figura 7

El modelo de navegante aleatorio imagina a un navegante que comienza con una página web al azar y luego elige aleatoriamente los enlaces a seguir. Si el internauta está en la página 2, por ejemplo, elegirá aleatoriamente entre la página 1 y la página 3 para visitar a continuación (los enlaces duplicados en la misma página se tratan como un solo enlace y los enlaces de una página a sí misma también se ignoran). . Si eligieran la página 3, el internauta elegiría aleatoriamente entre la página 2 y la página 4 para visitar a continuación.

El PageRank de una página, entonces, puede describirse como la probabilidad de que un navegante aleatorio esté en esa página en un momento dado. Después de todo, si hay más enlaces a una página en particular, entonces es más probable que un internauta aleatorio termine en esa página. Además, es más probable que se haga clic en un enlace de un sitio más importante que en un enlace de un sitio menos importante al que enlazan menos páginas, por lo que este modelo también maneja la ponderación de los enlaces según su importancia.

Una forma de interpretar este modelo es como una Cadena de Markov, donde cada página representa un estado y cada página tiene un modelo de transición que elige entre sus enlaces al azar. En cada paso de tiempo, el estado cambia a una de las páginas vinculadas por el estado actual.

Al muestrear estados aleatoriamente de la cadena de Markov, podemos obtener una estimación del PageRank de cada página. Podemos comenzar eligiendo una página al azar y luego seguir enlaces al azar, manteniendo un registro de cuántas veces hemos visitado cada página. Una vez que hayamos reunido todas nuestras muestras (basándonos en un número que elegimos de antemano), la proporción del tiempo que estuvimos en cada página podría ser una estimación de la clasificación de esa página.

Sin embargo, esta definición de PageRank resulta ligeramente problemática si consideramos una red de páginas como la siguiente.

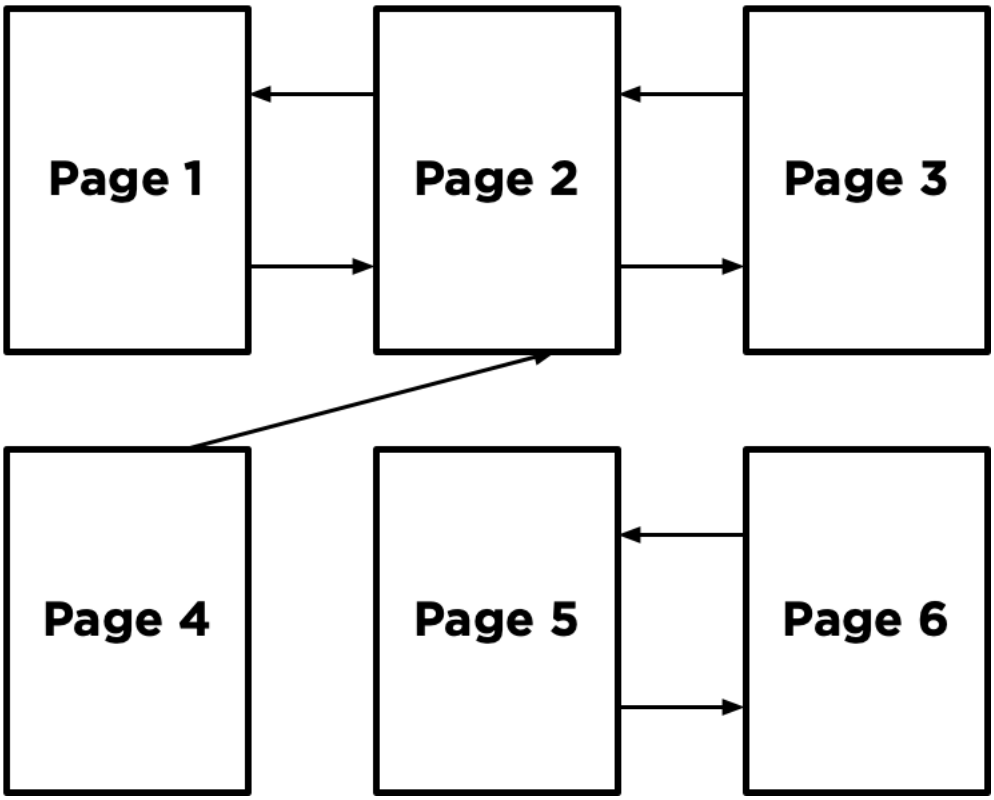

Figura 8

Imaginemos que comenzamos aleatoriamente tomando muestras de la página 5. Luego no tendríamos más remedio que ir a la página 6, y luego no tendríamos más remedio que ir a la página 5 después de eso, y luego a la página 6 nuevamente, y así sucesivamente. Terminaríamos con una estimación de 0,5 para el PageRank de las páginas 5 y 6, y una estimación de 0 para el PageRank de todas las páginas restantes, ya que pasamos todo nuestro tiempo en las páginas 5 y 6 y nunca visitamos ninguna de las otras páginas.

