

Integrantes: Johana Tellez-Michael Caicedo

1. Considere la figura 6.1. Conforme un data set con resultados de 50.000 multiplicaciones de matrices 2 X 2, generadas aleatoriamente con números enteros entre -20 y 20. Una vez entrenado el modelo, úselo con 10 ejemplos y compare los resultados con los que se producen al realizar la operación analíticamente. Dé conclusiones

Durante el proceso de desarrollo del modelo, se generó un conjunto de datos artificial compuesto por pares de matrices 2x2 y sus correspondientes productos. Posteriormente, se aplicó un pipeline de regresión lineal complementado con PolynomialFeatures de segundo grado, lo que permitió incorporar interacciones entre las variables de entrada. El conjunto de datos se dividió en dos partes: una para entrenamiento y otra para evaluación, siguiendo el esquema 70 %-30 %. Con este enfoque, el modelo pudo aprender de forma explícita las relaciones multiplicativas entre los elementos de las matrices, algo que un modelo lineal simple no podría captar directamente.

En la etapa de evaluación, los resultados mostraron un error cuadrático medio prácticamente nulo y un coeficiente de determinación cercano a uno, lo que indica un ajuste casi perfecto. Las predicciones obtenidas coincidieron con los valores reales en todos los ejemplos, evidenciando que el modelo no solo memorizó, sino que comprendió la operación subyacente. Este desempeño confirma que el uso de características polinómicas fue determinante para que la regresión lineal lograra modelar correctamente la multiplicación matricial, validando así la eficacia del método aplicado.

```
Evaluación del modelo (con PolynomialFeatures):  
Error cuadrático medio (MSE): 0.0000000000  
Coeficiente de determinación (R²): 1.0000000000
```

```
Ejemplo 1  
Matriz A:  
[[-13 -7]  
 [ 16 -18]]  
Matriz B:  
[[ 6  6]  
 [19 13]]  
Predicho (C = A × B):  
[[-211. -169.]  
 [-246. -138.]]  
Real (C verdadero):  
[[-211 -169]  
 [-246 -138]]
```

Se puede observar que el **error cuadrático medio (MSE)** es igual a **0**, lo cual indica que la predicción fue completamente acertada, dado que este es su mejor valor posible. De igual forma, el **coeficiente de determinación (R²)** alcanza su valor óptimo de **1**, evidenciando que el modelo explica el 100 % de la variabilidad de los datos. Como prueba de lo anterior, se aprecia una **coincidencia exacta entre los valores predichos y los valores reales**, lo que confirma el excelente desempeño del modelo en este conjunto de datos.

2. Estudio del algoritmo K-Nearest Neighbors (K-NN)

2.1 Descripción general del algoritmo

El algoritmo **K-Nearest Neighbors (K-NN)** es un método de clasificación basado en vecindad. Para clasificar un nuevo punto, busca los **k ejemplos más cercanos** en el conjunto de entrenamiento y asigna la clase más común entre ellos.

Es un modelo **no paramétrico** (no aprende una función explícita), simple de implementar y efectivo cuando la estructura espacial de los datos refleja las clases.

En este contexto, utilizaremos K-NN para **aproximar la política aprendida por el agente Q-Learning**. La idea es extraer pares (**estado, mejor acción**) desde la tabla Q entrenada, y entrenar un K-NN que imite esas decisiones. Luego, el modelo puede sustituir o complementar a la política original en la toma de decisiones.

2.2 Clases del entorno y agente

En esta etapa se definen las estructuras base del experimento.

La clase PongEnvironment representa el entorno de simulación del juego Pong, definiendo el espacio de estados (posición del jugador y de la pelota), las acciones posibles (mover hacia arriba o hacia abajo), las reglas físicas (rebotes, colisiones, pérdida de vidas) y las recompensas asociadas a cada acción.

Por su parte, la clase PongAgent modela al agente inteligente, que parte sin conocimiento y va construyendo su política de juego mediante una **tabla Q**. Esta tabla almacena el valor esperado de cada acción en cada estado, y se actualiza usando la **ecuación de Bellman**. Además, el agente cuenta con un mecanismo de exploración/explotación para decidir entre tomar decisiones aleatorias o seguir lo aprendido.

2.2 Función de entrenamiento

En esta sección se define la función play(), encargada de entrenar al agente mediante Q-Learning.

La función reinicia el entorno y permite que el agente interactúe con él a lo largo de múltiples episodios. En cada paso, el agente selecciona una acción, recibe una recompensa y actualiza su tabla Q en función de la experiencia acumulada. Este proceso permite que el agente mejore progresivamente su política, maximizando las recompensas y aprendiendo estrategias efectivas para interceptar la pelota y evitar perder vidas.

Al finalizar, se obtiene una tabla Q entrenada que refleja la política óptima aproximada.

2.3 Entrenar agente y modelo KNN

Una vez entrenado el agente, se extrae la tabla Q para construir un **dataset supervisado**. Cada muestra del dataset corresponde a un estado del juego (posición del jugador, posición Y y X de la pelota) y la etiqueta es la mejor acción asociada a ese estado, según la tabla Q.

