Laboratorio 2 Introducción a la Inteligencia Artificial

David Alejandro Avendaño Rodríguez, Manuel Santiago Ibañez Gutierrez

31 de agosto de 2025

1. Introducción

En la actualidad, el análisis de datos y su representación gráfica se han convertido en pilares fundamentales para la investigación científica, el desarrollo tecnológico y la toma de decisiones estratégicas. El volumen y la complejidad de la información que se genera día a día requieren no solo de herramientas que permitan su almacenamiento y procesamiento, sino también de entornos que faciliten su interpretación a través de visualizaciones claras, interactivas y escalables.

En este contexto, el ecosistema de librerías en Python orientado al análisis y visualización de datos ofrece un flujo de trabajo robusto y flexible. Dicho ecosistema está compuesto por herramientas especializadas en la manipulación de grandes volúmenes de información —como `pandas`, `dask` o `xarray`— que se complementan con librerías diseñadas para la representación gráfica —tales como `hvPlot`, `HoloViews`, `Datashader`, `Bokeh` y `Matplotlib`. Esta integración no solo agiliza el proceso de exploración y entendimiento de datos, sino que también habilita el desarrollo de soluciones interactivas mediante plataformas como `Streamlit`, orientadas a la comunicación efectiva de resultados.

El presente documento tiene como propósito describir y analizar este flujo de trabajo, detallando la función de cada una de las librerías involucradas y resaltando la importancia de su interoperabilidad. De esta manera, se busca ofrecer una visión global sobre cómo la combinación de estas herramientas constituye un marco eficiente para transformar datos en conocimiento útil y aplicable.

Objetivos

Objetivo general

Diseñar y documentar un flujo de trabajo en Python que integre librerías de análisis, visualización e interacción de datos, con el fin de demostrar cómo estas herramientas permiten transformar información cruda en resultados gráficos claros y útiles para la toma de decisiones.

Objetivos específicos

1. Implementar la lectura, manipulación y preprocesamiento de datos mediante librerías de análisis como `pandas`, garantizando la correcta estructuración de la información.
2. Desarrollar visualizaciones estáticas e interactivas utilizando librerías como `hvPlot`, `HoloViews` y `Bokeh`, con el fin de facilitar la interpretación de patrones y tendencias.
3. Integrar las visualizaciones obtenidas en un entorno interactivo a través de `Streamlit`, permitiendo la exploración dinámica de los resultados por parte de los usuarios.

2. Primer Punto: Resumen de librerías de visualización y análisis de datos

Para tener una contextualización, se tiene en cuenta un ecosistema de librerías de Python orientadas al análisis y visualización de datos, organizadas en un flujo que integra fuentes de datos, representación intermedia y salidas gráficas interactivas. A continuación, siendo las principales herramientas:

- **pandas**: Biblioteca fundamental para la manipulación y análisis de datos estructurados en Python, basada en estructuras como `DataFrame`.
- **dask**: Extiende las capacidades de `pandas` y `NumPy` para el procesamiento distribuido y paralelo de grandes volúmenes de datos.
- **GeoPandas**: Especializada en el manejo y análisis de datos espaciales, integrando geometrías y operaciones geoespaciales sobre `DataFrames`.
- **RAPIDS**: Conjunto de bibliotecas aceleradas por GPU, diseñadas para análisis de datos y aprendizaje automático en gran escala.
- **xarray**: Orientada al manejo de arreglos multidimensionales (n-dimensiones), con soporte para datos etiquetados, común en ciencias del clima y datos geoespaciales.
- **ibis**: Proporciona una capa de abstracción para consultas analíticas sobre distintos motores de bases de datos, facilitando la portabilidad de código.
- **NetworkX**: Herramienta especializada en el análisis y modelado de grafos y redes complejas.
- **DuckDB**: Motor de base de datos embebido, optimizado para consultas analíticas directamente sobre archivos de datos.
- **polars**: Alternativa moderna a `pandas`, optimizada en rendimiento mediante paralelización y uso intensivo de memoria eficiente.