Para garantizar que siempre podamos llegar a algún otro lugar del corpus de páginas web, introduciremos en nuestro modelo un factor de amortiguación d. Con probabilidad d(donde dnormalmente se establece alrededor 0.85de), el navegante aleatorio elegirá uno de los enlaces de la página actual al azar. Pero de lo contrario (con probabilidad 1 - d), el navegante aleatorio elige una de todas las páginas del corpus al azar (incluida aquella en la que se encuentra actualmente).

Nuestro navegante aleatorio ahora comienza eligiendo una página al azar y luego, para cada muestra adicional que nos gustaría generar, elige un enlace de la página actual al azar con probabilidad y elige cualquier página al azar con dprobabilidad 1 - d. Si realizamos un seguimiento de cuántas veces ha aparecido cada página como muestra, podemos tratar la proporción de estados que se encontraban en una página determinada como su PageRank.

Algoritmo iterativo

También podemos definir el PageRank de una página usando una expresión matemática recursiva. Sea PR(p)el PageRank de una página determinada p: la probabilidad de que un navegante aleatorio termine en esa página. ¿ Cómo definimos PR(p)? Bueno, sabemos que hay dos formas en que un navegante aleatorio podría terminar en la página:

Con probabilidad 1 - d, el internauta eligió una página al azar y terminó en la página p. Con probabilidad d, el internauta siguió un enlace de una página ia otra p.

La primera condición es bastante sencilla de expresar matemáticamente: se 1 - d divide por N, donde N es el número total de páginas de todo el corpus. Esto se debe a que la probabilidad 1 - d de elegir una página al azar se divide equitativamente entre todas las N páginas posibles.

Para la segunda condición, debemos considerar cada página posible i que enlace a la página p. Para cada una de esas páginas entrantes, NumLinks(i)sea el número de enlaces en la página i. Cada página i a la que enlaza p tiene su propio PageRank, PR(i)que representa la probabilidad de que estemos en la página i en un momento dado. Y dado que desde la página i viajamos a cualquiera de los enlaces de esa página con la misma probabilidad, dividimos PR(i)por el número de enlaces NumLinks(i)para obtener la probabilidad de que estuviéramos en la página i y eligiéramos el enlace a la página p.

Esto nos da la siguiente definición del PageRank de una página p.

$$PR(p) = \frac{1-d}{N} + d\sum_{i} \frac{PR(i)}{NumLinks(i)}$$

Figura 9

En esta fórmula, des el factor de amortiguación, N es el número total de páginas en el

corpus, i abarca todas las páginas que enlazan con la página py NumLinks(i)es el número de enlaces presentes en la página i.

Entonces, ¿cómo haríamos para calcular los valores de PageRank para cada página? Podemos hacerlo mediante iteración: comience asumiendo que el PageRank de cada página es 1 / N(es decir, que tiene la misma probabilidad de estar en cualquier página). Luego, use la fórmula anterior para calcular nuevos valores de PageRank para cada página, según los valores de PageRank anteriores. Si seguimos repitiendo este proceso, calculando un nuevo conjunto de valores de PageRank para cada página en función del conjunto de valores de PageRank anterior, eventualmente los valores de PageRank convergerán (es decir, no cambiarán más que un pequeño umbral con cada iteración).

En este proyecto, implementará ambos enfoques para calcular el PageRank: calculará tomando muestras de páginas de un navegante aleatorio de la Cadena de Markov y aplicando iterativamente la fórmula de PageRank.

Descargar el código: `problema3-ranking-paginas.zip`.

Explicación del problema

Abre pagerank.py. Observe primero la definición de dos constantes en la parte superior del archivo: **DAMPING** representa el factor de amortiguación y se establece inicialmente en 0.85. **SAMPLES** representa la cantidad de muestras que usaremos para estimar el PageRank usando el método de muestreo, inicialmente establecido en 10,000 muestras.

