Escuela de
Ciencia y Tecnología
ECyT_UNSAM

Red neuronal

"Estadísticas de monstruos de Dungeons and Dragons 5ª edición"

ALUMNO
Gonzalo Manuel Arias

MATERIA: Matemática III

Docente: Josefina Bompensieri y Tomás Prudente

UNSAM, Escuela de Ciencia y Tecnología.

2024

Parte 1: Análisis de la base de datos

Selección de la base de datos:

La base de datos a utilizar es una base de datos de clasificación llamada “D&D 5e Monster stats” por Jairo Hernandez. Puede encontrarse haciendo click aquí.

Esta base de datos fue elegida por el uso de datos sintéticos, para un juego de rol de sobremesa o llamado TTRPG, que es con el tipo de datos que destaca el uso de redes neuronales. También por el uso posible del autor y círculos de jugadores, tanto para la creación de juegos con monstruos personalizados, con un mejor balance en sus estadísticas.

Descripción de cada columna:

- Name: Variable categórica. Indica el nombre del monstruo.
- Armor Class: Variable cuantitativa discreta. Indica el número mínimo a sacar en los dados para dañar al monstruo.
- Hit points: Variable cuantitativa discreta. Indica el número de puntos de daño que hay que hacerle al monstruo para derrotarlo.
- Challenge Rating (CR): Variable cuantitativa continua. Una aproximación de para que nivel aproximado de grupo podrían ser recomendables los monstruos.
- Ability scores: Variables cuantitativas discretas. Indican las distintas estadísticas de combate y rol de los monstruos.
    - Strength (STR): Fuerza.
    - Dexterity (DEX): Destreza.
    - Constitution (CON): Constitución.
    - Intelligence (INT): Inteligencia.
    - Charisma (CHA): Carisma.
- Descriptions: Variables categóricas. Describen el comportamiento de los monstruos.
    - Type: Tipo de monstruo.
    - Alignment: Indica el alineamientos de disposición ética correspondiente al monstruo de los 9 existentes en el juego.

Análisis de correlaciones:

	Armor Class	Hit Points	STR	DEX	CON	INT	WIS	CHA	CR
Armor Class	1.000000	0.689411	0.559285	0.259032	0.655396	0.590243	0.476521	0.639579	0.739156
Hit Points	0.689411	1.000000	0.737359	-0.083645	0.868687	0.490320	0.442377	0.592163	0.933268
STR	0.559285	0.737359	1.000000	-0.218743	0.848086	0.310819	0.344478	0.429674	0.677147
DEX	0.259032	-0.083645	-0.218743	1.000000	-0.155937	0.273928	0.358252	0.269375	0.013128
CON	0.655396	0.868687	0.848086	-0.155937	1.000000	0.425709	0.409846	0.541861	0.824834
INT	0.590243	0.490320	0.310819	0.273928	0.425709	1.000000	0.632674	0.893249	0.604942
WIS	0.476521	0.442377	0.344478	0.358252	0.409846	0.632674	1.000000	0.703053	0.536870
CHA	0.639579	0.592163	0.429674	0.269375	0.541861	0.893249	0.703053	1.000000	0.683857
CR	0.739156	0.933268	0.677147	0.013128	0.824834	0.604942	0.536870	0.683857	1.000000

La columna objetivo es Challenge rating (CR), ya que esta red neuronal se desarrolló con el objetivo de predecir esta característica del monstruo, a partir del resto de columnas.

Se puede ver que las más influyentes al CR, en orden descendiente, son: Hit points, CON, Armor Class, CHA, STR, INT, WIS Y por último DEX. Todas estas son correlaciones positivas, es decir que al aumentar, CR tiende a aumentar también

	Armor Class	Hit Points	STR	DEX	CON	INT	WIS	CHA	CR
Armor Class	1.000000	0.689411	0.559285	0.259032	0.655396	0.590243	0.476521	0.639579	0.589674
Hit Points	0.689411	1.000000	0.737359	-0.083645	0.868687	0.490320	0.442377	0.592163	0.641260
STR	0.559285	0.737359	1.000000	-0.218743	0.848086	0.310819	0.344478	0.429674	0.576087
DEX	0.259032	-0.083645	-0.218743	1.000000	-0.155937	0.273928	0.358252	0.269375	0.033057
CON	0.655396	0.868687	0.848086	-0.155937	1.000000	0.425709	0.409846	0.541861	0.657988
INT	0.590243	0.490320	0.310819	0.273928	0.425709	1.000000	0.632674	0.893249	0.498406
WIS	0.476521	0.442377	0.344478	0.358252	0.409846	0.632674	1.000000	0.703053	0.392173
CHA	0.639579	0.592163	0.429674	0.269375	0.541861	0.893249	0.703053	1.000000	0.540213
CR	0.589674	0.641260	0.576087	0.033057	0.657988	0.498406	0.392173	0.540213	1.000000

