Instituto Politécnico Nacional
Escuela Superior de Cómputo
Ingeniería en Inteligencia Artificial
Grupo: 5BM1
Unidad de aprendizaje: Aprendizaje de Máquina
Práctica #1, 20Q
Profesor: Abdiel Reyes Vera
Alumnos:
Arenas Trejo Diego Israel
Garduño Villeda Joshua Hashem
Velázquez Arrieta Eduardo Uriel

índice de contenido

Índice de imágenes

Introducción

Esta práctica tiene como objetivo recrear el juego 20Q (Twenty Questions), originalmente desarrollado por Robin Burgener en 1988. La mecánica del juego es simple: el programa intenta adivinar un concepto (un objeto, animal, lugar, etc.) en el que piensa el usuario haciéndole un máximo de veinte preguntas de sí o no.

A diferencia de los sistemas expertos tradicionales (más rígidos y complejos), 20Q se caracteriza por su adaptabilidad y escalabilidad, lo que permite su implementación en diversos dispositivos con fines más avanzados.

Su funcionamiento se basa en una base de datos de objetos. Tras cada respuesta, el algoritmo descarta todos los conceptos que no se ajustan a ella, reduciendo así la lista de posibilidades. Este proceso se repite iterativamente hasta que se alcanza el límite de preguntas o se identifica un único elemento. Si el programa acierta antes de las veinte preguntas, el juego termina. En caso de no acertar tras las veinte, realiza cinco preguntas extra. Si aún así falla, aprende de la partida guardando las respuestas del usuario para refinar sus futuras predicciones.

El juego no solo representa un ejemplo de razonamiento lógico, ya que este puede analizarse desde la perspectiva del aprendizaje de máquina. Cuando un usuario responde una pregunta, el sistema obtiene un dato que reduce el espacio de búsqueda, lo cual actúa como un proceso de clasificación supervisada.

Este mecanismo de descarte se alinea directamente con el aprendizaje de máquina, específicamente con los árboles de decisión. Cada pregunta actúa como un nodo de decisión que divide el conjunto de posibles conceptos. Las respuestas de "sí" o "no" guían al sistema por las ramas del árbol, guiándolo hacia una predicción final. En esencia, cada respuesta del usuario es un dato que ayuda a clasificar y reducir el conjunto de posibilidades, de forma similar a un proceso de clasificación supervisada.

Los árboles de decisión son herramientas muy intuitivas para la clasificación y la regresión. Su objetivo es aprender reglas sencillas que se pueden usar para predecir un resultado. La clave está en que el árbol se construye dividiendo repetidamente el conjunto de datos en subconjuntos más pequeños y homogéneos, basándose en la característica que mejor separa los datos.

Para esta práctica estaremos trabajando con el Lenguaje de Programación Python, el cual es de alto nivel, de propósito general y ampliamente utilizado en la actualidad. Se sabe que la sintaxis es clara y legible.

Se utiliza en una gran variedad de campos, como la ciencia de datos, el desarrollo web, la automatización de tareas, la Inteligencia Artificial y el Aprendizaje Automático.

Desarrollo

Con base en el ejemplo práctico realizado en clase decidimos implementar un árbol de decisiones, ya que por la misma naturaleza del juego(solo responder si o no), nos ayuda para poder ir descartando los objetos/personas/lugares/animales que no cumplan con las características.

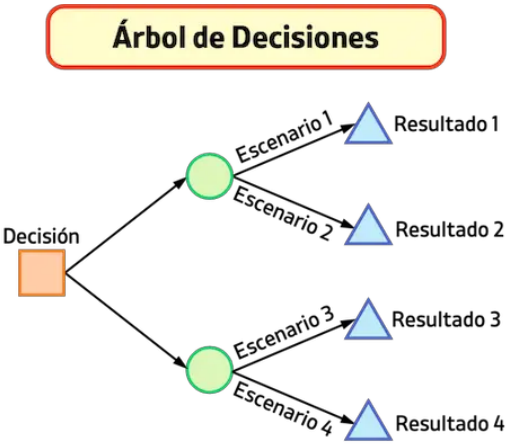

Imagen 1: Árbol de decisiones

A su vez esto también representa un problema, ya que si las preguntas son ambiguas y poco claras puede que el jugador malinterprete y pueda llegar a descartar la opción en la que esté pensando.