Ahora, eche un vistazo a la función main. Espera un argumento de línea de comando, que será el nombre de un directorio de un corpus de páginas web para las que nos gustaría calcular PageRanks. La función crawl toma ese directorio, analiza todos los archivos HTML en el directorio y devuelve un diccionario que representa el corpus. Las claves de ese diccionario representan páginas (por ejemplo, "2.html") y los valores del diccionario son un conjunto de todas las páginas vinculadas por la clave (por ejemplo {"1.html", "3.html"}).

Luego, la función main llama a la función sample_pagerank, cuyo propósito es estimar el PageRank de cada página mediante muestreo. La función toma como argumentos el corpus de páginas generado por **crawl**, así como el factor de amortiguación y la cantidad de muestras a utilizar. En última instancia, sample_pagerank debería devolver un diccionario donde las claves sean el nombre de cada página y los valores sean el PageRank estimado de cada página (un número entre 0 y 1).

La función main también llama a la función iterate_pagerank, que también calculará el PageRank para cada página, pero utilizando el método de fórmula iterativa en lugar de mediante muestreo. Se espera que el valor de retorno esté en el mismo formato, y esperamos que la salida de estas dos funciones sea similar cuando se les proporciona el mismo corpus.

Especificación

Complete la implementación de transition_model, sample_pagerank y iterate_pagerank.

transition_model debería devolver un diccionario que represente la distribución de probabilidad sobre qué página visitaría a continuación un navegante aleatorio, dado un corpus de páginas, una página actual y un factor de amortiguación.

- La función acepta tres argumentos: **corpus**, **page**, y **damping_factor**.
  - *corpus* es un diccionario de Python que asigna el nombre de una página a un conjunto de todas las páginas vinculadas por esa página.
  - *page* es una cadena que representa en qué página se encuentra actualmente el navegante aleatorio.
  - *damping_factor* es un número de coma flotante que representa el factor de amortiguación que se utilizará al generar las probabilidades.
- El valor de retorno de la función debe ser un diccionario de Python con una clave para cada página del corpus. Cada clave debe asignarse a un valor que represente la probabilidad de que un internauta aleatorio elija esa página a continuación. Los valores en esta distribución de probabilidad devuelta deben sumar 1.
  - Con probabilidad *damping_factor*, el navegante aleatorio debería elegir aleatoriamente uno de los enlaces de *page* con igual probabilidad.
  - Con probabilidad $1 - damping\_factor$, el navegante aleatorio debería elegir aleatoriamente una de todas las páginas del corpus con igual probabilidad.
- Por ejemplo, si *corpus* fuera {"1.html": {"2.html", "3.html"}, "2.html": {"3.html"}, "3.html": {"2.html"}}, *page* fuera "1.htmlz *damping_factor* fuera 0.85, entonces la salida de *transition_model* debería ser {"1.html": 0.05, "2.html": 0.475, "3.html": 0.475}. Esto se debe a que con probabilidad 0.85, elegimos aleatoriamente pasar de la página 1 a la página 2 o 3 (por lo que cada página 2 o 3 tiene probabilidad 0.425 de comenzar), pero cada página obtiene una página adicional 0.05 porque con probabilidad 0.15 elegimos aleatoriamente entre todas las tres páginas.
- Si *page* no tiene enlaces salientes, entonces *transition_model* debería devolver una distribución de probabilidad que elija aleatoriamente entre todas las páginas con igual probabilidad. (En otras palabras, si una página no tiene enlaces, podemos pretender que tiene enlaces a todas las páginas del corpus, incluida ella misma).

La función sample_pagerank debe aceptar un corpus de páginas web, un factor de amortiguación y una cantidad de muestras, y devolver un PageRank estimado para cada página.

- La función acepta tres argumentos: **corpus**, **a damping_factory**, y **n**.
  - **corpus** es un diccionario de Python que asigna el nombre de una página a un conjunto de todas las páginas vinculadas por esa página.

- **damping_factory** es un número de coma flotante que representa el factor de amortiguación que utilizará el modelo de transición.
- **n** es un número entero que representa el número de muestras que se deben generar para estimar los valores de PageRank.

