Título del trabajo (elegir uno original)

Domínguez-Adame Ruiz, Alberto
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
albdomrui@alum.us.es

Vilaplana de Trias, Francisco David
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
fravilde1@alum.us.es

***Resumen*—El objetivo principal de este trabajo es diseñar y evaluar distintos modelos de aprendizaje automático relacional para clasificar los nodos de un grafo a partir de sus propiedades, y ver como interactua el modelo antes y después de introducir los atributos relacionales. Para este fin, hemos realizado cuatro modelos diferentes: KNN, Naive Bayes, Árbol de Decisión y Redes Neuronales. Primeramente construimos el modelo, lo entrenamos y lo evaluamos, todo esto sin atributos relacionales. Seguidamente se calculan los atributos relacionales, en nuestro caso Centralidad de intermediación, agrupamiento (clustering) y grado (Degree). A continuación repetimos los pasos después de añadir los atriutos relacionales a el conjunto de datos. Finalmente comparamos los resultados relacionales con los no relacionales para concluir si los atributos relacionales mejoran o no el rendimiento de los modelos, y entonces elegiremos el modelo que nos haya proporcionado mejores resultados en general (relacionales y no relacionales).**

***Palabras clave*—Inteligencia Artificial, KNN, Naive Bayes, Árbol de Decisión, Redes Neuronales, Aprendizaje Automático Relacional...**

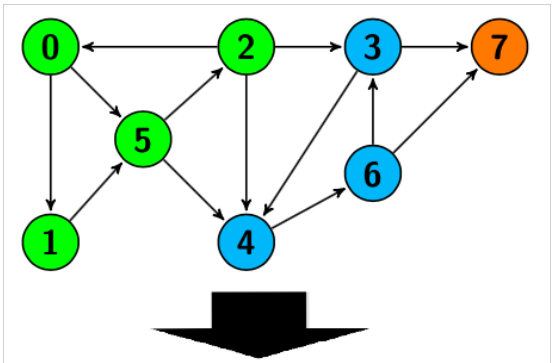
I. Introducción

Para empezar, el aprendizaje se define como la adquisición del conocimiento de algo por medio del estudio, el ejercicio o la experiencia, en especial de los conocimientos necesarios para aprender algún arte u oficio. Al hablar sobre el aprendizaje automático estaríamos entrando en el campo de la Inteligencia Artificial (IA). Podemos definir esta como un programa de computación diseñado para realizar determinadas operaciones que se consideran propias de la inteligencia humana, como el Aprendizaje Automático (Machine Learning), una rama de la inteligencia artificial, cuyo objetivo es el desarrollo de un sistema (modelo matemático que realiza una determinada tarea) el cual tiene un mejor desempeño con la experiencia, dados una serie de datos.

Nuestro estudio hace uso de datos relacionales, los cuales están unidos entre sí (tienen una relación) que queda representado como aristas en un grafo, entendiendo un grafo como un conjunto de nodos unido (o no) mediante aristas. En la Fig. 1 podemos ver un ejemplo de un grafo a partir de unos datos relacionales.

Para la realización de este trabajo se ha hecho uso del entorno de desarrollo Jupyter y de la herramienta notebook. El código está escrito mediante el lenguaje de programación Python, usando principalmente las librerías pandas, NetworkX y scikit-learn, y otras como keras, tensorflow y matplot.

Los modelos de clasificación usados son: KNN, Naive Bayes, Árboles de Decisión y Redes Neuronales.



NodeID	PR
1	0.1
2	0.3
...	...

NID_From	NID_To
1	5
2	3
...	...

Fig. 1. Ejemplo de Grafo

II. Conjunto de datos y Tarea de predicción a realizar

Hemos elegido un dataset (fíjese en la Fig. 2) de la página de streamings en directo twitch.tv, en la que se ve: las visitas, días (ambos de tipo entero) que transmitido un canal en directo, si tiene contrato o no con twitch (partner de tipo Boolean) y si el contenido del canal (id de tipo Integer) es o no para adultos (mature de tipo Boolean). Al ser la propia Twitch la que ofrece este contrato, hemos decidido que la predicción para nuestros modelos sea el atributo partner (true/false). Las aristas (edges) representan si dos usuarios tienen una relación de amistad.

Este dataset tenía dos tipos de id asociados a cada usuario, uno generado aleatoriamente y otro donde al usuario se le asignaba un valor entre 0 y el numero total de usuarios que recoge el conjunto de datos, por lo que decidimos que eliminar el id generado de forma aleatoria para mayor claridad y sólo hace falta un id por usuario para identificarlos.

