MSEL
Modelling in Science Education and Learning

Modelling in Science Education and Learning
Volume 14(1), 2021 doi: https://doi.org/10.4995/msel.2021.15023.
Instituto Universitario de Matemática Pura y Aplicada
Universitat Politècnica de València

Redes neuronales en el fútbol
Neural networks in football

Llorenç Sancho Barrios, Nofre Sanmartín Vich, Carlos Roger de la Resurrección
Máster Universitario en Investigación Matemática
Universitat Politècnica de València–Universitat de València
llosanba@alumni.upv.es, osanvic@posgrado.upv.es, crogdel@posgrado.upv.es

Abstract

El aprendizaje automático brinda la capacidad de examinar conjuntos de datos masivos y descubrir patrones dentro de los datos sin depender de suposiciones a priori. Su aplicación al ámbito del deporte, que está experimentando un rápido crecimiento, se divide en modelos predictivos (programas de entrenamiento, resultados...) y explicativos (lesiones). En esta memoria, que forma parte de un proyecto final para una asignatura de máster, empleamos técnicas de aprendizaje no supervisado (mapas autoorganizados y clústering) para agrupar jugadores en función de diferentes estadísticas (pases, goles, faltas, etc) y comparamos los resultados con sus posiciones reales de juego. Asimismo, se describen las herramientas utilizadas para implementar y visualizar los resultados, con el objetivo de que un lector pueda inspirarse para realizar su propio proyecto.

Machine learning provides the ability to examine massive datasets and discover patterns within the data without relying on a priori assumptions. Its application to the field of sport (which is experiencing rapid growth) is divided into predictive (training programmes, results...) and explanatory (injuries) models. In this report, which is part of a final project for a master's degree course, we use unsupervised learning techniques (self-organised maps and clustering) to group players according to different statistics (passes, goals, fouls, etc.) and compare the results with their real playing positions. We also describe the tools used to implement and visualise the results, so that a reader can be inspired to carry out their own project.

Palabras clave: Mapas Autoorganizados, shiny, R, Fútbol, Inteligencia Artificial, Aprendizaje no supervisado
Keywords: Self-Organized Maps, shiny, R, Football, Artificial Intelligence, Unsupervised Learning

1. Introducción

El análisis de datos es una de las disciplinas de las matemáticas que más interés despierta hoy en día. Los últimos avances tecnológicos, sobre todo en computación, han permitido el procesamiento y almacenamientos de ingentes cantidades de datos, de los cuales puede extraerse información no accesible desde otros métodos. Las aplicaciones en medicina, astrofísica o incluso en ajedrez hacen de esta una herramienta sin la que no pueden entenderse muchos avances actuales, hasta el punto de la existencia de empresas cuyo producto es, precisamente, soluciones para el estudio de datos.

Por este motivo, en el Máster Universitario de Investigación en Matemáticas impartido conjuntamente por la Universitat Politècnica de València y la Universitat de València, se imparte una asignatura de una rama del análisis de datos llamada *Redes Neuronales y Algoritmos Genéticos*. El método de evaluación en esta asignatura consiste en un trabajo final en el que se debe realizar un estudio de datos aplicando algún tipo de red neuronal utilizando la librería *shiny* del *software R* para visualizar los resultados en una plataforma web interactiva.

La temática fue libremente elegida por nosotros, que decidimos que fuera sobre el deporte, más específicamente sobre el fútbol. Este deporte es muy popular en todo el mundo, por lo que despierta un gran interés y hay una gran cantidad de datos disponibles para realizar los análisis. Nuestro propósito es examinar diferentes estadísticas recogidas sobre jugadores de fútbol relativas al pase, aplicar un algoritmo de aprendizaje (en particular el SOM) y comparar los grupos obtenidos con las diferentes posiciones reales de los jugadores.

Los proyectos realizados para la asignatura de la que hemos hablado antes debían de que ser expuestos al resto de compañeros y profesores mediante un seminario y son parte de la evaluación. Las prácticas de la asignatura se han realizado con el lenguaje de programación *R* y para la elaboración del trabajo, se propuso a los alumnos hacer uso de un paquete llamado *shiny*, que permite crear una plataforma interactiva y enseñar los resultados. Esta herramienta resulta tremendamente útil para esta tarea, pero además el hecho de que sea reactiva (que sea capaz de actualizarse instantáneamente según unos parámetros que introduzca un usuario), permite realizar pruebas para calibrar los distintos parámetros con rapidez, facilidad y claridad.

No obstante, lo que sucedió hacia el final del curso alteró los planes: la pandemia del Covid-19 nos dejó confinados, haciendo imposible una exposición presencial, ni siquiera la asistencia al campus. El confinamiento nos impidió quedar para poder hacer el trabajo y tuvimos que elaborar el proyecto (e incluso este artículo) de manera telemática. Pese a las evidentes dificultades que nos trajo esta situación, fue una experiencia vital de lo más interesante, teniendo que hacer uso de nuestra creatividad para solventar problemas a los que no nos habíamos enfrentado antes. La realización del proyecto nos ocupó entre tres y cuatro semanas, quedando unos 4 días a la semana y compaginando el trabajo con el resto de tareas de las demás asignaturas.

En este escrito se pretende explicar el proyecto, con el objetivo de intentar servir de inspiración para la realización de un proyecto de características similares, o bien como actividad para proponer para una posible evaluación.

2. Fundamentos teóricos

2.1. Redes neuronales

Son muchos los conceptos matemáticos y algoritmos que están basados en procesos naturales. Este es el caso de las redes neuronales: son algoritmos basados en el proceso de transmisión de información de las neuronas en los seres vivos.

De forma muy simple, se podría decir que una neurona tiene tres tipos de componentes que intervienen principalmente en este proceso: las *dendritas*, el *soma* y el *axón*. Las dendritas reciben señales o impulsos eléctricos provenientes de otras neuronas, a través del espacio sináptico. La acción de transmisores químicos en las dendritas modifican estas señales entrantes, normalmente escalando la frecuencia de la señal entrante. El soma es el cuerpo de la neurona, y es donde se suman la totalidad de las señales entrantes. Si esta suma de señales consigue sobrepasar un cierto umbral la neurona se excita produciendo, así mismo, una señal de salida a través de los axones.