- El valor de retorno de la función debe ser un diccionario de Python con una clave para cada página del corpus. Cada clave debe asignarse a un valor que represente el PageRank estimado de esa página (es decir, la proporción de todas las muestras que correspondieron a esa página). Los valores de este diccionario deben sumar 1.
- La primera muestra debe generarse eligiendo de una página al azar.
- Para cada una de las muestras restantes, la siguiente muestra debe generarse a partir de la muestra anterior según el modelo de transición de la muestra anterior.
  - Probablemente querrás pasar la muestra anterior a tu **transition_model** función, junto con **corpus** y **damping_factor**, para obtener las probabilidades de la siguiente muestra.
  - Por ejemplo, si las probabilidades de transición son {"1.html": 0.05, "2.html": 0.475, "3.html": 0.475}, entonces el 5 % de las veces que la siguiente muestra generada debería ser "1.html", el 47,5 % de las veces que la siguiente muestra generada debería ser "2.html", y el 47,5 % de las veces que la siguiente muestra generada debería ser "3.html".
- Puedes suponer que **n** será al menos 1.

La función iterate_pagerank debe aceptar un corpus de páginas web y un factor de amortiguación, calcular PageRanks según la fórmula de iteración descrita anteriormente y devolver el PageRank de cada página con una precisión de 0.001.

- La función acepta dos argumentos: **corpus** y **damping_factor**.
  - **corpus** es un diccionario de Python que asigna el nombre de una página a un conjunto de todas las páginas vinculadas por esa página.
  - **damping_factor** es un número de coma flotante que representa el factor de amortiguación que se utilizará en la fórmula de PageRank.
- El valor de retorno de la función debe ser un diccionario de Python con una clave para cada página del corpus. Cada clave debe asignarse a un valor que represente el PageRank de esa página. Los valores de este diccionario deben sumar 1.
- La función debe comenzar asignando a cada página un rango de $1/N$, donde **N** es el número total de páginas del corpus.
- Luego, la función debería calcular repetidamente nuevos valores de clasificación basados en todos los valores de clasificación actuales, de acuerdo con la fórmula de PageRank en la sección .ªntecedentes". (es decir, calcular el PageRank de una página basándose en el PageRanks de todas las páginas que enlazan con ella).

- Una página que no tiene ningún enlace debe interpretarse como si tuviera un enlace para cada página del corpus (incluida ella misma).

- Este proceso debe repetirse hasta que ningún valor de PageRank cambie más que 0.001 entre los valores de clasificación actuales y los nuevos valores de clasificación.

No debe modificar nada más que pagerank.py, que tiene las tres funciones que la especificación requiere que implemente, aunque puede escribir funciones adicionales y/o importar otros módulos de la biblioteca estándar de Python. También puede importar numpy o pandas, si está familiarizado con ellos, pero no debe utilizar ningún otro módulo Python de terceros.

Consejos

Puede que las funciones del módulo de Python random le resulten útiles para tomar decisiones de forma pseudoaleatoria.

Problema 4 - Herencia

Objetivo

Escribe una IA para evaluar la probabilidad de que una persona tenga un rasgo genético particular.

```
$ python heredity.py data/family0.csv
Harry:
    Gene:
        2: 0.0092
        1: 0.4557
        0: 0.5351
    Trait:
        True: 0.2665
        False: 0.7335
James:
    Gene:
        2: 0.1976
        1: 0.5106
        0: 0.2918
    Trait:
        True: 1.0000
        False: 0.0000
Lily:
    Gene:
        2: 0.0036
        1: 0.0136
        0: 0.9827
    Trait:
        True: 0.0000
        False: 1.0000
```

Introducción

Las versiones mutadas del gen GJB2 son una de las principales causas de discapacidad auditiva en los recién nacidos. Cada persona es portadora de dos versiones del gen, por lo que cada persona tiene el potencial de poseer 0, 1 o 2 copias de la versión GJB2 para discapacidad auditiva. Sin embargo, a menos que una persona se someta a pruebas genéticas, no es tan fácil saber cuántas copias de GJB2 mutado tiene una persona. Se trata de un "estado oculto": información que tiene un efecto que podemos observar (discapacidad auditiva), pero que no necesariamente conocemos directamente. Después de todo, algunas personas pueden tener 1 o 2 copias de GJB2 mutado pero no presentar discapacidad auditiva, mientras que otras pueden no tener copias de GJB2 mutado y aun así presentar discapacidad auditiva.