Para este estudio hemos decidido que el atributo a predecir será partner. Para ello, haremos uso de los atributos: days, views y mature (el id no es relevante en este caso). Lo hemos planteado como una tarea de clasificación binaria, ya que partner solo tiene dos valores posibles (True o False).

	id	partner	days	views	mature
0	2299	False	1459	9528	0
1	153	False	1629	3615	1
2	397	False	411	46546	1
3	5623	False	953	5863	1
4	5875	False	741	5594	1
...	...	...	...	...	...
7121	3794	False	2624	3174	0
7122	6534	False	2035	3158	1
7123	2041	False	1418	3839	1
7124	6870	False	2046	6208	1
7125	3919	False	1797	3545	0

Fig. 2. Tabla de Datos

	id	partner	days	views	mature	degree	clustering	bcentrality
0	2299	False	1459	9528	0	7	0.142857	0.000434
1	153	False	1629	3615	1	19	0.093567	0.000828
2	397	False	411	46546	1	9	0.055556	0.000044
3	5623	False	953	5863	1	3	0.000000	0.000004
4	5875	False	741	5594	1	2	0.000000	0.000030
...	...	...	...	...	...	...	...	...
7121	3794	False	2624	3174	0	4	0.333333	0.000010
7122	6534	False	2035	3158	1	5	0.100000	0.000046
7123	2041	False	1418	3839	1	3	0.000000	0.000004
7124	6870	False	2046	6208	1	20	0.110526	0.000247
7125	3919	False	1797	3545	0	2	0.000000	0.000002

7126 rows × 8 columns

Fig. 3. Tabla de Datos Relacional

Por último hemos codificado (codificación one-hot) los valores de mature de Booleano a Integer, pasando True-False a 1-0 respectivamente, para que sea más eficiente estimar las probabilidades del atributo partner.

III. Métricas Relacionales

Como este es un trabajo de desarrollo de modelos orientados al aprendizaje automático grafos, hemos elegido una serie de métricas relacionales que aplicaremos como atributos relacionales. En concreto hemos elgido tres métricas:

1) **Betweenness centrality** (Centralidad de intermediación): La centralidad en un grafo puede ser entendida como una medida que determina la relevancia de un nodo dentro del grafo y permite comparar o contrastar dicho vértice con otros. La Betweenness Centrality es una medida de centralidad que cuantifica la frecuencia en la que un nodo se encuentra en el camino más corto entre dos nodos determinados. Cuando en un grafo existen nodos de alta intermediación, estos suelen jugar un rol importante en la estructura a la que pertenecen. Estos nodos también poseen capacidades de ser controladores o reguladores de los flujos de información dentro de la estructura total del grafo.
2) **Clustering** (agrupamiento): es una tarea que tiene como finalidad principal lograr el agrupamiento de nodos, para lograr construir subconjuntos de datos conocidos como Clusters.
3) **Degree** (grado): Número de aristas (relaciones) que inciden sobre el nodo.

IV. Algoritmos de Aprendizaje Automático

A continuación exponemos los algoritmos que hemos utilizado con nuestro dataset:

A. KNN

El algoritmo kNN (del inglés *k Nearest Neighbors*, k vecinos más cercanos), es un clasificador de aprendizaje supervisado no paramétrico, por lo que la cantidad de parámetros del modelo coincide exactamente con la cantidad de ejemplos de entrenamiento. Se basa en la proximidad de los k nodos más cercanos calculando la cercanía a partir de los atributos, si estos fueran discretos sería necesaria una codificación de los mismo.

Analizando nuestro dataset, se puede observar como el atributo "*views*" toma valores de mayor magnitud con respecto al resto lo que provoca que tenga más en cuenta para la predicción del modelo. Con el fin de evitar esta situación se hace uso de la normalización a los atributos del conjunto de entrenamiento. Existe una gran variedad de maneras de normalizar un atributo numérico, nosotros hemos optado por el método min-max [1] que consiste en que a cada atributo restarle el valor mínimo de este y dividir esta diferencia entre la resta del valor máximo y mínimo:

$$v_i' = \frac{v_i - m}{M - m} \tag{1}$$

Para realizar kNN es necesario dar un valor al hiperparámetro k, referido al número de vecinos que se va a tener en cuenta a la hora de clasificar un nuevo dato, nosotros le hemos dado valores del uno al diez para más tarde evaluar el rendimiento de cada uno y elegir el mejor. También existen métodos para calcular la distancia, en nuestro caso hemos usado:

- Distancia de *Hamming* para el dataset sin atributos relacionales:

$$d(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} \mathbb{1}(x_i \neq x_i') \tag{2}$$

  Hemos usado esta métrica ya que según la docmentación de la librería [2] es la recomendada para valores enteros.

