Inteligencia Artificial: Práctica 1

Ismael Sallami Moreno

ism350zsallami@correo.ugr.es

Asignatura: Inteligencia Artificial
Tema: Notas de Prácticas
Fecha: 17 de marzo de 2025

Universidad de Granada

Como enlace de interés adjunto el enlace de la conversación con ChatGPT: `https://chatgpt.com/share/67d47275-91dc-8012-9740-57ce1ed7abcb`

Además voy a añadir ambos códigos generados en la práctica 1 con Python con la IA, así como el archivo .json que he usado para entrenar el modelo.

1. Archivo .json

```
{
    "type": "text",
    "data": {
      "positivo": [
        "Me encanta este lugar",
        "Hoy ha sido un gran día",
        "Qué alegría verte",
        "Este producto es increíble",
        "Me siento muy feliz",
        "Gracias por tu ayuda",
        "Todo salió mejor de lo esperado",
        "Es un placer trabajar contigo",
        "Estoy muy emocionado por esto",
        "La comida estaba deliciosa",
        "Eres una persona maravillosa",
        "Estoy muy agradecido",
        "Fue una experiencia increíble",
        "Lo pasé genial ayer",
        "Este es el mejor regalo que he recibido"
      ],
      "negativo": [
        "Odio cuando pasa esto",
        "Hoy ha sido un día terrible",
        "No soporto esta situación",
        "Este producto es un desastre",
        "Me siento muy triste",
        "No puedo creer que haya sucedido esto",
        "Todo salió peor de lo esperado",
        "Trabajar aquí es un infierno",
        "Estoy muy decepcionado",
        "La comida estaba horrible",
        "Eres una persona insoportable",
        "Estoy muy frustrado",
        "Fue una experiencia espantosa",
        "Lo pasé fatal ayer",
        "Este es el peor regalo que he recibido"
      ]
    }
  }
```

Listing 1: Archivo .json para entrenar el modelo

2. Código 1

```
import sys
import heapq

```

```
class Puzzle8:
    def __init__(self, start_state):
        self.start_state = start_state
        self.goal_state = "123456780"
        self.moves = {
            0: [1, 3], 1: [0, 2, 4], 2: [1, 5],
            3: [0, 4, 6], 4: [1, 3, 5, 7], 5: [2, 4, 8],
            6: [3, 7], 7: [4, 6, 8], 8: [5, 7]
        }

    def heuristic(self, state):
        distance = 0
        for i, val in enumerate(state):
            if val != '0':
                goal_index = self.goal_state.index(val)
                distance += abs(i // 3 - goal_index // 3) + abs(i %
                    3 - goal_index % 3)
        return distance

    def get_neighbors(self, state):
        zero_index = state.index("0")
        neighbors = []
        for move in self.moves[zero_index]:
            new_state = list(state)
            new_state[zero_index], new_state[move] = new_state[move
                ], new_state[zero_index]
            neighbors.append("".join(new_state))
        return neighbors

    def solve(self):
        priority_queue = [(self.heuristic(self.start_state), 0,
            self.start_state, [])]
        visited = set()

        while priority_queue:
            _, cost, state, path = heapq.heappop(priority_queue)

            if state == self.goal_state:
                return path + [state]

            if state in visited:
                continue
            visited.add(state)

            for neighbor in self.get_neighbors(state):
                heapq.heappush(priority_queue, (cost + self.
                    heuristic(neighbor), cost + 1, neighbor, path +
                    [state]))

        return None

    def print_solution(self, solution):
        if not solution:
            print("No hay solución para este estado inicial.")
            return

        print("Solución encontrada:")
        for state in solution:
```

```
            for i in range(0, 9, 3):
                print(state[i:i+3])
            print("-")

if __name__ == "__main__":
    if len(sys.argv) != 2 or len(sys.argv[1]) != 9 or not sys.argv[1].isdigit():
        print("Uso: python solve_8puzzle.py <estado_inicial>")
        sys.exit(1)

    start_state = sys.argv[1]
    puzzle = Puzzle8(start_state)
    solution = puzzle.solve()
    puzzle.print_solution(solution)
```

Listing 2: Código generado en la práctica 1

3. Código 2

```
import sys
import heapq

class Puzzle8:
    def __init__(self, start_state):
        self.start_state = start_state
        self.goal_state = "123456780"
        self.moves = {
            0: [1, 3], 1: [0, 2, 4], 2: [1, 5],
            3: [0, 4, 6], 4: [1, 3, 5, 7], 5: [2, 4, 8],
            6: [3, 7], 7: [4, 6, 8], 8: [5, 7]
        }

    def heuristic(self, state):
        distance = 0
        for i, val in enumerate(state):
            if val != '0':
                goal_index = self.goal_state.index(val)
                distance += abs(i // 3 - goal_index // 3) + abs(i % 3 - goal_index % 3)
        return distance

    def get_neighbors(self, state):
        zero_index = state.index("0")
        neighbors = []
        for move in self.moves[zero_index]:
            new_state = list(state)
            new_state[zero_index], new_state[move] = new_state[move], new_state[zero_index]
            neighbors.append("".join(new_state))
        return neighbors

    def count_inversions(self, state):
        numbers = [int(c) for c in state if c != '0']
        inversions = 0
        for i in range(len(numbers)):
            for j in range(i + 1, len(numbers)):
```

```
                if numbers[i] > numbers[j]:
                    inversions += 1
        return inversions

    def is_solvable(self):
        return self.count_inversions(self.start_state) % 2 == 0

    def solve(self):
        if not self.is_solvable():
            return None

        priority_queue = [(self.heuristic(self.start_state), 0,
            self.start_state, [])]
        visited = set()

        while priority_queue:
            _, cost, state, path = heapq.heappop(priority_queue)

            if state == self.goal_state:
                return path + [state]

            if state in visited:
                continue
            visited.add(state)

            for neighbor in self.get_neighbors(state):
                heapq.heappush(priority_queue, (cost + self.
                    heuristic(neighbor), cost + 1, neighbor, path +
                    [state]))

        return None

    def print_solution(self, solution):
        if not solution:
            print("No hay solución para este estado inicial.")
            return

        print("Solución encontrada:")
        for state in solution:
            for i in range(0, 9, 3):
                print(state[i:i+3])
            print("-")

if __name__ == "__main__":
    if len(sys.argv) != 2 or len(sys.argv[1]) != 9 or not sys.argv
        [1].isdigit():
        print("Uso: python solve_8puzzle.py <estado_inicial>")
        sys.exit(1)

    start_state = sys.argv[1]
    puzzle = Puzzle8(start_state)

    if not puzzle.is_solvable():
        print("El estado inicial no es resoluble.")
        sys.exit(1)

    solution = puzzle.solve()
    puzzle.print_solution(solution)
```

Listing 3: Código generado en la práctica 1

Ambos códigos funcionan correctamente, ya que han sido comprobados en base a como se exponía en el guión. *Para cualquier aclaración o duda, no dudéis en contactar conmigo.*

Referencias

[1] Ismael Sallami Moreno, **Estudiante del Doble Grado en Ingeniería Informática + ADE**, Universidad de Granada, 2025.