Manejo de archivos

Una vez con una idea básica de cómo desarrollar el algoritmo principal decidimos crear 2 archivos para la práctica, diccionario.py y main.py.

diccionario.py

Código completo:

Sirve para almacenar los posibles “pensamientos del usuario” en un diccionario donde cada “pensamiento” se relaciona con ciertas características, por ejemplo:
“balon" : {"objeto": True, "entretenimiento": True, "pequenio": True, “deporte”:True}.
Con esto podemos deducir que es un objeto, sirve para hacer algún tipo de entrenamiento,es pequenio (el equipo está acostumbrado a trabajar con teclado en inglés, por lo que omitimos la ñ y acentos al menos en el código) y sirve para hacer deporte. También tenemos otro diccionario con las preguntas que también son un diccionario que funciona con un par clave valor de la siguiente manera:
“característica”: “¿La cosa en la que estás pensando cumple esta característica?”

Y algunos ejemplos son.
"objeto" :"¿Es un objeto fisico? (s/n): ",
"persona" : "¿Es una persona real o ficticia? (s/n): ",
"lugar" : "¿Es un lugar? (s/n): ",

Por último tenemos un arreglo con todas las características, destacando algunas de las más relevantes tenemos:
caracteristicas = ["objeto", "persona", "lugar", "vivo", "animal", "tecnologico","famoso", "natural", "comestible", …, "pintor"]

main.py:

Código completo:

```
import os
from diccionario import pensamiento
from diccionario import preguntas
from diccionario import caracteristicas
from collections import Counter


class Juego_20Q:
    def __init__(self):

        self.objetos_posibles = list(pensamiento.keys())
        self.caracteristicas_usadas = set()
        self.respuestas_jugador = {}

    def pregunta_optima(self):

        if not self.objetos_posibles:
            return None

        contador = Counter()

        for objeto in self.objetos_posibles:
            caracteristicas_objeto = pensamiento[objeto]

            for caracteristica, valor in caracteristicas_objeto.items():
                if valor and caracteristica not in self.caracteristicas_usadas and caracteristica in
preguntas:
                    contador[caracteristica] += 1

        if not contador:
            return None

        mejor_caracteristica = None
```

```
        mejor_diferencia = float('inf')
        total_objetos = len(self.objetos_posibles)

        for caracteristica, count in contador.items():
            diferencia = abs(count - (total_objetos / 2))
            if diferencia < mejor_diferencia:
                mejor_diferencia = diferencia
                mejor_caracteristica = caracteristica

        return mejor_caracteristica


    def hacer_preguntas(self, caracteristica):
        if caracteristica not in preguntas:
            return None
        pregunta = preguntas[caracteristica]
        while True:
            respuesta = input(pregunta.strip() + " ").lower()
            if respuesta in ['s', 'si', 'y', 'yes']:
                self.respuestas_jugador[caracteristica] = True
                self.caracteristicas_usadas.add(caracteristica)
                return True
            elif respuesta in ['no', 'n']:
                self.respuestas_jugador[caracteristica] = False
                self.caracteristicas_usadas.add(caracteristica)
                return False
            else:
                print('Por favor ingresa una opcion valida(s/n)')


#Filtra los objetos con las caracteristicas que el usuario afirme que tiene
    def filtrar_objetos(self):

        nuevos_objetos = []

        for objeto in self.objetos_posibles:
            coincide = True
            for caracteristica, valor_esperado in self.respuestas_jugador.items():
                if caracteristica in pensamiento[objeto]:
                    valor_objeto = pensamiento[objeto][caracteristica]
                    if valor_objeto != valor_esperado:
                        coincide = False
                        break

                elif valor_esperado:
                    coincide = False
                    break
```

```
        if coincide:
            nuevos_objetos.append(objeto)

    self.objetos_posibles = nuevos_objetos


def adivinar(self):
    if len(self.objetos_posibles) == 1:
        objeto = self.objetos_posibles[0]
        respuesta = input(f"¿Estas pensando en {objeto}? s/n: ").strip().lower()
        if respuesta in ['s', 'si', 'y', 'yes']:
            print("!Adivine!")
            return True
        else:
            print("No logre adivinar :(")
            return False
    elif len(self.objetos_posibles) == 0:
            print("Mi base de conocimientos no contiene lo que buscas, !ganaste!.")
            return True
    return False


def jugar(self):
    print("Bienvenido al juego 20Q")
    print("Tratare de adivinar lo que tienes en mente.")
    print("Responde con 's' pra si,  o con 'n' para no. \n ")


    max_preguntas = 20
    preguntas_realizadas = 0

    while preguntas_realizadas < max_preguntas and len(self.objetos_posibles) > 1:
        caracteristica = self.pregunta_optima()

        if not caracteristica:
            break

        respuesta = self.hacer_preguntas(caracteristica)
        preguntas_realizadas += 1
        self.filtrar_objetos()

        print(f"Posibilidads restantes: {len(self.objetos_posibles)}")
        print(f"Numero de preguntas realizadas: {preguntas_realizadas}\n")



        if len(self.objetos_posibles) <=3:
            if self.adivinar():
```

```
                return
            else:
                break

        car = False
        i = 0
        if len(self.objetos_posibles) > 1:
            print("\nSe me acabaron las preguntas. ¿Acaso estabas pensando en algo de
estos?")
            while  i < len(self.objetos_posibles) and not car:
                objeto = self.objetos_posibles[i]
                while True:
                    op = input(f"¿Acaso estabas pensando en {objeto} (s/n)?").lower().strip()

                    if op in ['s', 'si', 'yes', 'y']:
                        print("Adivine, GG ez win.")
                        car = True
                        break
                    elif op in ['n','no']:
                        break
                    else:
                        print("Opcion no valida, por favor ingresa una opcion valida (s/n).")
                i += 1
        if not car:
            print("Haz acabado con mi base de conocimientos, ganaste.GG")


if __name__ == "__main__":
    juego = Juego_20Q()
    juego.jugar()
```