Una red neuronal artificial es un grupo interconectado de neuronas similar a la vasta red de neuronas en un cerebro biológico. Esta se caracteriza por una serie de conexiones (dendritas) que conectan unos núcleos (neuronas). Estas neuronas se organizan en capas, que son agrupaciones de varias neuronas, no necesariamente del mismo tamaño. En los casos más simples cada neurona recibe señales de las neuronas de las capas anteriores.[1]

La cantidad de capas y la forma en la que se conectan las neuronas de estas se conoce como *arquitectura* de la red. Un ejemplo de una red con una arquitectura simple se puede ver en la Figura 1.

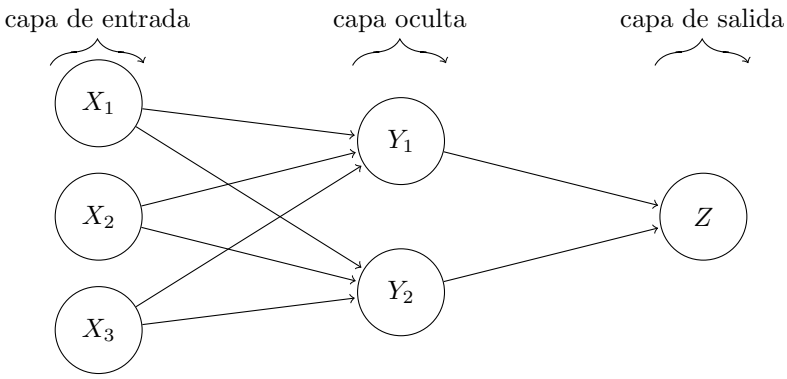


Figura 1 – Ejemplo de arquitectura de red con dos capas.

La primera capa, formada por las neuronas X_1, X_2, X_3 en la figura, se denomina *capa de entrada* y produce como señales de salida los valores de los datos introducidos en la red. La capas intermedias (la capa formada por los nodos Y_1, Y_2 en la figura) se conocen como *capas ocultas*, y el número de estas puede variar en cada red, pudiendo no haber ninguna. Finalmente, la última capa se denomina *capa de salida*.

Cada neurona tiene un estado interno, llamado *activación*, que viene representado por una función denominada normalmente *función de activación*. Habitualmente todas las neuronas de una misma capa tienen asociadas la misma función de activación. Algunos ejemplos de funciones

[1] En casos más complejos las neuronas pueden tener conexiones incluso con neuronas de su misma capa, pero este tipo de redes neuronales no se tratarán en este escrito, que se pretende solo introductorio.

de activación más utilizadas son la función **sigmoidea**, definida como

$$f(x) = \frac{1}{1+e^{-x}},$$

o la función de **Heaviside**, que se define como

$$f(x) = \begin{cases} 0 & \text{si} \quad x \leq \alpha \\ 1 & \text{si} \quad x > \alpha \end{cases}$$

donde α es un valor cualquiera.

Las conexiones entre las neuronas llevan asociadas unos pesos, que regulan las señales escalándolas, pudiendo incluso anularlas. Cuando cada una de las señales de entrada a una neurona ha sido escalada, estas se combinan formando un único valor de entrada. Por ejemplo, en la Figura 2, la neurona Y_1 recibe las señales entrantes x_1, x_2 y x_2 de las neuronas X_1, X_2 y X_3 respectivamente. Las conexiones entre estas neuronas llevan asociadas los pesos w_1, w_2 y w_3. Todos estos pesos multiplican (respectivamente) a las señales x_1, x_2, x_3 y formando un único valor entrante:

$$\omega = w_1x_1 + w_2x_2 + w_3x_3.$$

Dicho valor sirve como *input* para la función de activación de la neurona Y_1, obteniéndose la señal de salida de la neurona, $y_1 = f(\omega)$.

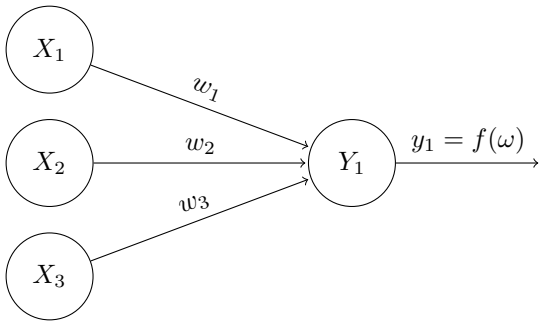


Figura 2 – Red neuronal con 3 señales de entrada y una sola neurona.

Además de la arquitectura de la red, otro aspecto que caracteriza a las redes neuronales es el *entrenamiento*, que afecta a cómo cambian los pesos de las conexiones en el proceso, es decir, cómo aprende la red neuronal. La modificación de los pesos se plantea como un problema de minimización de una función de pérdida sobre un conjunto de datos de entrenamiento. Esta modificación de pesos se produce en cada iteración, y se puede expresar como

$$w(t+1) = w(t) + \Delta w(t),$$

donde w sería el vector de las conexiones de una neurona con las neuronas de la capa anterior y $\Delta w(t)$ el cambio que se produce en la iteración $t+1$. $\Delta w(t)$ depende de un valor α, conocido como *la tasa de aprendizaje*, que es un parámetro que determina en cada iteración cuánto nos movemos en la dirección del gradiente de la función de pérdida. El valor de la tasa de aprendizaje determinará la convergencia de nuestra red: un valor demasiado alto producirá saltos sobre el mínimo buscado mientras que un valor demasiado bajo tardará mucho en converger. El valor de la tasa de aprendizaje es recomendable que esté en el intervalo $[0.01, 0.05]$. En algunos tipos de redes neuronales este valor es una variable decreciente en función de la iteración.

Existen diferentes tipos de aprendizaje o entrenamiento. En nuestro caso, se usará el *aprendizaje no supervisado*. Este es el modo de aprendizaje usado cuando no se dispone de un valor

de salida esperado de los datos de entrenamiento, y los algoritmos que aprenden de esta forma se conocen también como algoritmos *auto-organizados*. En este tipo de redes se modifican los pesos para que los grupos de datos más similares entre ellos den la misma salida en la red neuronal. Este es el caso de la red *SOM* ("Self-Organized Map"), de la que se hablará con más detalle en la próxima sección. Para más información sobre los distintos tipos de aprendizaje, véase Calabuig, García-Raffi y Sánchez-Pérez, 2021.

2.2. Self-Organized Maps: SOM

Los mapas autoorganizados (en inglés "Self-Organized Maps", SOM) son un tipo de red neuronal encuadrada dentro del aprendizaje de tipo no supervisado. Fue desarrollada por Teuvo Kohonen (*Helsinky University of Technology*), en 1982 (véase Kohonen, 1982). Este tipo de red neuronal está incluida en una clase más amplia de redes denominadas *redes competitivas*, donde las neuronas de salida de la red compiten entre sí para activarse, con el resultado de que solo una neurona de salida, o una neurona por grupo, está activada en un momento dado.