Cada niño hereda una copia del gen GJB2 de cada uno de sus padres. Si un padre tiene dos copias del gen mutado, le transmitirá el gen mutado al niño; si un padre no tiene copias del gen mutado, entonces no transmitirá el gen mutado al niño; y si uno de los padres tiene una copia del gen mutado, entonces el gen se transmite al niño con una probabilidad de 0,5. Sin embargo, una vez que un gen se transmite, tiene cierta probabilidad de sufrir una mutación adicional: pasar de una versión del gen que causa discapacidad auditiva a una versión que no la causa, o viceversa.

Podemos intentar modelar todas estas relaciones formando una red bayesiana de todas las variables relevantes, como en la siguiente, que considera una familia de dos padres y un solo hijo.

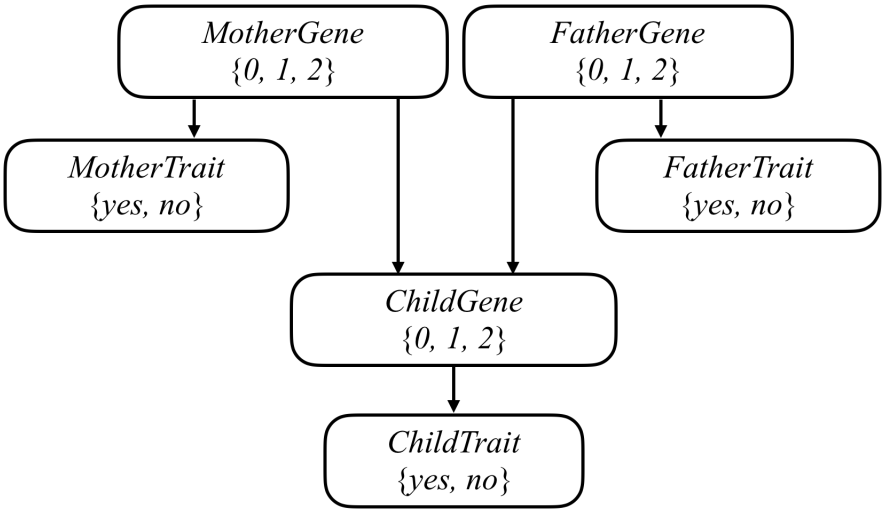

Figura 10

Cada persona de la familia tiene una variable aleatoria **Gene** que representa cuántas copias de un gen particular (por ejemplo, la versión con discapacidad auditiva de GJB2) tiene una persona: un valor que es 0, 1 o 2. Cada persona de la familia también tiene una variable aleatoria **Trait**, que es yes o no dependiendo de si esa persona expresa un rasgo (por ejemplo, discapacidad auditiva) basado en ese gen. Hay una flecha desde la variable **Gene** de cada persona hasta su variable **Trait** para codificar la idea de que los genes de una persona afectan la probabilidad de que tenga un rasgo particular. Mientras tanto, también hay una flecha desde la variable aleatoria Gene de la madre y el padre hasta la variable aleatoria Gene de su hijo: los genes del niño dependen de los genes de sus padres.

Su tarea en este proyecto es utilizar este modelo para hacer inferencias sobre una población. Dada información sobre las personas, quiénes son sus padres y si tienen un rasgo observable particular (por ejemplo, pérdida de audición) causado por un gen determinado, su IA inferirá la distribución de probabilidad de los genes de cada persona, así como la distribución de probabilidad de si alguno persona exhibirá el rasgo en cuestión.

Descargar el código: `problema4-herencia.zip`.

Explicación del problema

Eche un vistazo a uno de los conjuntos de datos de muestra en el directorio data abriendo `data/family0.csv` (puede abrirlo en un editor de texto o en una aplicación de hoja de cálculo como Google Sheets, Excel o Apple Numbers). Observe que la primera fila define las columnas de este archivo CSV: **name**, **mother**, **father** y **trait**. La siguiente fila indica que Harry tiene a Lily como madre, a James como padre y la celda vacía significa que no sabemos si Harry tiene el rasgo **trait** o no. Mientras tanto, James no tiene padres enumerados en nuestro conjunto de datos (como lo indican las celdas vacías para mothery father), y exhibe el rasgo (como lo indica 1en eltraitcelúla). Lily, por otro lado, tampoco tiene padres enumerados en el conjunto de datos, pero no exhibe el rasgo (como lo indica en la 0celda trait).