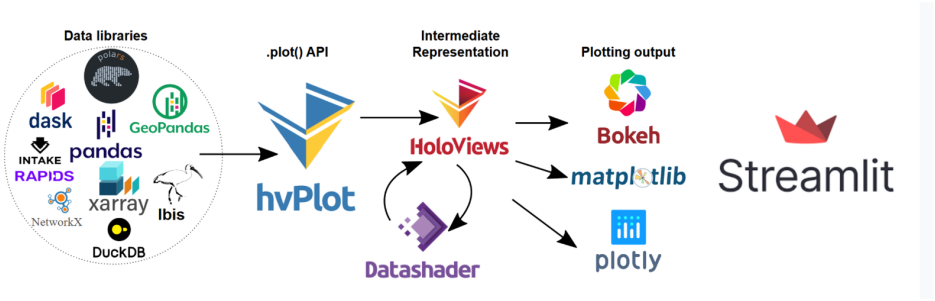

Figura 1: Interpretación ramificada de las librerías participantes.

Todas estas bibliotecas pueden integrarse mediante la API unificada **hvPlot**, la cual convierte los datos en representaciones intermedias a través de **HoloViews** y **Datashader**. Estas representaciones permiten generar salidas gráficas en librerías de visualización como **Bokeh**, **Matplotlib**, **Plotly** e incluso desplegarse en **Streamlit** para aplicaciones interactivas.

3. Segundo punto: Resumen de tesis acerca de agentes inteligentes

Un agente inteligente es una entidad, ya sea software o hardware, que tiene la capacidad de percibir su entorno mediante sensores y actuar sobre él a través de actuadores. Su principal objetivo es alcanzar metas específicas de manera autónoma. Para ello, debe tomar decisiones racionales, adaptarse a cambios en su entorno, interactuar con otros agentes o con usuarios, y ejecutar acciones que maximicen sus posibilidades de éxito.

El campo de potencial artificial es una técnica empleada en la planificación de trayectorias para robots móviles y agentes autónomos. Se basa en la idea de representar el espacio de movimiento como un campo de fuerzas. El objetivo ejerce una fuerza de atracción, que guía al agente hacia la meta. Los obstáculos generan una fuerza de repulsión, que evita colisiones. El movimiento del agente surge de la combinación de estas fuerzas, logrando que avance hacia su objetivo mientras esquiva los obstáculos. Este método es eficiente y sencillo de implementar, aunque puede presentar limitaciones como la aparición de mínimos locales que impiden llegar a la meta.

El modelo BDI es un enfoque de razonamiento para agentes inteligentes inspirado en la forma en que los humanos toman decisiones, ya sea la información que el agente posee sobre su entorno y su estado interno. Deseos, los objetivos o estados que el agente pretende alcanzar. Las intenciones, como los planes y acciones que el agente se compromete a ejecutar para cumplir sus deseos, considerando sus creencias.

Este enfoque permite que el agente no solo reaccione a estímulos externos, sino que también planifique, seleccione y ejecute acciones de manera deliberada. Por ello, se utiliza ampliamente en planificación de trayectorias, sistemas multiagente y robótica autónoma.

4. Tercer punto: Solución al algoritmo BDI

El script implementa un agente inteligente basado en el modelo BDI (Beliefs–Desires–Intentions) para la planificación de trayectorias en un entorno con obstáculos. El objetivo es que el agente navegue desde una posición inicial hasta un punto meta, evitando colisiones y resolviendo situaciones de atasco mediante mecanismos adaptativos. El entorno se modela utilizando un campo de potencial artificial, compuesto por dos componentes, una es el potencial atractivo, este atrae al agente hacia el objetivo, aumentando en función de la distancia al mismo y el potencial repulsivo, que repele al agente de los obstáculos, disminuyendo su influencia conforme aumenta la distancia. La combinación de estas fuerzas define la dirección de movimiento en condiciones normales.