Luego de la normalización de los datos se repitió el análisis de correlaciones. En este segundo cuadro se puede ver que se redujeron las correlaciones entre las columnas y la columna objetivo, pero se mantienen siendo las mismas las de mayor correlación (Hit points y constitution (CON).

Análisis de Factibilidad:

Este dataset es considerado adecuado porque los datos que este aporta son sintéticos. Esto es dado ya que fueron creados a partir de un juego de mesa y los monstruos fueron adaptados a través de distintas ediciones del juego.

A través del análisis de correlaciones se pudo observar que los datos tienden a tener correlación con la columna objetivo, lo que es beneficioso para la creación de una red neuronal que pueda ser entrenada y aprenda, sobretodo en el caso de *supervised learning* que es el caso de esta red, al tener los datos etiquetados y es entrenada para predecir resultados.

El objetivo del modelo de esta red es la predicción de si el monstruo creado cumple con las estadísticas necesarias para ser un jefe para un grupo de nivel 4 o si es un monstruo común. Siendo un monstruo con challenge rating mayor o igual a 3, un jefe viable para un grupo de nivel 4 y menor a esto el monstruo puede ser un enemigo común. Esto es beneficioso para el balance en los stats de nuevos monstruos creados por el usuario final, permitiendo ingresar las estadísticas de un monstruo nuevo para una campaña a la red y esta estimando si este es mayormente considerable como jefe o como monstruo común.

Datos Atípicos y Limpieza de Datos:

En este caso se decidió no hacer una limpieza de datos atípicos, principalmente porque la base de datos no presenta una gran cantidad de estos (0,6% de los datos totales).

Esto se puede ver porque la media y la mediana son similares en todas las columnas, a la vez que no hay una gran varianza en los datos.

En el caso de ser necesario eliminarlos, se podría ordenar los valores y eliminar un porcentaje de los extremos, eliminando así máximos y mínimos muy alejados de la media.

Al dejarlos, la red no aprendería de valores muy alejados a lo que se busca que aprenda, por lo cual no es considerado necesario, al no ser considerado un factor que disminuya el aprendizaje de la misma.

	count	mean	std	min	25%	50%	75%	max
Armor Class	428.0	14.233645	3.155841	5.0	12.0	14.0	17.0	25.0
Hit Points	428.0	79.168224	91.606197	1.0	19.0	51.0	110.0	676.0
STR	428.0	15.025701	6.388059	1.0	11.0	16.0	19.0	30.0
DEX	428.0	12.808411	3.128882	1.0	10.0	13.0	15.0	28.0
CON	428.0	15.004673	4.356478	3.0	12.0	14.0	17.0	30.0
INT	428.0	8.488318	5.519593	1.0	3.0	9.0	12.0	25.0
WIS	428.0	11.735981	2.936154	1.0	10.0	12.0	13.0	25.0
CHA	428.0	10.100467	5.535637	1.0	6.0	9.5	14.0	30.0
CR	428.0	4.552278	5.596778	0.0	0.5	2.0	6.0	30.0

En la imagen anterior se pueden ver la media, la mediana y la desviación estándar de cada columna, así mismo como sus máximos y mínimos. La que presenta una mayor desviación estándar y contiene la mayor cantidad de casos atípicos es Hit points, pero incluso con estos datos atípicos no fue considerado relevante para el aprendizaje de la red

Transformaciones Preliminares:

Se hicieron múltiples transformaciones a partir del análisis de los datos.

Inicialmente se hizo una transformación de la columna objetivo (CR):

```
df_nuevo['CR'] = df_nuevo['CR'].apply(lambda x: 0 if x < 3 else 1)
```

Esto se hizo con el objetivo de transformar los valores menores a 3 a 0, que representaría que es un monstruo común y los mayores o iguales a este a 1, que representa que el enemigo puede ser un jefe viable para un grupo de nivel 4.