El desarrollo de los mapas autoorganizados como redes neuronales está motivado por un rasgo adicional del cerebro humano: el cerebro está organizado en distintas zonas mediante un orden topológico de grafo (véase Haykin, 1999 y Fausett, 1994), de manera que un impulso nervioso proveniente de los sentidos excita todas las neuronas de una vecindad de este grafo. Por ejemplo, entradas sensoriales provenientes del tacto, vista y oído son aplicadas a diferentes áreas del córtex cerebral, ordenada con esta topología particular (véase Haykin, 1999).

Una red Neuronal SOM está formada por un espacio de entrada con m vectores de $\mathbb{R}^n$, donde m es el número de entradas y n la información que disponemos de cada entrada; y una capa de neuronas de salida. Estas neuronas, siguiendo el proceso biológico, están ordenadas topológicamente en vecindades, que pueden representarse mediante un grafo (G, τ), donde G es el conjunto de las neuronas y τ es un conjunto de aristas que caracterizan el grafo. Un ejemplo de topología en las neuronas puede verse en la Figura 3. Notemos que el número de neuronas, m, es independiente del espacio $\mathbb{R}^n$.

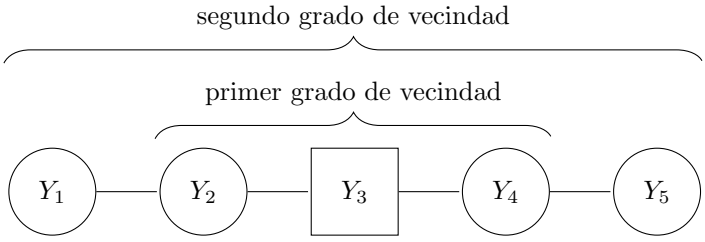


Figura 3 – Ejemplo de topología de una dimensión para las neuronas de salida. Las neuronas que tiene un grado de vecindad 1 con Y_3 son Y_2 y Y_4, mientras que las neuronas que tienen un grado de vecindad 2 son Y_1 y Y_5.

El objetivo es agrupar los vectores de entradas de manera que los elementos de cada agrupación sean lo más similares posible. Se asocia a cada neurona $Y_i \in G$ con un vector $w_i \in \mathbb{R}^n$ que llamaremos centro de la neurona. Cada neurona representará una agrupación. El proceso de entrenamiento consiste en mover los centros de las neuronas de manera que cada neurona represente una agrupación real del conjunto de entrada. La topología del grafo es utilizada para que agrupaciones dadas por dos elementos del grafo vecinos sean más similares a dos agrupaciones no vecinas. El número de neuronas es finito así que podemos suponer que $|G| = p \in \mathbb{N}$. De esta manera, llamaremos a las neuronas Y_i con $1 \leq i \leq p$.

Sea Y_i una neurona de G con $1 \leq i \leq p$. El vector $w_i = (w_{1,i}, \ldots, w_{n,i}) \in \mathbb{R}^n$ es el vector de pesos asociado a la neurona Y_i, es decir, para cada $1 \leq j \leq n$, el elemento $w_{j,i}$ es el peso

asociado a la conexión entre la entrada X_j y la neurona Y_i. De esta manera, la señal de entrada de la neurona Y_i es $\omega_i = \sum_{k=1}^{n} w_{k,i}x_k$. A partir de ese valor, se define el **potencial sináptico** de la neurona Y_i como:

$$h_i = \omega_i - \frac{1}{2}\sum_{k=1}^{n} w_{k,i}^2.$$

Entonces la función de activación se define como:

$$f_i(\omega) = \begin{cases} 1 & \text{si} \quad h_i = \max\{h_j \,|\, 1 \leq j \leq p\} \\ 0 & \text{otro caso} \end{cases}$$

Observemos que la arquitectura de la red consta de p neuronas en una sola capa y por tanto no tiene capas ocultas.

Como se ha comentado antes, este tipo de redes se llaman competitivas, porque cada vector de entrada solo "excita" una neurona de salida, que suele llamarse la *neurona ganadora*. Las neuronas $Y_i \in G$ competirán entre ellas comparando la distancia (normalmente euclídea) del vector de entrada con cada vector centro, es decir, se escoge el índice i de entre todas las neuronas de G que cumple que $d(X, w_i)$ es mínima, siendo $d(x, y) = \sqrt{\sum_{j=1}^{n}(x_j - y_j)^2}$. De esta forma cada entrada X activa una sola neurona Y_j. La neurona ganadora obtenida por la función $i(X)$ es precisamente la que mayor potencial sináptico h_i tiene, por lo que ambos enfoques son equivalentes.

Teorema 1. Si Y_i es la neurona con mayor potencial sináptico h_i para un patrón de entrada X dado, entonces Y_i es también la neurona ganadora.

Demostración: Para demostrar el resultado, hay que comprobar que la distancia del vector de pesos w_i al patrón de entrada X es la menor entre las distancias de todos los vectores de pesos, es decir, que

$$d(X, w_i) \leq d(X, w_k)$$

para cada índice k. Pero esto se da, ya que

$$d(X, w_i)^2 = \|X - w_i\|^2 = \langle X - w_i, X - w_i\rangle.$$

Como el producto escalar es una forma bilineal, se obtiene que

$$\langle X - w_i, X - w_i\rangle = \langle X, X\rangle + \langle w_i, w_i\rangle - 2\langle X, w_i\rangle = \langle X, X\rangle - 2h_i.$$