Para el archivo main implementamos la parte lógica del programa, que a su vez se divide en varios métodos que explicaremos a continuación.

Primero tenemos una clase llamada Juego_20Q

Declaración de clase y constructor

Primero tenemos la declaración de clase con su método constructor que inicializa el atributo de objetos_posibles con una lista de todas las keys del diccionario pensamiento, el atributo de caracteristicas_usadas inicializado con un set que guardara las características ya preguntadas, al final tenemos el atributo respuestas_jugador que es un diccionario que guarda las respuestas del jugador.

Método de pregunta óptima:

El método recibe el parámetro self, que sirve para acceder a las instancias de la clase y los atributos, después tenemos que si no hay posibles objetos para contar sus características no retorna nada, después inicializamos un diccionario con el método counter que nos sirve para contar las veces que se repite las características de cada uno de los posibles “pensamientos”.
Una vez contamos el número de veces que se repiten todas las características, buscamos cuál es la más grande, pero no solo se toma en cuenta el número de veces que se repite si no también la que mejor divida el conjunto total de “pensamientos”.

Método de hacer pregunta:

Recibe 2 parámetros, self y característica, como ya vimos en el código de diccionario las preguntas también están en un diccionario par clave valor, que en este caso se toma la clave “característica” con su valor asociado en este caso la pregunta para poder hacérsela al jugador, también se incluye las posibilidades de que el usuario ingrese algo que no es válido.

Método para filtrar objetos:

Solo recibe self, la función de este método es “podar” las ramas del árbol que no se vayan descartando, esto con base en las respuestas del jugador.
Este método es fundamental, ya que sirve para descartar todos los “pensamientos” que no cumplan con las características del usuario.

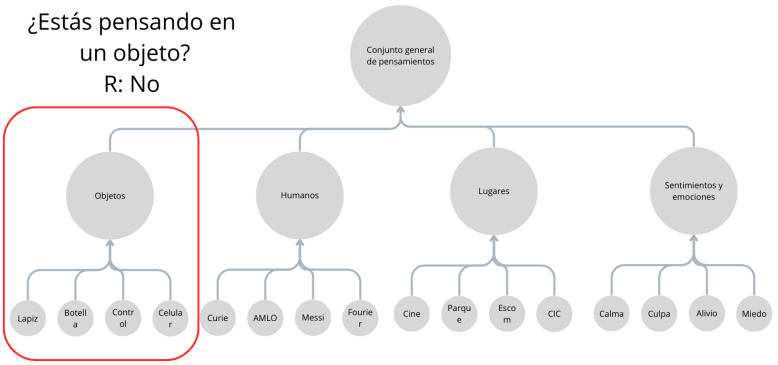

Imagen 2: Funcionamiento del filtrado de objetos

Método adivinar

Después de varias preguntas el programa va a llegar a los nodos base u hojas, este método toma simplemente pregunta si a la hoja a la que se ha llegado es en lo que estaba pensando el jugador.