Para el resto de las columnas se efectuó una normalización:

```
for column in columnas_Normalizadas:
    min_val = df_nuevo[column].min()
    max_val = df_nuevo[column].max()
    df_nuevo[column] = (df_nuevo[column] - min_val) / (max_val - min_val)
```

Esto lo que hace es reescalar los datos entre 0 y 1, así evitar números muy grandes que puedan dificultar las velocidades de procesamiento, a su vez que permite tener una visualización clara de que valores se acercan más al máximo (1) y al mínimo (0).

La última transformación hecha fue eliminar los datos no numéricos, al ser datos exclusivamente para el rol y no influir en la dificultad de un enemigo (como nombre, tipo o alineamiento).

Una vez efectuadas estas transformaciones, se procedió a la creación de un nuevo archivo con únicamente estos datos.

Parte 2: Desarrollo de la Red Neuronal

Arquitectura de la red:

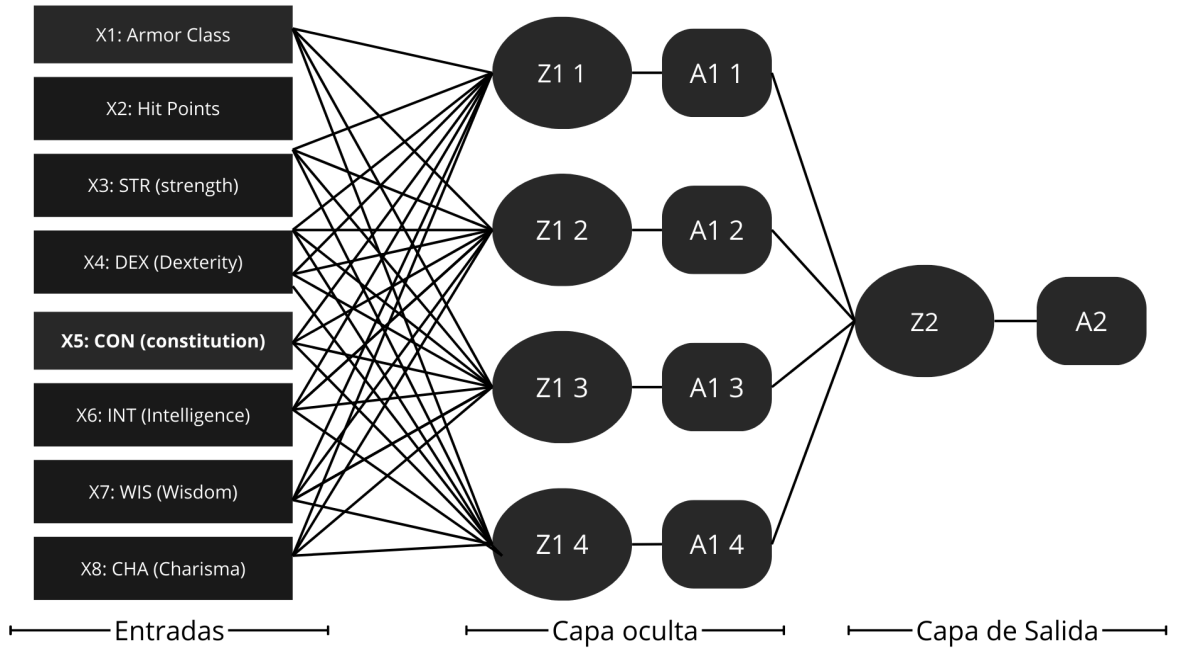
La arquitectura de esta red neuronal está diseñada de la siguiente forma:

- 8 Entradas.
- 1 Capa oculta con 4 neuronas.
  - Cada uno de ellos con activación ReLU.
- 1 Capa de salida con 1 nodo.
  - Con una capa de activación sigmoidea.

Se seleccionó esta cantidad de capas y neuronas ya que al ser 8 entradas, fue elegida una cantidad de nodos equivalente a la mitad de estas. El número de neuronas y de capas podría haber variado según la complejidad de la red, con la posibilidad de agregar más capas ocultas y de variar la cantidad de neuronas de cada una.

La función ReLU para las capas ocultas que transforma los valores negativos a 0 y el resto los mantiene en una función lineal con pendiente 1, fue seleccionada con el objetivo de evitar los valores negativos y no transformar los positivos. Esto es para evitar una disminución en el aprendizaje que podrían generar los valores negativos.