Ahora, como estamos suponiendo que la neurona Y_i es la que mayor potencial sináptico tiene, i.e., $h_k \leq h_i$ para cualquier índice k, se sigue que

$$\langle X, X\rangle - 2h_i \leq \langle X, X\rangle - 2h_k = d(X, w_k)^2,$$

de donde se obtiene el resultado. □

Una vez obtenida una neurona ganadora, empieza la parte de entrenamiento de la red, es decir, la modificación de los pesos. Como se ha introducido en la primera sección, esta modificación se puede expresar como

$$w_i(t+1) = w_i(t) + \Delta w_i(t),$$

siendo $w_i(t)$ el vector de pesos asociado a la neurona Y_i en la iteración t. Estos cambios en los pesos imitan las propiedades del cerebro humano comentadas antes: excitan una vecindad del

grafo G alrededor de la neurona ganadora Y_{i_0}. En la primera iteración se considera un grado de vecindad alrededor de la neurona ganadora, que se denomina amplitud. Una característica propia del SOM es que esta amplitud va descendiendo conforme aumenta el número de iteraciones t, por lo que se puede expresar dicha amplitud como una función $\sigma(t)$ dependiente de la iteración, siendo $\sigma_0 = \sigma(0)$ la amplitud inicial. Puesto que es deseable que la neurona ganadora excite a su vez las neuronas vecinas con una intensidad proporcional a su grado de vecindad, se define una **función vecindad** $h : \{1, \ldots, p\} \times \{1, \ldots, p\} \times \mathbb{N} \longrightarrow \mathbb{R}$, como $h(i, k, t)$, donde $i = i(X)$ es el índice de la neurona ganadora para la entrada X, k es el índice de otra neurona cualquiera $Y_k \in G$ y t es la iteración actual. Esta función h ha de cumplir que

(I) Tenga un máximo absoluto si $i = j$.

(II) Sea monótona decreciente en función del grado de vecindad entre Y_i e Y_j.

(III) Sea monótona decreciente en función de la iteración t.

(IV) Sea simétrica respecto los grados de vecindad.

h modula el cambio de los pesos según este parentesco del que se hablaba antes. Un ejemplo de función vecindad muy utilizado es la *función Gaussiana*, definida como

$$h(i, k, t) = \exp\left(-\frac{d_{i,k}^2}{2\sigma(t)^2}\right),$$

donde $d_{i,k}$ es un entero no negativo que indica el grado de vecindad entre la neurona ganadora de índice i y la neurona k (por ejemplo, $d_{i,k} = 1$ indicaría que las neuronas están unidas por una arista en el grafo, $d_{i,k} = 2$ indicaría ser vecinos de vecinos, y $d_{i,j} = 0$ que son la misma neurona) y $\sigma(t)$ es la amplitud de la vecindad, que sería una función positiva y monótona decreciente.

De esta forma, el cambio $\Delta w_i(t)$ se escribe como

$$\Delta w_i(t) = \alpha(t) h(i, k, t)(X - w_i),$$

siendo $\alpha(t)$ la tasa de aprendizaje presentada en la primera sección.

Este proceso se repite con el siguiente vector de entrada. Una vez entrenada la red con todos los vectores de entrada, se vuelve a entrenar un número determinado de veces definido a priori.

3. Planteamiento del trabajo

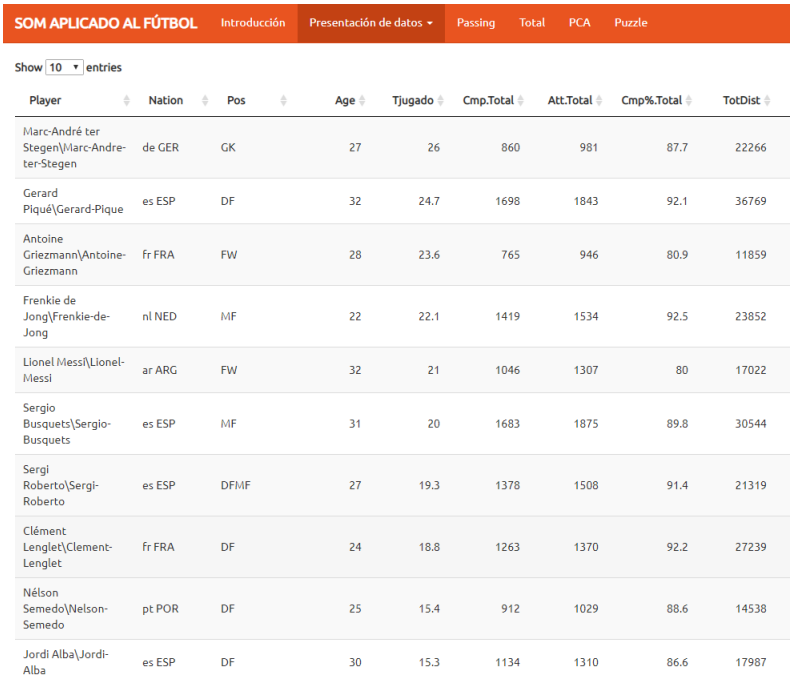
Nuestro trabajo, como hemos comentado previamente, forma parte del proyecto final de una asignatura del máster. En el ámbito del análisis de datos hay tres aspectos clave a tener en cuenta: las hipótesis, es decir, las preguntas científicas que queremos contestar; los datos disponibles; y la audiencia que verá nuestro análisis final. En nuestro caso, buscamos hacer un análisis descriptivo, puesto que nos planteamos si existe una correlación entre las estadísticas de pase de un jugador de fútbol y la posición en la que juega.

Nuestro propósito es por tanto examinar diferentes estadísticas recogidas sobre jugadores de fútbol relativas al pase, aplicar un algoritmo de aprendizaje supervisado (SOM) y comparar los grupos obtenidos con el SOM con las diferentes posiciones reales de los jugadores. Una vez hemos establecido nuestros objetivos, pasamos a la etapa de tratamiento de datos.

3.1. Obtención de los datos

Es posible que la primera tarea a la que se tiene que hacer frente al realizar un análisis de datos (o al menos es el caso de los autores de este escrito) sea la de buscar una fuente de datos con la que poder trabajar. Para este trabajo, los datos se obtuvieron de la página web fbref.com, a la que agradecemos enormemente su contribución.

Los datos descargados consistieron en una serie de marcadores relativos a los pases para cada equipo de las 5 grandes ligas europeas: *La Liga, Premier, Serie A, Bundesliga y Ligue 1*. Ejemplos de esos marcadores son el número de pases cortos, largos; así como métricas más sofisticadas como asistencias esperadas. Más información y descripciones sobre estos marcadores pueden encontrarse en la propia página. También se descargaron datos no usados posteriormente en el propio algoritmo, como nombre de jugador, edad o equipo de pertenencia, pero que fueron de ayuda para la visualización de los resultados. Un ejemplo de la forma que tenían estos datos, para el caso del equipo F.C. Barcelona puede verse en la Figura 4.