Abre heredity.py y echa un vistazo primero a la definición de PROBS. PROBS es un diccionario que contiene una serie de constantes que representan probabilidades de varios eventos diferentes. Todos estos eventos tienen que ver con cuántas copias de un gen en particular tiene una persona (en adelante, simplemente .el gen") y si una persona exhibe un rasgo particular (en adelante, .el rasgo") basado en ese gen. Los datos aquí se basan libremente en las probabilidades de la versión con discapacidad auditiva del gen GJB2 y el rasgo de discapacidad auditiva, pero al cambiar estos valores, ¡podrías usar tu IA para hacer inferencias sobre otros genes y rasgos también!

Primero, PROBS["gene"]representa la distribución de probabilidad incondicional sobre el gen (es decir, la probabilidad si no sabemos nada sobre los padres de esa persona). Según los datos del código de distribución, parecería que en la población hay un 1 % de probabilidad de tener 2 copias del gen, un 3 % de probabilidad de tener 1 copia del gen y un 96 % de probabilidad de tener 0. copias del gen.

A continuación, PROBS["trait"]representa la probabilidad condicional de que una persona presente un rasgo (como una discapacidad auditiva). En realidad, se trata de tres distribuciones de probabilidad diferentes: una para cada valor posible de gene. También PROBS["trait"][2]lo es la distribución de probabilidad de que una persona tenga el rasgo dado que tiene dos versiones del gen: en este caso, tiene un 65 % de posibilidades de exhibir el rasgo y un 35 % de posibilidades de no exhibirlo. Mientras tanto, si una persona tiene 0 copias del gen, tiene un 1 % de posibilidades de exhibir el rasgo y un 99 % de posibilidades de no exhibirlo.

Finalmente, PROBS["mutation"]es la probabilidad de que un gen mute de ser el gen en cuestión a no ser ese gen, y viceversa. Si una madre tiene dos versiones del gen, por ejemplo, y por lo tanto le transmite una a su hijo, hay un 1 % de posibilidades de que mute y deje de ser el gen objetivo. Por el contrario, si una madre no tiene versiones del gen y, por lo tanto, no se lo transmite a su hijo, existe un 1 % de posibilidades de que mute y se convierta en el gen objetivo. Por lo tanto, es posible que incluso si ninguno de los padres tiene copias del gen en cuestión, su hijo tenga 1 o incluso 2 copias del gen.

En última instancia, las probabilidades que calcule se basarán en estos valores en PROBS.

Ahora, eche un vistazo a la función main. La función primero carga datos de un archivo en un diccionario people. people asigna el nombre de cada persona a otro diccionario que contiene información sobre ella: incluido su nombre, su madre (si aparece una en el conjunto de datos), su padre (si aparece uno en el conjunto de datos) y si se observa que tienen la rasgo en cuestión (True si lo hacen, False si no lo hacen y None si no lo sabemos).

A continuación, main define un diccionario de probabilities, con todas las probabilidades inicialmente establecidas en 0. Esto es en última instancia lo que calculará su proyecto: para cada persona, su IA calculará la distribución de probabilidad sobre cuántas copias del gen tiene, así como si tienen el rasgo o no. probabilities[”Harry”][”gene”][1], por ejemplo, será la probabilidad de que Harry tenga 1 copia del gen y probabilities[”Lily”][”trait”][False]será la probabilidad de que Lily no presente el rasgo.

Si no está familiarizado, este diccionario probabilities se crea utilizando un diccionario de comprensión de Python, que en este caso crea un par clave/valor para cada uno personen nuestro diccionario de people.

En última instancia, buscamos calcular estas probabilidades basándonos en alguna evidencia: dado que sabemos que ciertas personas exhiben o no el rasgo, nos gustaría determinar estas probabilidades. Recuerde de la conferencia que podemos calcular una probabilidad condicional sumando todas las probabilidades conjuntas que satisfacen la evidencia y luego normalizar esas probabilidades para que cada una sume 1. Su tarea en este proyecto es implementar tres funciones para hacer precisamente eso: joint_probability para calcular una probabilidad conjunta, update para agregar la probabilidad conjunta recién calculada a la distribución de probabilidad existente y luego normalize para garantizar que todas las distribuciones de probabilidad sumen 1 al final.

Especificación

Complete las implementaciones de joint_probability, update y normalize.

La función joint_probability debe tomar como entrada un diccionario de personas, junto con datos sobre quién tiene cuántas copias de cada uno de los genes y quién exhibe el rasgo. La función debería devolver la probabilidad conjunta de que ocurran todos esos eventos.