La selección de la función sigmoidea para la capa de salida es dada por su capacidad de convertir los datos en un número entre 0 y 1, acercando a 0 los datos negativos y a 1 los positivos, así mismo agrupando los datos cercanos a 0 en el medio. Esto facilita las salidas binarias en redes que dividen sus resultados en valores menores a 0,5 y mayores a 0,5, como es el caso de esta.



Cada línea equivale a un W para la matriz de peso, las que salen desde entrada equivalen a W1 (hay 32 elementos en esa matriz (8 X 4) e influyen en la función Z1. Dentro de Z1 está cada B1 perteneciente a cada neurona (hay 4 de ellos en una matriz 4 X 1). Las funciones A1 tienen cada función de activación ReLU que luego actúan para las entradas de

la capa de salida, a la que llegan 4 pesos (W2) en una matriz 1 X 4 que se les suma el sesgo de Z2 (B2) para luego ser activados por la función sigmoidea de A2.

Implementación de la Red Neuronal:

Esta sección es para complementar lo comentado en el código de la red neuronal, a su vez que dando explicaciones de ciertas decisiones dentro del código.

Paso a paso del desarrollo de la red:

1. Carga de datos de entrada:

   Se asignó la base de datos actualizada para ser analizada en la red, junto con las 8 columnas que van a ser las entradas (X) y la columna de salida (CR), así como la cantidad total de datos.

   Junto con esto se designaron ⅔ del dataset para su entrenamiento y ⅓ para el testeo.

2. Inicialización aleatoria de los pesos y sesgos:

```
np.random.seed(10)
w_hidden = np.random.rand(4, 8) *2 - 1
w_output = np.random.rand(1, 4) *2 - 1

b_hidden = np.random.rand(4, 1) *2 - 1
b_output = np.random.rand(1, 1) *2 - 1
```

   Se inicializan aleatoriamente pero con una semilla para una fácil replicación de los datos. Los pesos de la primer capa están en una matriz de 4 x 8, los de la segunda capa en una de 1 x 4. Los sesgos de la primer capa están en una matriz de 4 X 1 y el de la segunda es un número.

   Cada valor aleatorio se multiplicó por dos y se le restó uno, para mantener los valores entre -1 y 1, teniendo así valores negativos y positivos.

3. Se definieron las funciones de activación a utilizar.
4. Se definió un *forward propagation* con una capa oculta y una capa de salida:

   Se obtienen a partir de esta Z1, A1, Z2 y A2.

5. Se calcula la precisión inicial de los datos de prueba:

   Esta da un número cercano a 0,5, lo cual es muy bajo, pero es porque los pesos y sesgos fueron aleatorizados. Se hizo esta precisión inicial para comparar con la precisión de la red entrenada al final.

6. Se definió un *backpropagation*, a su vez que un paso de aprendizaje (L) y una cantidad de iteraciones:

   Estas fueron cambiadas más adelante una vez entrenada la red neuronal.

   Este back propagation recibía los valores de la *forward propagation* y los utilizaba para actualizar los pesos y sesgos. Y a partir de estos valores, con la derivada de la función de costo, evaluada en W1, W2, B1 y B2 se puede ver la influencia de cada uno en el resultado final, haciendo que en el siguiente paso sea posible encontrar el valor correcto para cada una de estas variables.

7. Establecer un descenso de gradiente estocástico:

   Se estableció un ciclo con 100.000 iteraciones en el que se seleccionaba un X y un Y del grupo de entrenamiento seleccionado al inicio.

   Se efectúa un *forward propagation* con este valor de X y luego un *back propagation*. Luego de esto se actualizan W1, W2, B1 y B2 para la siguiente

iteración. A partir de continuar iterando, la red consigue acercarse a los valores más acertados de cada una de estas variables.

8. Cálculo de precisión de la red entrenada:

   Se hace nuevamente el cálculo de precisión de la red, esta vez para los datos de entrenamiento y por separado con los casos de prueba. Estos resultados nos demuestran que tan acertadas fueron las predicciones de las variables modificables (W1, W2, B1 y B2) tanto para casos en los que ya tiene el resultado final (los de entrenamiento) como para datos en los que no se tiene los resultados finales (los de prueba).

   Esto da aproximadamente un 93%, que significa que en el 93% de casos, la red es capaz de predecir si el monstruo a evaluar va a ser un jefe o un monstruo común para un grupo de nivel 4.