El ciclo de razonamiento del agente se ejecuta, donde en cada iteración:Actualiza sus creencias, genera un deseo según su estado, planifica las intenciones a partir de ese deseo, propone un nuevo movimiento, verificando colisiones con obstáculos. En caso de colisión, aplica un pequeño desvío aleatorio para evitar bloqueos.

La simulación utiliza matplotlib para representar la trayectoria del agente (línea azul discontinua), su posición actual (círculo rojo), el objetivo (cruz verde). los obstáculos (cuadrados negros). los puntos de atasco detectados (estrellas naranjas) y una gráfica en paralelo que muestra la evolución de la energía potencial, lo cual permite evaluar si el agente avanza efectivamente hacia un mínimo global (la meta).

Gracias a esta combinación de campo de potencial artificial y modelo BDI, el agente logra desplazarse de forma autónoma hacia el objetivo, es capaz de detectar y registrar situaciones de atasco, implementa estrategias de escape y evitación basadas en su memoria de entornos problemáticos, mantiene un ciclo deliberativo que le permite alternar entre navegación, escape y evasión según el contexto.

En conjunto, el script demuestra un sistema de planificación de trayectorias inspirado en razonamiento humano, donde el agente no solo reacciona al entorno, sino que también planifica y adapta sus acciones para alcanzar de forma robusta el objetivo.

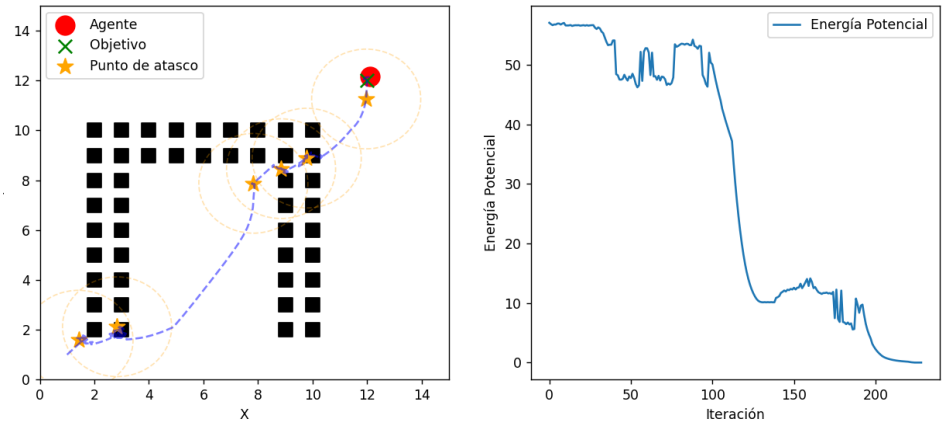

Figura 2: Interpretación gráfica del algoritmo BDI.

El gráfico de la izquierda representa el recorrido seguido por el agente dentro del entorno con obstáculos en forma de herradura. El agente (círculo rojo) partió desde la esquina inferior izquierda y avanzó hacia el objetivo (cruz verde), ubicado en la parte superior derecha. Durante el trayecto, se identificaron múltiples puntos de atasco (estrellas naranjas), los cuales corresponden a zonas donde

el agente experimentó dificultades para avanzar debido a la configuración de los campos de potencial artificial. En dichos puntos, el modelo BDI aplicó mecanismos de escape que le permitieron superar los bloqueos locales y retomar la trayectoria hacia la meta.

El gráfico de la derecha muestra la evolución de la energía potencial a lo largo de las iteraciones. Se observa un comportamiento inicial con oscilaciones relativamente altas, reflejando la presencia de atascos y movimientos de corrección en la trayectoria. Posteriormente, la energía desciende de forma significativa, evidenciando que el agente logra escapar de las trampas locales y aproximarse al objetivo de manera más estable. Finalmente, el valor de la energía converge hacia cero, indicando que el agente alcanza con éxito la posición final deseada.

En conjunto, ambos gráficos confirman que el algoritmo BDI, en combinación con campos de potencial artificial, es capaz de guiar al agente hasta el objetivo, aun cuando este enfrenta zonas de estancamiento que, de no contar con dicho mecanismo, podrían impedir la convergencia de la trayectoria.