SOM APLICADO AL FÚTBOL Introducción Presentación de datos Passing Total PCA Puzzle

Show 10 entries

Player	Nation	Pos	Age	Tjugado	Cmp.Total	Att.Total	Cmp%.Total	TotDist
Marc-André ter Stegen\Marc-Andre-ter-Stegen	de GER	GK	27	26	860	981	87.7	22266
Gerard Piqué\Gerard-Pique	es ESP	DF	32	24.7	1698	1843	92.1	36769
Antoine Griezmann\Antoine-Griezmann	fr FRA	FW	28	23.6	765	946	80.9	11859
Frenkie de Jong\Frenkie-de-Jong	nl NED	MF	22	22.1	1419	1534	92.5	23852
Lionel Messi\Lionel-Messi	ar ARG	FW	32	21	1046	1307	80	17022
Sergio Busquets\Sergio-Busquets	es ESP	MF	31	20	1683	1875	89.8	30544
Sergi Roberto\Sergi-Roberto	es ESP	DFMF	27	19.3	1378	1508	91.4	21319
Clément Lenglet\Clement-Lenglet	fr FRA	DF	24	18.8	1263	1370	92.2	27239
Nélson Semedo\Nelson-Semedo	pt POR	DF	25	15.4	912	1029	88.6	14538
Jordi Alba\Jordi-Alba	es ESP	DF	30	15.3	1134	1310	86.6	17987

Figura 4 – Ejemplo de algunos de los datos descargados del equipo F.C. Barcelona.

En un primer momento, los datos descargados no tenían un formato tal que fuera posible trabajar con ellos. El principal problema que nos encontramos fue el de las entradas vacías (N/A, del inglés *not avaiable*, para entradas deliberadamente vacías). Este problema era especialmente evidente en el caso de los porteros, donde en algunas métricas más ofensivas tenían N/A. Este hecho nos impedía eliminar las entradas con el valor N/A en algún campo porque nos hubiésemos quedado sin porteros. Para solucionar este problema convertimos todos los N/A en 0. Nos aseguramos que este cambio era consistente con nuestra base de datos.

Otros problemas, de naturaleza más sensible, aparecieron una vez utilizado el algoritmo y examinados los resultados. Por ejemplo, la mayoría de entradas de nuestra base de datos hacían referencia al total de la temporada. Para que el tiempo de juego no sesgara el algoritmo decidimos transformar estas entradas dividiendo por los partidos jugados del jugador. Por ejemplo, si un jugador ha dado 10 asistencias y ha jugado 15 partidos, en la entrada relativa a las asistencias, tendremos $10/15 = 0.66$ asistencias por cada 90 minutos. Sin embargo, esto nos

condujo a un segundo problema. Algunos jugadores con pocos minutos jugados daban valores extremos en entradas como por ejemplo las asistencias. Por ello, decidimos eliminar de la base de datos los jugadores que habían jugado menos de 2 partidos ya que podrían contaminar los resultados del algoritmo.

Un problema anecdótico que tuvimos fueron los nombres de algunos jugadores de países de origen con alfabeto cirílico. R no era capaz de leer de los archivos .csv los nombres con alguna letra cirílica. Nuestra solución fue cambiar la letra de los nombres de estos jugadores por una letra similar del alfabeto latino.

3.2. Implementación del SOM y clústers

El lenguaje de programación utilizado es R, un software libre y ampliamente manejado en análisis de datos, cuyas librerías pueden descargarse en r-project.org. El tratamiento de los datos comentado antes se hizo usando este software. El paquete principal de R que nos permite aplicar el mapa autoorganizado es el paquete de Kohonen. Para más información, se puede consultar cran.r-project.org/package=kohonen.

La topología que elegimos para las neuronas de salida fue hexagonal, como puede verse en la Figura 5. En cuanto a las dimensiones del mallado, experimentando con las diferentes configuraciones, llegamos a la conclusión de que una distribución de 4 filas y 5 columnas nos daba unos resultados bastante satisfactorios. Si adoptábamos dimensiones muy pequeñas para el mallado, como era de esperar, el algoritmo no separaba los datos convenientemente y hacía agrupaciones exageradamente abultadas. En cambio, si las dimensiones del mallado eran demasiado grandes, el algoritmo acentuaba las diferencias entre los jugadores de forma excesiva, debido a que tenía demasiadas celdas que rellenar.

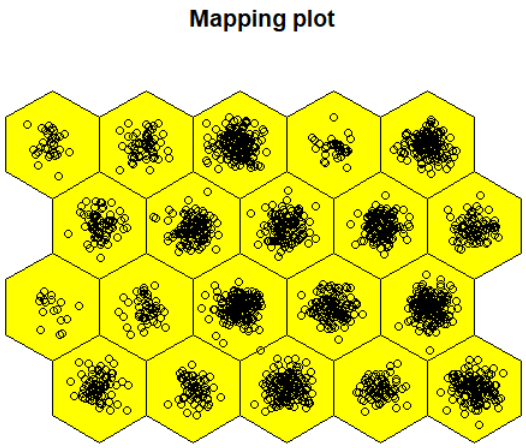


Figura 5 – Ejemplo topología de red hexagonal de 5 filas y 4 columnas.

Otro aspecto a tener en cuenta es el número de iteraciones. Es obvio que con un número bajo de iteraciones se obtiene un resultado pobre, ya que el algoritmo se entrena poco y por tanto se ajusta poco a los datos de entrenamiento. No obstante, a mayor número de iteraciones aumenta el coste computacional. Se hicieron distintas pruebas en un rango de 500 a 1500 iteraciones, gracias a la flexibilidad que nos aportó la plataforma Shiny, de la que hablaremos más adelante.

Respecto a la tasa de aprendizaje, se utilizó la que venía por defecto en la función *som()* del paquete Kohonen. Esta tasa se puede expresar como

$$\alpha(t) = 0.05 * \left(1 - \frac{t}{T_{max}}\right) + \frac{t}{T_{max}} * 0.01,$$

donde T_{max} es el número máximo de iteraciones que se realizarán. Notemos que es una función lineal decreciente que en la primera iteración ($t = 0$) vale 0.05 y en la última tiene un valor de 0.01.

La vecindad del algoritmo, al igual que la tasa de aprendizaje, se dejó por defecto y es también una función lineal decreciente. Si no se especifica otra cosa, la función *som()* calcula un valor de vecindad de forma que en la primera iteración se activan la neuronas con un grado de vecindad de hasta 2/3 del máximo grado de vecindad, mientras que en las última iteraciones solamente es activada la neurona ganadora.