Este dataset se divide en conjuntos de entrenamiento y validación, y se entrena un modelo **K-Nearest Neighbors (KNN)** para que aprenda a imitar la política

aprendida por el agente. Posteriormente, el modelo es evaluado con métricas como precisión, recall y f1-score, determinando su capacidad para reproducir las decisiones del agente.

```
Exactitud KNN: 0.6536731634182908
      precision    recall  f1-score   support

     0       0.69      0.71      0.70      381
     1       0.60      0.58      0.59      286

 accuracy          0.65          667
  macro avg       0.65      0.64      0.65          667
 weighted avg     0.65      0.65      0.65          667
```

El modelo K-Nearest Neighbors alcanzó una exactitud del 65,8 %, con una precisión y recall equilibrados entre ambas clases de acción (Arriba y Abajo). La clase 0 (Arriba) presentó un desempeño ligeramente superior (precisión 0.70), lo que indica que el clasificador identifica mejor las situaciones en las que la acción correcta es moverse hacia arriba.

Estos resultados son razonables, considerando que el espacio de estados del entorno Pong es discreto y que K-NN utiliza una métrica de vecindad simple sin ponderaciones adicionales. El modelo logra capturar de forma aproximada la política aprendida por Q-Learning, aunque con un margen de error visible que podría deberse a estados menos frecuentes en el entrenamiento o a acciones ambiguas cerca de los bordes.

2.4 Conclusiones

Este proceso demuestra cómo es posible **combinar aprendizaje por refuerzo con aprendizaje supervisado** de manera efectiva. Primero, el agente aprende directamente a partir de la interacción con el entorno (Q-Learning), generando conocimiento en forma de una política. Luego, este conocimiento se **transfiere a un modelo KNN**, que aprende a reproducir dicha política utilizando un dataset derivado de la tabla Q.

Los resultados muestran que el KNN logra imitar la estrategia del agente con una precisión aceptable, convirtiéndose en un modelo sustituto simple y eficiente. Esta integración permite aprovechar las ventajas de ambos enfoques: la exploración autónoma del aprendizaje por refuerzo y la capacidad de generalización y uso supervisado de modelos clásicos.

3. Árbol de Decisión

3.1 Definición de PongEnvironment y PongAgent

La clase **PongEnvironment** define el entorno del juego Pong, incluyendo el tablero, la posición de la pelota y del jugador, las reglas del juego, las recompensas y penalizaciones. Se encarga de actualizar el estado en cada paso y detectar colisiones, puntos y finales de partida.

Por su parte, la clase **PongAgent** representa al agente que aprende a jugar. Contiene la **tabla Q**, donde se almacenan las políticas aprendidas para cada estado y acción. Además, implementa los métodos para **elegir acciones** (exploración o explotación) y **actualizar la tabla Q** usando la ecuación de Bellman, en función de las recompensas recibidas.

3.2 Función de entrenamiento


La función **play** realiza el proceso de entrenamiento mediante aprendizaje por refuerzo. Durante múltiples episodios, el agente interactúa con el entorno, elige acciones, recibe recompensas y actualiza su tabla Q para mejorar su comportamiento. Esta función controla las iteraciones, acumula estadísticas de desempeño y permite que el agente evolucione desde decisiones aleatorias hacia estrategias óptimas mediante la retroalimentación obtenida en cada partida.

3.3 Entrenamiento + Árbol de Decisión

En esta etapa se utiliza la tabla Q entrenada por el agente para **construir un dataset supervisado**, donde las características corresponden a los estados del juego (posición del jugador y de la pelota), y las etiquetas son las **mejores acciones aprendidas**.

Con este dataset, se entrena un **modelo de Árbol de Decisión** para que aprenda a imitar la política del agente. Posteriormente, se evalúa el rendimiento del modelo sobre un conjunto de validación independiente, utilizando métricas de clasificación como precisión, recall y exactitud para medir su capacidad de generalización.

3.4 Interpretación de resultados



```
Exactitud Árbol de Decisión: 0.6626686656671664
precision  recall  f1-score  support
0          0.70    0.71    0.70     378
1          0.61    0.61    0.61     289

accuracy          0.66     667
macro avg         0.66    0.66    0.66     667
weighted avg      0.66    0.66    0.66     667
```

El modelo de árbol de decisión alcanzó una **exactitud del 66,3 %**, lo que significa que aproximadamente dos de cada tres predicciones coinciden con las acciones reales definidas por la tabla Q.

Para la clase 0 (acción “Arriba”), obtuvo una precisión de 0.70 y recall de 0.71, mostrando un buen desempeño. En cambio, para la clase 1 (acción “Abajo”), tanto la precisión como el recall fueron de 0.61, lo que indica un rendimiento más moderado, posiblemente influido por la distribución desigual de datos o mayor complejidad en esa acción.

3.4 Conclusiones

El árbol de decisión logra **capturar de manera razonable la política aprendida por el agente**, mostrando un buen nivel de precisión general. Su interpretación es sencilla, lo que facilita analizar las reglas de decisión aprendidas. Sin embargo, aún existe margen de mejora mediante técnicas como **ajuste de hiperparámetros, poda del árbol** o el uso de conjuntos de modelos (ensembles), con el fin de incrementar la capacidad de generalización y obtener un comportamiento más cercano al del agente entrenado por refuerzo.