Conclusión

El ecosistema representado en la figura muestra cómo la interacción entre librerías de análisis de datos y herramientas de visualización ofrece un flujo de trabajo integral y altamente eficiente. En la etapa inicial, bibliotecas como `pandas`, `dask`, `xarray`, `GeoPandas`, entre otras, proporcionan estructuras y operaciones avanzadas para la manipulación de grandes volúmenes de información, incluyendo datos distribuidos, geoespaciales y en distintos formatos. Posteriormente, mediante la API unificada de `hvPlot`, es posible generar representaciones intermedias que permiten abstraer la complejidad técnica, garantizando una transición fluida hacia `HoloViews` y `Datashader`. Estas últimas no solo potencian la escalabilidad de los gráficos, sino que también aseguran una adecuada representación de conjuntos masivos de datos.

La versatilidad del ecosistema se hace evidente en la etapa final, donde se integran librerías de visualización consolidadas como `Bokeh`, `Matplotlib` y `Plotly`, cada una con fortalezas particulares en gráficos interactivos, personalización y compatibilidad multiplataforma. Finalmente, herramientas de despliegue como `Streamlit` permiten transformar estos análisis en aplicaciones accesibles e interactivas, cerrando el ciclo desde la manipulación de datos hasta su presentación en entornos colaborativos y de fácil acceso.

En conjunto, este flujo refleja la importancia de la interoperabilidad entre librerías, mostrando cómo la integración de diferentes niveles —desde la adquisición y procesamiento de datos hasta su visualización y distribución— contribuye al desarrollo de soluciones más ágiles, escalables y orientadas a la toma de decisiones basada en datos.

5. Script

Listing 1: Algoritmo BDI

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

K_atractivo = 0.5
K_repulsivo = 3.0
radio_repulsion = 3.0

# Funci n para calcular el campo de potencial
def calcular_potencial(posicion_agente, objetivo, obstaculos, epsilon=0.25):
    # Potencial de atracci n hacia el objetivo
    potencial_atractivo = 0.5 * K_atractivo * np.linalg.norm(objetivo -
        posicion_agente)**2

    # Potencial de repulsi n de los obst culos
    potencial_repulsivo = 0
    for obstaculo in obstaculos:
```

```
        distancia = np.linalg.norm(posicion_agente - obstaculo)
        if 1 <= distancia < radio_repulsion:
            potencial_repulsivo += 0.5 * K_repulsivo * (1 / distancia - 1 / radio_repulsion)**2
        elif epsilon < distancia < 1:
            potencial_repulsivo += 0.5 * K_repulsivo * (1 / distancia - 1 / radio_repulsion)
        else:
            potencial_repulsivo += 0

    # Potencial total
    return potencial_atractivo + potencial_repulsivo

# Funci n para calcular gradiente del campo de potencial
def calcular_gradiente(posicion_agente, objetivo, obstaculos, epsilon=0.15):
    gradiente = np.zeros(2)
    for i in range(2):
        delta_pos = np.zeros(2)
        delta_pos[i] = epsilon
        potencial_pos = calcular_potencial(posicion_agente + delta_pos, objetivo, obstaculos)
        potencial_neg = calcular_potencial(posicion_agente - delta_pos, objetivo, obstaculos)
        gradiente[i] = (potencial_pos - potencial_neg) / (2 * epsilon)
    return gradiente

# Clase para el agente BDI
class AgenteBDI:
    def __init__(self, posicion_inicial, objetivo, obstaculos):
        self.posicion = posicion_inicial
        self.objetivo = objetivo
        self.obstaculos = obstaculos
        self.estado = "navegando"
        self.contador_atascado = 0
        self.historial_posiciones = [posicion_inicial.copy()]
        self.direccion_escape = None
        self.contador_escape = 0
        self.puntos_atasco = []
        self.contador_evitacion = 0
        self.direccion_evitacion = None