9. Permitir una entrada de datos de usuario para el uso de esta red:

   Finalmente, se permite al usuario ingresar un dataset de un monstruo, este es normalizado con los máximos y mínimos de la base de datos original y se hace un *forward propagation* que dará como resultado A2, que en caso de ser mayor a 0,5, esta red devolverá “Jefe” indicando que este monstruo es viable como jefe para un grupo de nivel 4, o devolverá “monstruo común” indicando que no es viable como jefe.

Evaluación de la Red Neuronal:

Una vez finalizada la red neuronal y encontrados los valores óptimos para los pesos y sesgos, se hizo un refinamiento de ciertos valores inicializados, específicamente el paso de aprendizaje (L) y las iteraciones para un mejor aprendizaje, para esto se hizo un archivo que repite la red neuronal con una misma semilla aleatoria múltiples veces, cambiando estas variables y graficando los análisis de precisión tanto para las variables de entrenamiento como para las de prueba, para poder establecer una comparación entre estas.

Se seleccionaron como posibles las siguientes iteraciones:

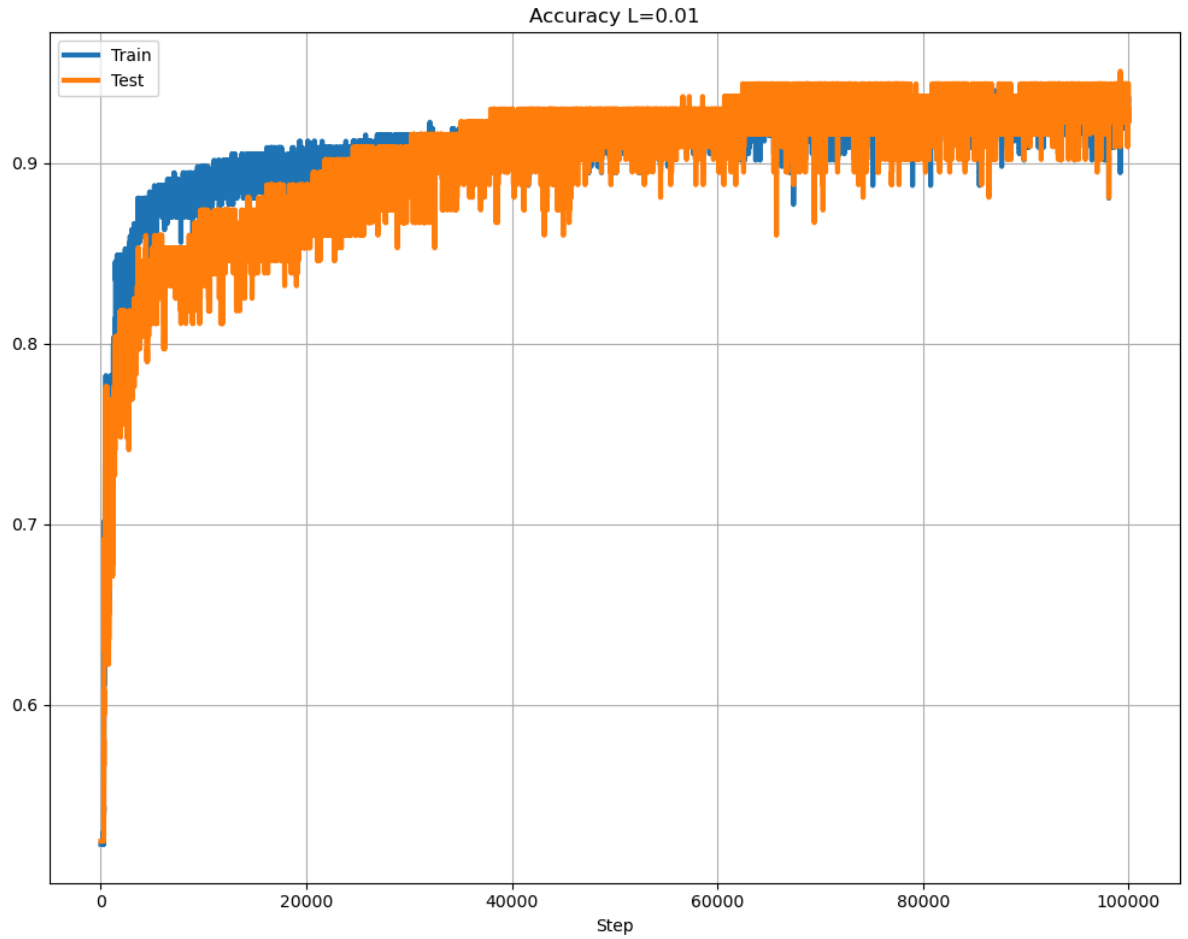
```
iters_l = (300,3000,30000,100_000)
L_l = (0.05, 0.01, 0.005, 0.001)
```

Una vez hechas las combinaciones posibles con estas iteraciones y pasos de aprendizaje se procedió a comparar los gráficos de aprendizaje, teniendo en cuenta cual tiene un mayor porcentaje de precisión sin mostrar overfitting (memorización de los datos de entrenamiento).