- Distancia *Manhattan* para el dataset con atributos relacionales:

$$d(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} |x_i - x_i'| \tag{3}$$

  En este caso es la adecuada al tratarse de valores numéricos reales.

Para el cálculo del modelo ha sido necesaria la clase neighbors de la librería sklearn. Por último, una vez se ha construido el modelo lo evaluamos mediante validación cruzada (cross validation) con 10 pliegues. Se divide el dataset según el numero de pliegues, siendo uno de estos el que se usará para probar el modelo y el resto para entrenarlo, de tal manera que cada pliegue acabe siendo usado como subconjunto de prueba. Cada una de estas iteraciones dará como resultado una tasa de acierto, la media de estas será el valor asociado a cada k. Finalmente se comparan y se escoge el valor de k que obtuviera la media la más alta.

B. Naive Bayes

Naive Bayes se trata de un modelo de clasificación paramétrico, ya que no depende de la cantidad de ejemplos que contega el conjunto de entrenamiento. Los valores deben ser discretos, y en caso contrario deberán someterse a un proceso de discretización, donde se divide el rango de los valores de los atributos en subintervalos, los cuales constituiran los posibles valores de la variable discretizada. En nuestro era necesaria la discretización al tratarse de enteros con una gran diferencia entre los valores. Para realizar la tarea de clasificación se aprende primero los siguientes valores de los parámetros:

- La probabilidad de cada clase c:

$$\mathbb{P}(c) = \frac{N_c}{N} \tag{4}$$

  Donde N_c es la cantidad total de ejemplos que pertenecen a la clase c, y N la cantidad total de los ejemplos que hay en el conjunto de entrenamiento.

- La probabilidad de para cada atributo X, cada posible valor de x el atributo y cada clase c:

$$\mathbb{P}(X = x|c) = \frac{N_{X=x,c}}{Nc} \tag{5}$$

  Donde $N_{X=x,c}$ es la cantidad total de ejemplos del conjunto de entrenamiento que pertenecen a la clase c en los que el atributo X toma el valor x.

A la hora de discretizar los atributos hemos usado KBinsDiscretizer [3], de la libretria sklearn. Para los parámetros usamos una codificación one-hot, ya que algunos atributos tiene valores numéricos muy elevados comparados con otros. Por esta razón hemos elgido esta codificación, pues proporciona mejor resultado que otras (como por ejemplo la de tipo ordinal).

A la hora de construir el modelo se ha usado MultinomialNB(alpha=k) [4], pues es el más apropiado para clasificación de atributos numéricos discretizados. El hiperparámetro usado, k, se usa para realizar el suavizado de Laplace. Dicho suavizado se realiza cuando alguna de las probabilidades calculadas para una clase c toma el valor 0, haciendo que un ejemplo nunca se clasifique en dicha clase c. La fórmula que nos indica el suavizado de Laplace es la siguiente:

$$\mathbb{P}(X = x|c) = \frac{N_{X=x,c} + 1}{N_c + k|X|} \tag{6}$$

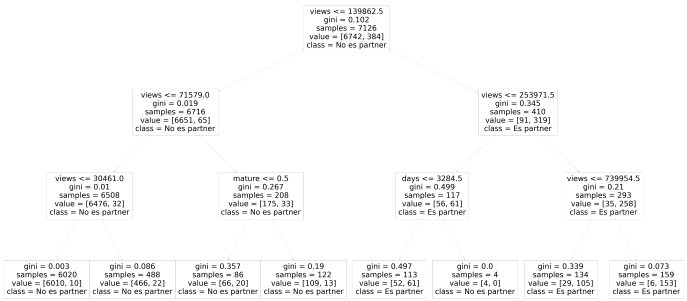


Fig. 4. Árbol de Decisión

Para la evaluación de este modelo también hemos usado el método de validación cruzada con 10 pliegues.

C. Árboles de Decisión

Los Árboles de Clasificación y regresión (del inglés CART, clasification and regresion tree) son modelos para adecuados para tareas de tanto de clasificación como de regresión dados atributos necesariamente numéricos, ya sean continuos o discretos.En un CART cada nodo interno está etiquetado con un atributo y un valor umbral para dicho atributo, mientras que cada hoja está etiquetada con una clase si es una tarea de clasificación, o un número en caso de que se trate de una tarea de regresión. Dado nuestro conjunto de datos, la tarea a abordar es una de clasificación, por lo que elegimos el tipo de árbol correspondiente.