- La función acepta cuatro valores como entrada: **people**, **one_gene**, **two_genes**, y **have_trait**.
    - **people** es un diccionario de personas como se describe en la sección Çomprensión”. Las claves representan nombres y los valores son diccionarios que contienen claves **mother** y **father**. Puede suponer que **mother** y **father** están en blanco (no hay información de los padres en el conjunto de datos) o que **mother** y **father** ambos se referirán a otras personas en el **people** diccionario.

  - **one_gene** es un conjunto de todas las personas para quienes queremos calcular la probabilidad de que tengan una copia del gen.
  - **two_genes** es un conjunto de todas las personas para quienes queremos calcular la probabilidad de que tengan dos copias del gen.
  - **have_trait** es un conjunto de todas las personas para quienes queremos calcular la probabilidad de que tengan el rasgo.

- Para cualquier persona que no esté en **one_gene** o **two_genes**, nos gustaría calcular la probabilidad de que no tenga copias del gen; y para cualquiera que no esté en **have_trait**, nos gustaría calcular la probabilidad de que no tenga el rasgo.

- Por ejemplo, si la familia está formada por Harry, James y Lily, entonces llamar a esta función donde **one_gene** = {"Harry"}, **two_genes** = {"James"} y **trait** = {"Harry", "James"} debería calcular la probabilidad de que Lily tenga cero copias del gen, Harry tenga una copia del gen, James tenga dos copias del gen, Harry exhibe el rasgo, James exhibe el rasgo y Lily no exhibe el rasgo.

- Para cualquier persona que no tenga padres enumerados en el conjunto de datos, utilice la distribución de probabilidad **PROBS["gene"]** para determinar la probabilidad de que tenga un número particular del gen.

- Para cualquiera que tenga padres en el conjunto de datos, cada padre le transmitirá uno de sus dos genes a su hijo al azar, y existe la **PROBS["mutation"]** posibilidad de que mute (pase de ser el gen a no ser el gen, o viceversa).

- Utilice la distribución de probabilidad **PROBS["trait"]** para calcular la probabilidad de que una persona tenga o no un rasgo en particular.

La función **update** agrega una nueva distribucion de probabilidad conjunta a las distribuciones de probabilidad existentes en **probabilities**.

- La función acepta cinco valores como entrada: **probabilities**, **one_gene**, **two_genes**, **have_trait**, y **p**.
  - **probabilities** es un diccionario de personas como se describe en la sección Çomprensión". Cada persona está asignada a una "gene"distribución y una "trait"distribución.
  - **one_gene** es un conjunto de personas con una copia del gen en la distribución conjunta actual.
  - **two_genes** es un conjunto de personas con dos copias del gen en la distribución conjunta actual.
  - **have_trait** es un conjunto de personas con el rasgo en la distribución conjunta actual.
  - **p** es la probabilidad de la distribución conjunta.

- Para cada persona **person** en **probabilities**, la función debe actualizar la **probabilities**[**person**][”**gene**”] distribución y **probabilities**[**person**][”**trait**”] la distribución agregando **p** el valor apropiado en cada distribución. Todos los demás valores deben dejarse sin cambios.
- Por ejemplo, si ”Harry.[es]tuvieran en ambos **two_genes** y en **have_trait**, entonces **p** se agregarían a **probabilities**[”**Harry**”][”**gene**”][**2**] y a **probabilities**[”**Harry**”][”**trait**”][**True**
- La función no debería devolver ningún valor: sólo necesita actualizar el **probabilities** diccionario.

La función **normalize** actualiza un diccionario de probabilidades de modo que cada distribución de probabilidad esté normalizada (es decir, suma 1, con proporciones relativas iguales).

- La función acepta un único valor: **probabilities**.
  - **probabilities** es un diccionario de personas como se describe en la sección .[Ex]plicacion del problema”. Cada persona está asignada a una ”gene”distribución y una ”trait”distribuc
- Para ambas distribuciones para cada persona en **probabilities**, esta función debe normalizar esa distribución de modo que los valores en la distribución sumen 1 y los valores relativos en la distribución sean los mismos.
- Por ejemplo, si **probabilities**[”**Harry**”][”**trait**”][**True**] fuera igual a 0.1 y **probabilities**[”**Harry**”][”**trait**”][**False**] fuera igual a 0.3, entonces su función debería actualizar el valor anterior para que sea 0.25 y el último valor para que sea 0.75: los números ahora suman 1 y el último valor sigue siendo tres veces mayor que el valor anterior.
- La función no debería devolver ningún valor: sólo necesita actualizar el **probabilities** diccionario.