Para una mejor organización de los datos, el conjunto de imágenes puede ser visto en la carpeta llamada “accuracy” en el código de la red neuronal.

Con L = 0,01 y las iteraciones = 100.000 este gráfico mantiene su precisión por encima del 90% tanto para el análisis de las muestras de entrenamiento como para las de prueba, sin mostrar *overfitting*, es decir que no memorice los datos de las X de entrenamiento.

El gráfico del paso de aprendizaje e iteraciones elegido es el siguiente:



Análisis de *overfitting*:

El overfitting es una memorización de los datos de entrenamiento por parte de la red que hace que sea muy sensible ante los datos de prueba y falle. Esto se da al no haber aprendido una relación entre los datos de entrada y la salida, si no que es una replicación de los datos de entrenamiento para el resto de la población. Esto se puede evitar con el quitado de datos atípicos (el *overfitting* es muy sensible a varianzas altas) y evitando entrenar por una cantidad de iteraciones mayores a las necesarias, dependiendo del paso de aprendizaje.

Se puede ver en los gráficos de precisión en los casos que da overfitting, ya que esta disminuye considerablemente en los datos de prueba, lo que hace que sea mejor mantener las precisiones hasta antes de bajar esta precisión.

Parte 3: Comparación con Scikit-Learn

Se implementó una red neuronal con la librería Scikit-Learn similar a la hecha manualmente, con una estructura similar.

```
nn = MLPClassifier(solver='sgd',
    hidden_layer_sizes=(4, ),
    activation='relu',
    max_iter=100_000,
    learning_rate_init=.01)
```

Se mantiene la misma cantidad de nodos en la capa oculta (4), se utiliza ReLU como función de activación (no se utiliza la sigmoidea para la capa de salida) y se mantienen las iteraciones y el paso de aprendizaje a la red hecha manualmente.

Comparación:
Los resultados de precisión de la red neuronal hecha con Scikit-Learn son los siguientes:

```
Puntaje del conjunto de entrenamiento: 0.912281
Puntaje del conjunto de prueba: 0.923077
```

Y los obtenidos con la red neuronal hecha manualmente los siguientes:

```
Puntaje del conjunto de entrenamiento: 0.9438596491228071
Puntaje del conjunto de prueba: 0.9300699300699301
```

Los valores conseguidos con la red neuronal son ligeramente mayores, y posee la ventaja que al poder modificar los funcionamientos internos se puede conseguir mejorar aún más esta precisión, a su vez que también se tiene una mayor flexibilidad y potencia al poder modificar el algoritmo y no tener que depender de una librería hecha por terceros.

Scikit-Learn tiene la ventaja de ser una interfaz cómoda en la que poder hacer rápidamente una red neuronal con una precisión buena y mismo para comparar resultados con los que armar una red neuronal propia.

Usar librerías de terceros como Scikit-Learn puede traer ventajas de comodidad y velocidad que muchas veces es preferible a una mayor personalización y potencia, dependiendo del proyecto a desarrollar.

Parte 4: Conclusión final

El desarrollo de una red neuronal desde 0 me permitió conocer en profundidad cómo es que puede funcionar el *machine learning* y aplicarlo a interés o proyectos personales, como es mi caso, sobre un juego de mesa que me gusta que permite mucha personalización del jugador, al coste de quizá hacer un juego poco balanceado sin querer.

El conocer sobre las redes neuronales ayuda para saber cómo puede llegar a funcionar las inteligencias artificiales y ciertos scripts que antes desconocía como el reconocimiento de imágenes, en proyectos de muy alto nivel.

Llevado a escalas más bajas, también me aporta un código completamente conocido y desarrollado por mí para ayuda del desarrollo de futuras redes neuronales como pequeños proyectos, e incluso para poder incluirlas en proyectos a mayor escala. También para casos en los que necesite una red neuronal más rápida pude ver como se usa Scikit Learn y en base a eso también poder aprender otras librerías que tienen usos similares.