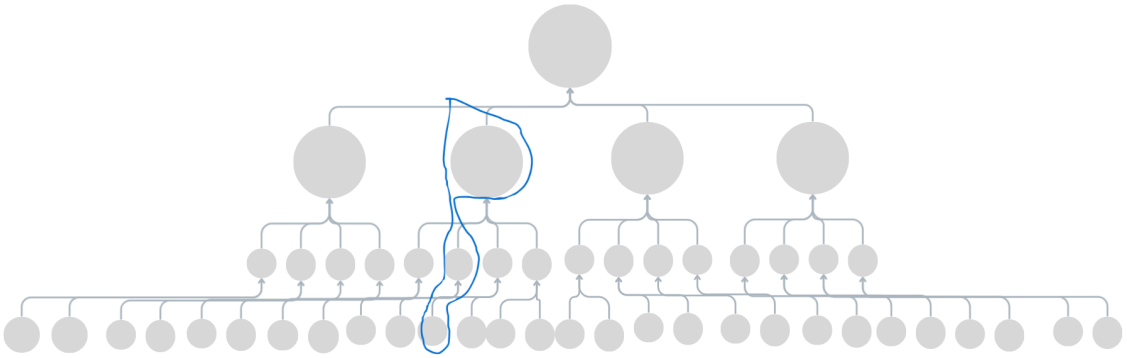

Imagen 3: Nodo raíz

Método jugar

Primero le da la bienvenida al jugador con instrucciones de cómo debe responder, luego se declaran dos variables, una para conocer el máximo de preguntas que se puede realizar y otra para saber cuántas preguntas se han realizado.
Después se empiezan a hacer las preguntas y con un while se cuentan la cantidad de preguntas y se toma en cuenta que la cantidad de “pensamientos” disponibles sea mayor a cero.
En caso de que queden varios objetos con características muy parecidas o iguales se ejecuta un último ciclo que recorre todas esas posibilidades restantes.

Conclusiones

Tomando como punto de referencia la actividad que realizamos en clase, en la cual todos los compañeros del aula debíamos adivinar en qué estaba pensando el profesor en solo 20 preguntas, fue un gran parteaguas en cuanto a dificultad de esta práctica. Ahora tomando como referencia el juego 20Q (Twenty Questions), el cual estaba basado en un árbol de decisión, y cada respuesta del usuario elimina las ramas menos irrelevantes, se podría tener una mejor idea del funcionamiento de este algoritmo. Sin embargo, se debe ser muy específico con respecto a lo que serían las características de cada objeto. Ya que existe mucha ambigüedad sobre los conocimientos de cada persona, por ejemplo, lo que una persona considere como algo hecho a mano o algo prefabricado, Si algo es de la naturaleza o si fue creado o transformado por el ser humano.
De igual manera, se investigó el funcionamiento de este juego. A diferencia de las indicaciones del profesor sobre solo poder responder con “si” y “no” a las preguntas que realiza el código, en 20Q podrás responder con “a veces” o “desconocido”, esto le daba un nivel de certeza menor a la inteligencia artificial del juego, por lo cual éste podía fallar a veces y no adivinar lo que estaba pensando el usuario. En este caso ocurrió un problema similar en el testeo de nuestro código, habiendo casos donde al primer intento no adivinaba lo que estaba pensando el usuario. Pero si se volvía a intentar con el mismo objeto una segunda vez, si lograba adivinarlo.
Al solo poder responder con sí y no, la ambigüedad de las respuestas se eleva mucho, y en este caso, nuestro programa si sigue un modelo como el de 20Q para eliminar preguntas, pero cuando se responde que no, se arroja una pregunta aleatoria sobre otro tema para explorar esta rama, y este es un problema que ocurre frecuentemente. Como mencionaba al ser de manera aleatoria no hay manera de controlar como el programa manejara el cambio de tema para poder adivinar las preguntas.

De igual manera el límite de preguntas fue otra limitante dentro del desarrollo de la práctica. Ya que de igual manera existe un juego conocido como “Akinator” el cual tiene la misma temática que 20Q, con la distinción de que Akinator se centra en adivinar personajes únicamente. Al ser un campo mucho más reducido es mucho más fácil para el programa sesgar las respuestas desde un inicio, preguntando tan solo si el personaje es ficticio o no. Aun así este programa no tiene límite de preguntas, por lo que si el personaje que el usuario eligió no es muy conocido, o simplemente no está registrado dentro de la base de datos del sistema, este puede llegar incluso a sobrepasar las 100 preguntas al usuario y aun así nunca adivinar lo que el usuario estaba pensando.

Definitivamente la complejidad de un programa así es enorme, el área de error es incluso todavía mayor habiendo tantas limitantes como el número de respuestas, la ambigüedad de las mismas, e incluso la opción de respuestas que el usuario puede dar. Sin embargo, es posible poder programar un software que realice esta tarea, pero este debe tener una base de datos extensa, incluso enorme, ya que la cantidad de información que almacena el cerebro humano es inmensa. Y adivinar qué pensó el usuario en solo 20 preguntas es una actividad muy compleja que solo se puede lograr catalogando toda la información y segmentando de manera específica cada una de las respuestas que da el usuario.

Referencias

(S/f). Espacenet.com. Recuperado el 29 de agosto de 2025, de https://worldwide.espacenet.com/patent/search/family/035636631/publication/EP1710735A1?q=pn%3DEP1710735

20Q - wikiwand. (s/f). Wikiwand.com. Recuperado el 29 de agosto de 2025, de https://www.wikiwand.com/es/articles/20Q

¿Qué es un árbol de decisión? (2025, enero 30). Ibm.com. https://www.ibm.com/es-es/think/topics/decision-trees