No debe modificar nada más que heredity.py, las tres funciones que la especificación requiere que implemente, aunque puede escribir funciones adicionales y/o importar otros módulos de la biblioteca estándar de Python. También puede importar numpyo pandas, si está familiarizado con ellos, pero no debe utilizar ningún otro módulo Python de terceros.

Ejemplo de probabilidad conjunta

Para ayudarle a pensar en cómo calcular probabilidades conjuntas, incluimos a continuación un ejemplo.

Considere el siguiente valor para **people**:

```
{
  'Harry': {'name': 'Harry', 'mother': 'Lily', 'father': 'James', 'trait':
    None},
  'James': {'name': 'James', 'mother': None, 'father': None, 'trait': True
   },
  'Lily': {'name': 'Lily', 'mother': None, 'father': None, 'trait': False}
}

```

Aquí mostraremos el cálculo de **joint_probability(people, {"Harry"}, {"James"}, {"James"})**. Según los argumentos, **one_gene** es {"Harry"}, **two_genes** es {"James"}, y **has_trait** es {"James"}. Por lo tanto, esto representa la probabilidad de que: Lily tenga 0 copias del gen y no tenga el rasgo, Harry tenga 1 copia del gen y no tenga el rasgo, y James tenga 2 copias del gen y sí tenga el rasgo.

Comenzamos con Lily (el orden en que consideramos a las personas no importa, siempre y cuando multipliquemos los valores correctos, ya que la multiplicación es conmutativa). Lily tiene 0 copias del gen con probabilidad 0.96 (esto es $PROBS["gene"][0]$). Dado que tiene 0 copias del gen, no tiene el rasgo con probabilidad 0.99 (esto es $PROBS["trait"][0][False]$). Por lo tanto, la probabilidad de que tenga 0 copias del gen y no tenga el rasgo es $0{,}96 \times 0{,}99 = 0{,}9504$.

A continuación, consideremos a Santiago. James tiene 2 copias del gen con probabilidad 0.01 (esto es $PROBS["gene"][2]$). Dado que tiene 2 copias del gen, la probabilidad de que tenga el rasgo es 0.65. Por lo tanto, la probabilidad de que tenga 2 copias del gen y tenga el rasgo es $0{,}01 \times 0{,}65 = 0{,}0065$.

Finalmente, consideramos a Harry. ¿Cuál es la probabilidad de que Harry tenga 1 copia del gen? Hay dos maneras en que esto puede suceder. O recibe el gen de su madre y no de su padre, o recibe el gen de su padre y no de su madre. Su madre Lily tiene 0 copias del gen, por lo que Harry obtendrá el gen de su madre con probabilidad 0.01 (esto es $PROBS["mutation"]$), ya que la única forma de obtener el gen de su madre es si esta muta; por el contrario, Harry no obtendrá el gen de su madre con probabilidad 0.99. Su padre James tiene 2 copias del gen, por lo que Harry obtendrá el gen de su padre con probabilidad 0.99 (esto es $1 - PROBS["mutation"]$), pero obtendrá el gen de su madre con probabilidad 0.01 (la posibilidad de una mutación). Ambos casos se pueden sumar para obtener $0{,}99 \times 0{,}99 + 0{,}01 \times 0{,}01 = 0{,}9802$, la probabilidad de que Harry tenga 1 copia del gen.

Dado que Harry tiene 1 copia del gen, la probabilidad de que no tenga el rasgo es 0.44 (esto es $PROBS["trait"][1][False]$). Entonces, la probabilidad de que Harry tenga 1 copia del gen y no tenga el rasgo es $0{,}9802 \times 0{,}44 = 0{,}431288$.

Por lo tanto, toda la probabilidad conjunta es solo el resultado de multiplicar todos estos valores para cada una de las tres personas: $0{,}9504 \times 0{,}0065 \times 0{,}431288 = 0{,}0026643247488$.

Consejos

Recuerde que para calcular una probabilidad conjunta de múltiples eventos, puede hacerlo multiplicando esas probabilidades. Pero recuerde que para cualquier niño, la probabilidad de que tenga una cierta cantidad de genes está condicionada a los genes que tienen sus padres.