La función *som()* acepta dos funciones $h(i, k, t)$ para los cambios de los pesos en el proceso de entrenamiento. Estas son la función Gaussiana y otra función llamada *Bubbles*, que se define como:

$$f(d_{i,j}, r) = \begin{cases} 1 & si \quad d_{i,j} \leq r \\ 0 & si \quad d_{i,j} > r \end{cases}$$

donde $d_{i,j}$ es la distancia desde una neurona Y_j a la neurona ganadora Y_i, y r es la amplitud. En el caso del trabajo presentado aquí, se utilizó esta última función para entrenar la red.

Estas y otras características sobre el paquete *Kohonen* se encontraron en el código fuente, que los autores del mismo (Wehrens, R. y Kruisselbrink, J.) han puesto disponible en el repositorio de Github. Este código puede encontrarse en https://github.com/cran/kohonen.

Con la topología que hemos seleccionado para el SOM, y usando unas dimensiones de 4 filas y 5 columnas, tenemos 20 agrupaciones donde se clasifican todos los jugadores. Un ejemplo de clasificación puede verse en la Figura 5.

A pesar de tener ya una primera clasificación de los datos, estos resultados todavía no son de gran utilidad. Idealmente nos gustaría tener entre 5 y 11 agrupaciones, ya que en un equipo de fútbol hay 11 posiciones pero hay algunas muy parecidas. Para ello podríamos haber cambiado el tamaño del grafo reduciendo las neuronas pero en este proceso perdemos potencia a la hora de entrenar la red, como hemos comentado anteriormente. Por este motivo, hemos optado por aplicar un algoritmo de clúster para agrupar los resultados obtenidos con el SOM. Por el tipo de problema que estamos tratando, optamos por usar el método *Ward*.

La implementación del algoritmo clúster en R se hizo utilizando la función *hclust()*. Más información sobre esta librería puede encontrarse en:

https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/hclust.

Una vez usada la función *hclust()*, obtenemos un dendograma de árbol donde podemos visualizar como se agruparían las neuronas en función de un número de clústers deseado. Por ejemplo, en la Figura 6 se puede ver el orden en el que se han creado las agrupaciones de neuronas. Gracias a este gráfico, podemos visualizar como serían las agrupaciones en función del número de clústers. En el caso de la figura, se han elegido tres clústers, aunque otro número de clústers podría haber sido posible. En la Figura 7 puede verse como se traduce esta elección en la clasificación hecha con el SOM. Vemos que en cada agrupación no hay ninguna neurona que no tenga vecinas del mismo clúster.

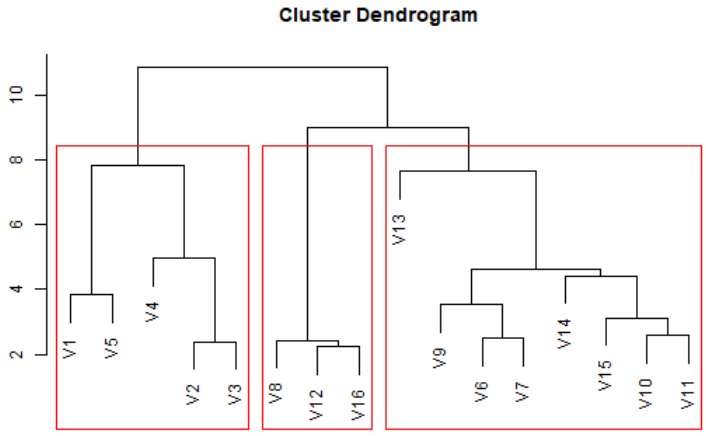


Figura 6 – Dendograma con las agrupaciones del clústering aplicado a las neuronas. Cada Vn representa una neurona.

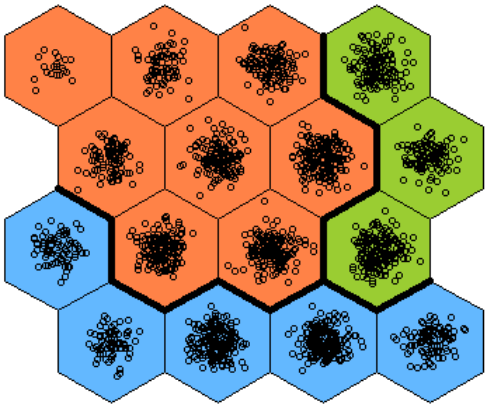


Figura 7 – Representación final de la clasificación, después de usar el SOM y el algoritmo clúster.

3.3. Presentación del trabajo

Como ya se ha comentado a lo largo del trabajo la librería que hemos empleado es *shiny*, un paquete de R destinado al desarrollo de aplicaciones web interactivas. La ventaja de este paquete es su reactividad, que permite visualizar diferentes escenarios de análisis en cuestión de segundos con un simple clic. Al desarrollar la aplicación, es posible implementar distintas opciones para que un usuario que haga uso de la misma elija con qué parámetros quiere realizar el algoritmo. Esta característica aporta una versatilidad enorme, no solo a la hora de exponer los resultados, sino también al realizar la investigación sobre los datos ya que es posible realizar pruebas con distintos parámetros y ver los resultados al momento.

A continuación explicaremos como hicimos nuestra plataforma, y como fuimos exponiendo nuestro trabajo el día de la presentación.

Nuestra plataforma está formada por una serie de pestañas, en cada una de las cuales se trabaja con diferentes bases de datos. Se ha implementado una serie de algoritmos para clasificarlos y se muestran unos gráficos para visualizar los resultados. En la primera pestaña se hace la presentación y se resume lo que se va a hacer en el resto de la plataforma. Esta puede verse en la Figura 8.