    def actualizar_creencias(self):
        if len(self.historial_posiciones) > 10:
            desplazamiento = np.linalg.norm(self.historial_posiciones[-1] - self.historial_posiciones[-10])
            if desplazamiento < 0.5:
                self.estado = "atascado"
                self.contador_atascado += 1
                es_nuevo_atasco = True
                for punto in self.puntos_atasco:
                    if np.linalg.norm(self.posicion - punto) < 1.0:
                        es_nuevo_atasco = False
                        break
                if es_nuevo_atasco:
                    self.puntos_atasco.append(self.posicion.copy())
            else:
                self.estado = "navegando"
                self.contador_atascado = 0

        if np.linalg.norm(self.posicion - self.objetivo) < 0.5:
            self.estado = "completado"

```

```
74          for punto in self.puntos_atasco:
75              if np.linalg.norm(self.posicion - punto) < 2.0:
76                  self.estado = "evitando"
77                  break
78
79      def generar_deseos(self):
80          if self.estado == "completado":
81              return "permanecer"
82          elif self.estado == "atascado" and self.contador_atascado > 5:
83              return "escapar"
84          elif self.estado == "evitando":
85              return "evitar"
86          else:
87              return "navegar"
88
89      def planificar_intenciones(self, deseo):
90          if deseo == "permanecer":
91              return np.zeros(2)
92          elif deseo == "escapar":
93              punto_mas_cercano = None
94              distancia_minima = float('inf')
95              for punto in self.puntos_atasco:
96                  distancia = np.linalg.norm(self.posicion - punto)
97                  if distancia < distancia_minima:
98                      distancia_minima = distancia
99                      punto_mas_cercano = punto
100             if punto_mas_cercano is not None:
101                 direccion_alejamiento = self.posicion - punto_mas_cercano
102                 if np.linalg.norm(direccion_alejamiento) < 0.1:
103                     angulo = np.random.uniform(0, 2*np.pi)
104                     direccion_alejamiento = np.array([np.cos(angulo), np.sin(
                            angulo)])
105                 else:
106                     direccion_alejamiento = direccion_alejamiento / np.linalg.
                            norm(direccion_alejamiento)
107                 direccion_objetivo = self.objetivo - self.posicion
108                 if np.linalg.norm(direccion_objetivo) > 0:
109                     direccion_objetivo = direccion_objetivo / np.linalg.norm(
                            direccion_objetivo)
110                 direccion_combinada = 0.6 * direccion_alejamiento + 0.4 *
                        direccion_objetivo
111                 direccion_combinada = direccion_combinada / np.linalg.norm(
                        direccion_combinada)
112                 self.direccion_escape = direccion_combinada
113                 self.contador_escape = 0
114             else:
115                 if self.direccion_escape is None or self.contador_escape > 20:
116                     angulo = np.random.uniform(0, 2*np.pi)
117                     self.direccion_escape = np.array([np.cos(angulo), np.sin(
                            angulo)])
118                     self.contador_escape = 0
119                 else:
120                     self.contador_escape += 1
121             return self.direccion_escape * 0.3
122         elif deseo == "evitar":
123             puntos_cercanos = []
124             for punto in self.puntos_atasco:
125                 if np.linalg.norm(self.posicion - punto) < 2.0:
126                     puntos_cercanos.append(punto)
127             direccion_evitacion = np.zeros(2)
128             for punto in puntos_cercanos:
129                 direccion_punto = self.posicion - punto
```

```
                if np.linalg.norm(direccion_punto) > 0:
                    direccion_punto = direccion_punto / np.linalg.norm(
                        direccion_punto)
                    peso = 1.0 / (0.1 + np.linalg.norm(self.posicion - punto))
                    direccion_evitacion += peso * direccion_punto
            if np.linalg.norm(direccion_evitacion) > 0:
                direccion_evitacion = direccion_evitacion / np.linalg.norm(
                    direccion_evitacion)
            direccion_objetivo = self.objetivo - self.posicion
            if np.linalg.norm(direccion_objetivo) > 0:
                direccion_objetivo = direccion_objetivo / np.linalg.norm(
                    direccion_objetivo)
            direccion_combinada = 0.5 * direccion_evitacion + 0.5 *
                direccion_objetivo
            if np.linalg.norm(direccion_combinada) > 0:
                direccion_combinada = direccion_combinada / np.linalg.norm(
                    direccion_combinada)
            return direccion_combinada * 0.15
        else:
            return -calcular_gradiente(self.posicion, self.objetivo, self.
                obstaculos) * 0.1

    def ejecutar(self):
        """Ejecuta un ciclo completo BDI"""
        self.actualizar_creencias()
        deseo = self.generar_deseos()
        movimiento = self.planificar_intenciones(deseo)