El algoritmo CART para el aprendizaje del modelo es tal que dado un conjunto de ejemplos de entrenamiento $\mathscr{D}$, denotamos $\mathscr{D}_n$ al subconjunto de ejemplos que satisfacen todas las condiciones de los nodos desde la raíz hasta hasta n. Dado un nodo n un atributo X, siendo x_1, x_2, ... , x_k la secuencia de distintos valores que toma X en $\mathscr{D}_n$, estando estos ordenados de menor a mayor. Siendo $u_i = \frac{(x_i + x_{i+1})}{2}$ el umbral candidato para X y con i en el rango $(1, 2, ..., k-1)$. A continuación se divide $\mathscr{D}_n$ según el umbral. Si los ejemplos de X toman un valor menor o igual que u estos se agrupan en $\mathscr{D}_n^{Izq}$, mientras que si son mayores que u se agruparan en en el subconjunto $\mathscr{D}_n^{Der}$. Por último, se hace uso de una función I que permite medir la impureza de un conjunto de ejemplos, para poder elegir el par (X, u) que nos de una mejor partición de $\mathscr{D}_n$, siendo dicha función:

$$\textbf{arg mín}\ \frac{|\mathscr{D}_n^{Izq}|}{|\mathscr{D}|} I(\mathscr{D}_n^{Izq}) + \frac{|\mathscr{D}_n^{Der}|}{|\mathscr{D}|} I(\mathscr{D}_n^{Der}) \tag{7}$$

Una representación del algoritmo sería:

CART($\mathscr{D}$)
1 **Si** *NODIVISIBLE*($\mathscr{D}$) **entonces**
2 **Devolver** un nodo etiquetado con *ETIQUETA*($\mathscr{D}$)
3 **Si no entonces**
4 Elegir el par (X, u) que proporcione la mejor
5 partición $(\mathscr{D}^{Izq}, \mathscr{D}^{Der})$ de $\mathscr{D}$
6 $T_1 \leftarrow$ CART($\mathscr{D}^{Izq}$)
7 $T_2 \leftarrow$ CART($\mathscr{D}^{Der}$)
8 **Devolver** un nodo etiquetado con (X, u) y cuyos
9 hijos sean T_1 y T_2

La función de impureza usada para una tarea de clasificación suele serel índice de Gini, pues estima la probabilidad de clasificar incorrectamente un ejemplo. Tal que:

$$G(\mathscr{D}) = \sum_{c \in C} \hat{\pi}_c (1 - \hat{\pi}_c) = 1 - \sum_{c \in C} \hat{\pi}_c^2 \tag{8}$$

Para abordar esta tarea hemos hecho uso de las librerías sklearn y matplotlib para mostrar el árbol resultado por pantalla. Dada la naturaleza hemos decidido usar la función DecisionTreeClassifier de la clase tree, puesto que nos proporciona un árbol de decisión el cual ha sido limitado a una profundidad máxima de 5. Para terminar, realizamos una evaluación del modelo mediante el método score, que calcula el rendimiento del modelo, dados por separado el conjunto de ejemplos de prueba y la clase de cada uno de estos ejemplos.

D. Redes Neuronales

Las Redes Neuronales son un modelo de predicción inspirado en el funcionamiento del cerebro, permitiendo que programas informáticos reconozcan patrones y resuelvan problemas comunes en el campo de la Inteligencia Artificial, Aprendizaje Automático y Aprendizaje Profundo. Son un conjunto de neuronas artificiales, cada una ellas con un sesgo y un peso asociado e interconectadas entre sí. Se agrupan en forma de capas, una de entrada donde se reciben los datos del problema, una o varias capas ocultas y una de salida para dar la predicción según los datos introducidos. Una neurona mandará información a otra neurona de la siguiente capa si la salida de esta es superior al valor del sesgo, si no, no transmitirá informacion alguna.

A la hora de entrenar el modelo se le asginarán unos valores aleatorios a los pesos de cada neurona, el peso se irá actualizando en función del siguiente algoritmo:

1 Inicializar los pesos aleatoriamente
2 **Repetir** hasta que se cumpla la condición de terminación:
3 **Para cada** ejemplo de entrenamiento
4 **Si** la salida del perceptrón es incorrecta entonces
5 Actualizar los pesos
6 **Devolver** los pesos

Cuando en el bucle del algoritmo se han considerado todos los ejemplos se dice que ha transcurrido una época.