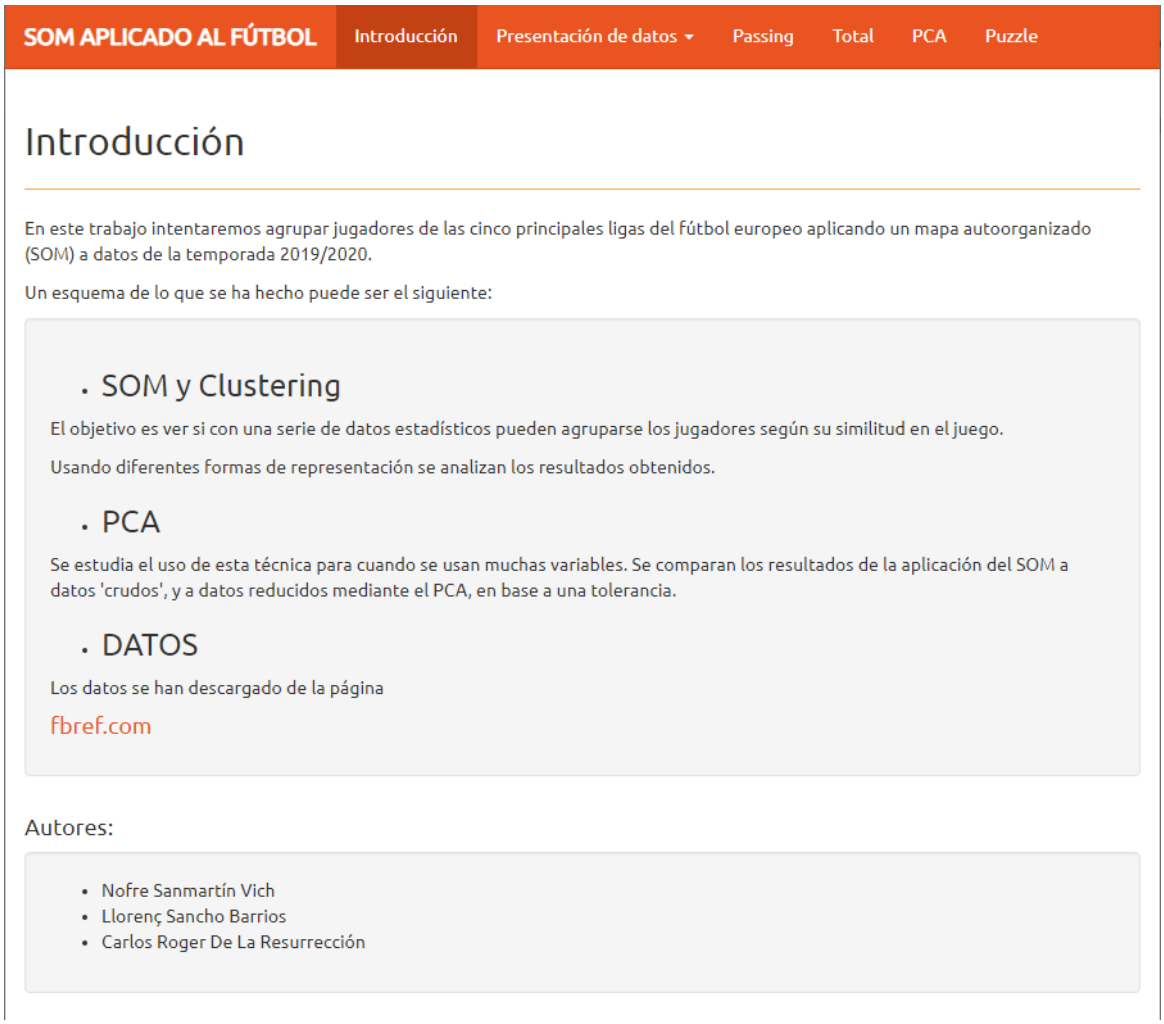


Figura 8 – Pestaña de presentación de la plataforma.

Dentro de la pestaña *Passing*, en la pestaña *SOM* realizamos el algoritmo SOM. Esto puede verse en la Figura 9. Podemos seleccionar el número de iteraciones y la dimensión de la topología deseados con dos deslizadores. Como salidas podemos ver tres gráficos. En el primer gráfico vemos el centro o los pesos de cada neurona, que nos sirve para saber cuáles son las características comunes dentro de cada neurona. En el segundo gráfico vemos la distancia de cada neurona a sus vecinas. Colores claros indican que los jugadores incluidos en una neurona están alejados de los jugadores de las neuronas vecinas en la topología. En este gráfico podemos ver que la distancia entre las diferentes neuronas no es constantes. Aunque las dibujemos todas equidistantes en el grafo, realmente vemos que en $\mathbb{R}^n$ hay algunas que cuyos centros están más cercas que otras. Para finalizar, vemos el dendograma que nos permite hacernos una idea de cuál podría ser el número de agrupaciones idóneo.