        # Proponer nueva posici n
        nueva_posicion = self.posicion + movimiento

        # Verificaci n de colisi n con obst culos
        colision = False
        for obstaculo in self.obstaculos:
            if np.linalg.norm(nueva_posicion - obstaculo) < 0.5:  # margen de
                seguridad
                colision = True
                break

        if not colision:
            self.posicion = nueva_posicion
        else:
            # Si choca, prueba un peque o desv o aleatorio
            angulo = np.random.uniform(0, 2*np.pi)
            self.posicion += 0.1 * np.array([np.cos(angulo), np.sin(angulo)])

        self.historial_posiciones.append(self.posicion.copy())
        return self.posicion, calcular_potencial(self.posicion, self.objetivo,
             self.obstaculos)


# Configuraci n inicial de la visualizaci n
fig, (ax, ax2) = plt.subplots(1, 2, figsize=(12, 5))

agente_posicion = np.array([1.0, 1.0])
objetivo = np.array([12, 12])
obstaculos = np.array([[2, 2],[2, 3], [2, 4],[2, 5],[2, 6],[2, 7],[2, 8],[2,
    9], [2, 10],[10, 10],
                       [10, 9],[10, 8],[10, 7],[10, 6],[10, 5],[10, 4],[10,
                           3],[10, 2],
                       [9, 2], [9, 3],[9, 4],[9, 5],[9, 6],[9, 7],[9, 8],[9,
                           9],
```

```
                        [3, 9],[3, 8],[3, 7],[3, 6],[3, 5],[3, 4],[3, 3], [3,
                            2],
                        [4,9],[5,9],[6,9],[7,9],[8,9],[3,10],[4,10],[5,10],[6,10],

                        [7,10],[8,10],[9,10]])

agente = AgenteBDI(agente_posicion, objetivo, obstaculos)
historial_energia_potencial = []

def update(frame):
    global historial_energia_potencial
    nueva_posicion, energia_potencial = agente.ejecutar()
    historial_energia_potencial.append(energia_potencial)

    ax.clear()
    ax.scatter(nueva_posicion[0], nueva_posicion[1], color='red', marker='o',
        s=200, label='Agente')
    ax.scatter(objetivo[0], objetivo[1], color='green', marker='x', s=100,
        label='Objetivo')

    for obstaculo in obstaculos:
        ax.scatter(obstaculo[0], obstaculo[1], color='black', marker='s', s
            =100)

    for i, punto in enumerate(agente.puntos_atasco):
        ax.scatter(punto[0], punto[1], color='orange', marker='*', s=150,
            label='Punto de atasco' if i == 0 else "")
        circulo = plt.Circle((punto[0], punto[1]), 2.0, color='orange', fill=
            False, alpha=0.3, linestyle='--')
        ax.add_patch(circulo)

    historial = np.array(agente.historial_posiciones)
    ax.plot(historial[:, 0], historial[:, 1], 'b--', alpha=0.5)

    ax.set_xlim(0, 15)
    ax.set_ylim(0, 15)
    ax.set_title(f'Agente BDI - Estado: {agente.estado}')
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.legend()

    ax2.clear()
    ax2.plot(historial_energia_potencial, label='Energ a Potencial')
    ax2.set_title('Evoluci n de la Energ a Potencial')
    ax2.set_xlabel('Iteraci n')
    ax2.set_ylabel('Energ a Potencial')
    ax2.legend()

ani = FuncAnimation(fig, update, frames=500, interval=200)
plt.show()
```