En nuestro caso (fíjese en la Fig. 5), tenemos una capa de entrada de 3 neuronas, correspondientes a cada uno de los atributos usados para calcular el objetivo. Para la capa oculta

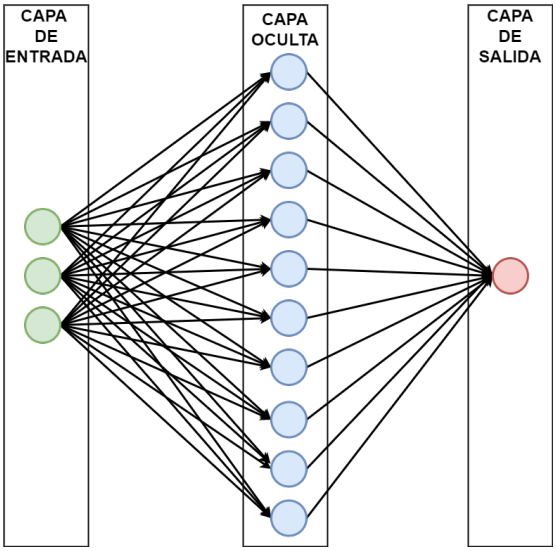


Fig. 5. Nuestra Red Neuronal

hemos optado por un total de 10 neuronas con función de activación rectificador (ReLU). Al tratarse de un problema de clasificación binaria, la última capa de nuestra red neuronal estará compuesta por una neurona con funcion de activación sigmoide, ya que los valores de salida oscilarán entre 0 y 1. Finalmente el algoritmo nos dará una tasa de aciertos, cada usuario será clasificado como Partner (1) si esta es mayor que 0.5.

Para el entrenamiento de la Red Neuronal usamos la entropía cruzada binaria como funcion de coste y el método de descenso por el gradiente (SGD, Stochastic Gradient Descent) para actualizar los pesos en la direccion $-\nabla C$.

· *Binary Crossentropy*:

$$-ylog_e(a^L) - (1-y)log_e(1-a^L) \tag{9}$$

Para la evaluación del modelo hemos dividido los datos en dos subconjuntos de prueba y entrenamiento en un 25% y un 75% respectivamente, aportándonos unos resultados con un rendimiento excelente.

V. Resultados

En esta sección se detallarán tanto los experimentos realizados como los resultados conseguidos. Para poder medir el rendimiento de los modelos mediante un solo elemento, realizamos la media de los resultados obtenidos en validación cruzada, y usamos dicha media para comparar con el resto de modelos.

A. KNN

El único contratiempo que nos encontramos a la hora de evaluar el modelo, fue que al repasar nuestro trabajo, observamos que al realizar el algoritmo KNN, a penas se tenía en cuenta el atributo mature con respecto a views, ya que toma valores mucho mas elevados, haciendo necesario una normalización de los atributos.

El mejor resultado obtenido al aplicar validación cruzada al modelo sin atributos relacionales es: 0.946, con el hiperparametro $k = 10$. Al añadir los atributos relacionales la mejor tasa de acierto sigue siendo para $k = 10$, aunque el valor aumenta a: 0.958.

Como ya se ha mencionado, para comparar los resultados se hace uso de la media de las tasa de aciertos. Sin atributos relacionales la media da : 0.934, mientras que con relacionales aumenta a : 0.955. Podemos concluir entonces, que el añadir los atributos relacionales mejoran el redimiento del modelo KNN.

B. Naive Bayes

En este caso, estabamos bastante familiarizados con la realización del modelo gracias a las clases prácticas. Lo único que tuvimos que investigar (meidante la documentación de sklearn) fue el proceso de discretización, como ya hemos mencionado anteriormente.

El mejor resultado obtenido al aplicar validación cruzada al modelo sin atributos relacionales es: 0.952, con el hiperparametro $k = 1$. Al añadir los atributos relacionales la mejor tasa de acierto sigue siendo para $k = 1$, aunque el valor aumenta a: 0.957.

Como ya se ha mencionado, para comparar los resultados se hace uso de la media de las tasa de aciertos. Sin atributos relacionales la media da : 0.949, mientras que con relacionales aumenta a : 0.955. Podemos concluir entonces, que el añadir los atributos relacionales mejoran el redimiento del modelo Naive Bayes.

C. Árbol de Decisión

Para árboles tenemos el método $score$ de sklearn que nos permite calcular el rendimiento del modelo mediante un solo valor, siendo el resultado devuelto por este método el que usaremos para la evaluación del modelo.

Sin atributos relacionales la tasa de acierto devuelta es: 0.981. Se puede observar que debido al peso de los otros atributos el algoritmo ignora el atributo mature. En cambio, al añadir los atributos relacionales el árbol cambia de forma drástica, pues ahora el algoritmo tiene en cuenta tanto los atributos relacionales como el atributo mature. No obstante esto empeora el rendimiento del modelo, pues la tasa de acierto baja: 0.980.

Podemos concluir entonces, que el añadir los atributos relacionales empeoran el redimiento del modelo Árbol de Decisión.

D. Redes Neuronales

A la hora de realizar este modelo no tuvimos problema en cuanto al código (no más que alguna revisión a la documentación de las librerías), gracias a la los ejemplos dados en clase práctica. Sin embargo, para el comprensión del algoritmo, las estructuras de las capas o la función de pérdida, tuvimos que apoyarnos en el temario de la asignatura.

Durante el entrenamiento de la red, la función de pérdida (binary cross-entropy) disminuye hasta: 0.631, y la tasa de acierto aumenta hasta: 0.947, al haber transcurrido 10 épocas. Evaluando con los atributos y objetivos de prueba, separados anteriormente, nos da un valor de la función de pérdida de: 0.353 y de tasa de acierto de: 0.942. Al añadir los atributos relacionales, y usando el mismo número de épocas, en el entrenamiento la función de pérdida disminuye hasta: 0.447 y la tasa de acierto aumenta hasta: 0.945. En la evaluación, con los atributos y objetivos de prueba la función de pérdida da una valor de: 0.438 y un valor para la tasa de acierto de: 0.946.

Como se puede observar, la tasa de acierto de la evaluación realizada aumenta ligeramente al añadir los atributos relacionales.

E. Comparación de los modelos

Hemos seleccionado por tanto el modelo de **Árboles de Decisión**, pues a pesar de que a añadir los atributos relacionales, baje la tasa de acierto, esta sigue siendo mayor que la obtenida con el resto de modelos que hemos probado. Es digno de mención que el modelo que más ha mejorado el rendimiento con la adicción de los atributos relacionales es KNN, pues la media de las tasas de acierto aumenta en un: 0.021.

VI. Conclusiones

Este trabajo nos ha resultado muy útil para repasar la teoría de los modelos de aprendizaje automático y aclarar conceptos clave de redes neuronales, al ser este un tema bastante más complejo que el resto de modelos. A medida que hemos ido realizando el trabajo, nuestro conocimiento del mismo ha mejorado, a base de construir y comprender los distintos modelos usados.

Si bien al principio nos abrumaba el uso y estructura de la herramienta LaTeX, a medida que le hemos dado uso nos hemos ido familiarizando con la herramienta, provocando así una mayor soltura al darle uso, o incluso resultando entretenido. Esto nos será de gran ayuda para trabajos futuros, ya sean en la carrera o fuera de ella.

También nos has sorprendido lo fácil que nos ha resultado volver a manejar python con facilidad, pues llevabamos sin usar python en la carrera desde el primer curso. Esto ha sido gracias a los apuntes que se nos proporciona en la página de la asignatura.

Para el control de versiones y trabajo simultáneo nos hemos apoyado en la herramienta github. Dicha herramienta nos ha facilitado la cooperación y comunicación, evitando así tener que hacer reuniones frecuentes, ahorrandonos tiempo de planificación.

En cuanto a las librerías usadas, nos han servido mucho para expandir nuestro conocimiento sobre librerías orientadas a la inteligencia artificial y dominar su uso. Iguialmente ha resultado muy útil para aprender y familiarizarnos con la lectura y comprensión la documentación de cualquier librería.

REFERENCIAS

[1] https://scikit-learn.org/stable/modules/preprocessing.html#normalization
[2] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.DistanceMetric.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.KBinsDiscretizer.html
[4] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
[5] https://scikit-learn.org/stable/modules/naive_bayes.html
[6] https://www.ibm.com/cloud/learn/neural-networks
[7] https://www.draw.io
[8] https://scikit-learn.org/stable/modules/tree.html
[9] https://machinelearningmastery.com/binary-classification-tutorial-with-the-keras-deep-learning-library/
[10] https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/
[11] Apuntes de la asignatura Inteligencia Artificial, Universidad de Sevilla (2021-2022)