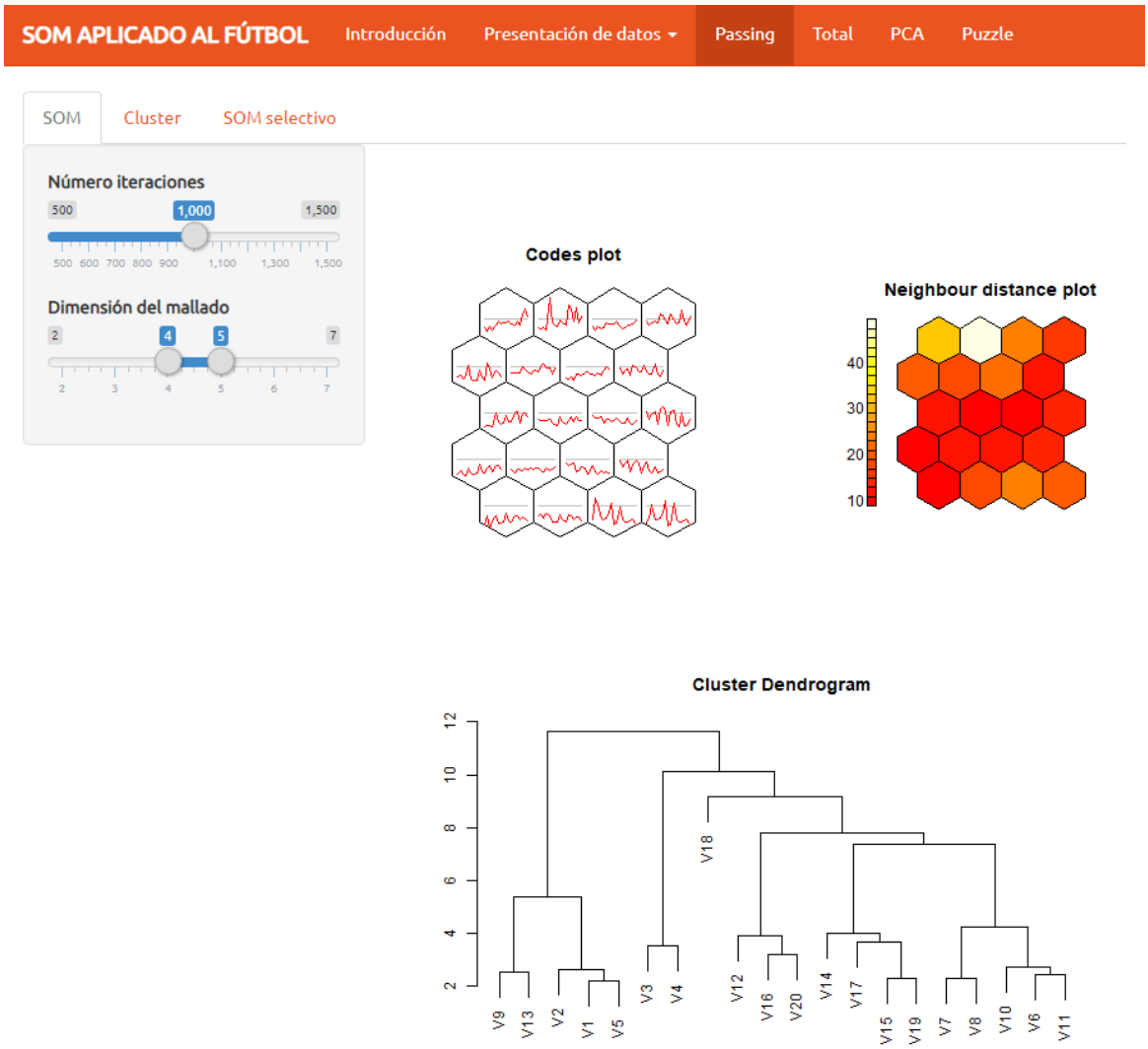


Figura 9 – En la pestaña Passing, la subpestaña *SOM* realiza la red neuronal y muestra diversos gráficos con los resultados.

En la pestaña *Cluster* realizamos el algoritmo de clustering. De nuevo se puede elegir el número de clústers que deseamos agrupar y por pantalla se muestra el dendograma, uniendo con un rectángulo rojo las neuronas de un mismo clúster. También mostramos por pantalla dos gráficos más, que no son más que la representación de los agrupamientos del clúster en la topología de la red. Además de las gráficas, en la aplicación se encuentra un selector que permite visualizar los jugadores de un grupo deseado en una tabla. Para finalizar, hay dos gráficos que permiten comparar los grupos en función de una métrica concreta que el usuario puede seleccionar a su gusto, como se puede ver en la Figura 11.

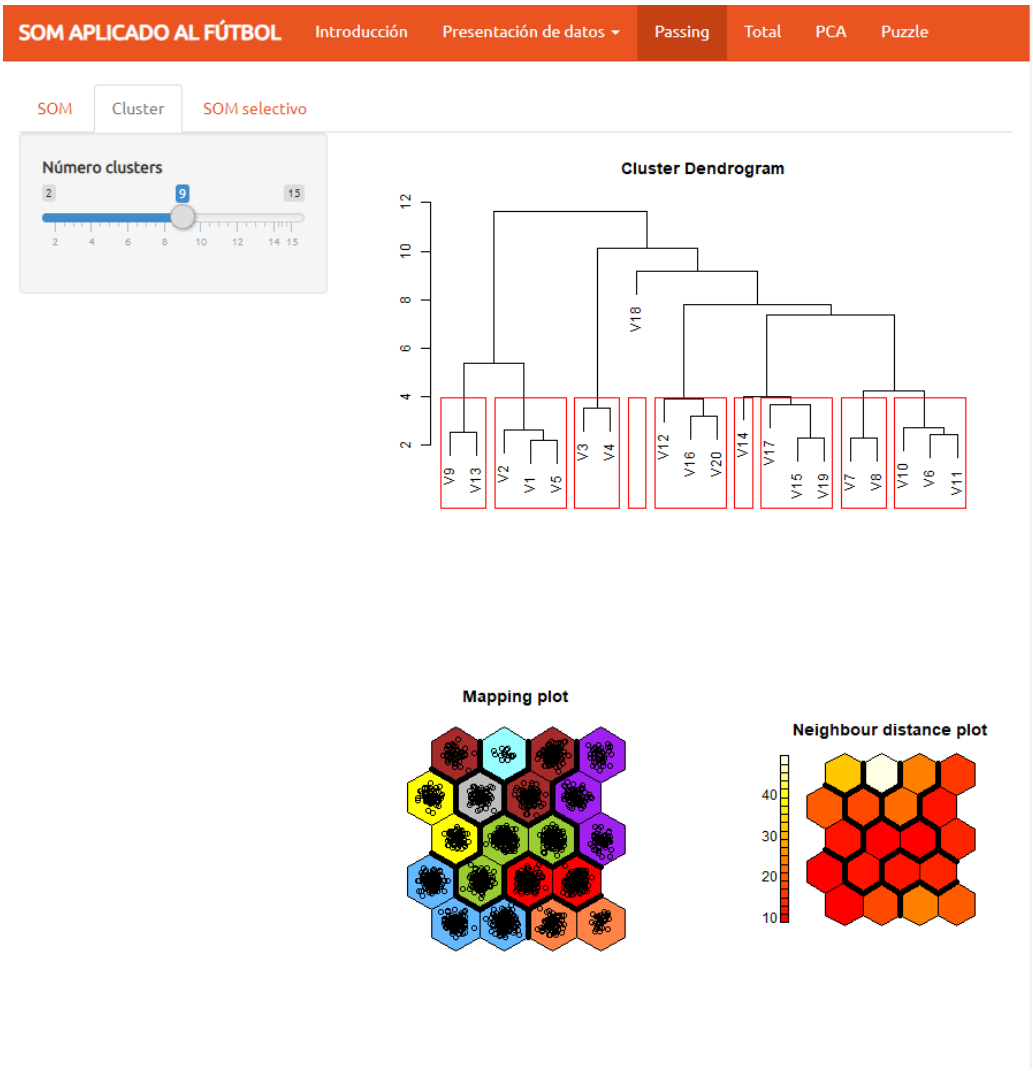


Figura 10 – Pestaña Passing, Cluster.

Otra funcionalidad de la pestaña *SOM* se encuentra en la pestaña *SOM selectivo*. En ella, se puede realizar otra vez el algoritmo del SOM sobre algunos grupos obtenidos en las pestañas anteriores, pudiendo elegir el grupo o grupos en que nos interese repetir el proceso. Esto nos permite separar grupos heterogéneos que, con todos los jugadores, el algoritmo no es capaz de separar.

Una vez obtenidas las primeras agrupaciones planteamos que pasaría si introdujésemos todos los datos disponibles para agrupar los jugadores. El resultado se encuentra en la pestaña *Total* donde se realiza el mismo proceso que en la pestaña anterior pero añadiendo los datos de acciones defensivas, tiros y posesión a los datos de pase.

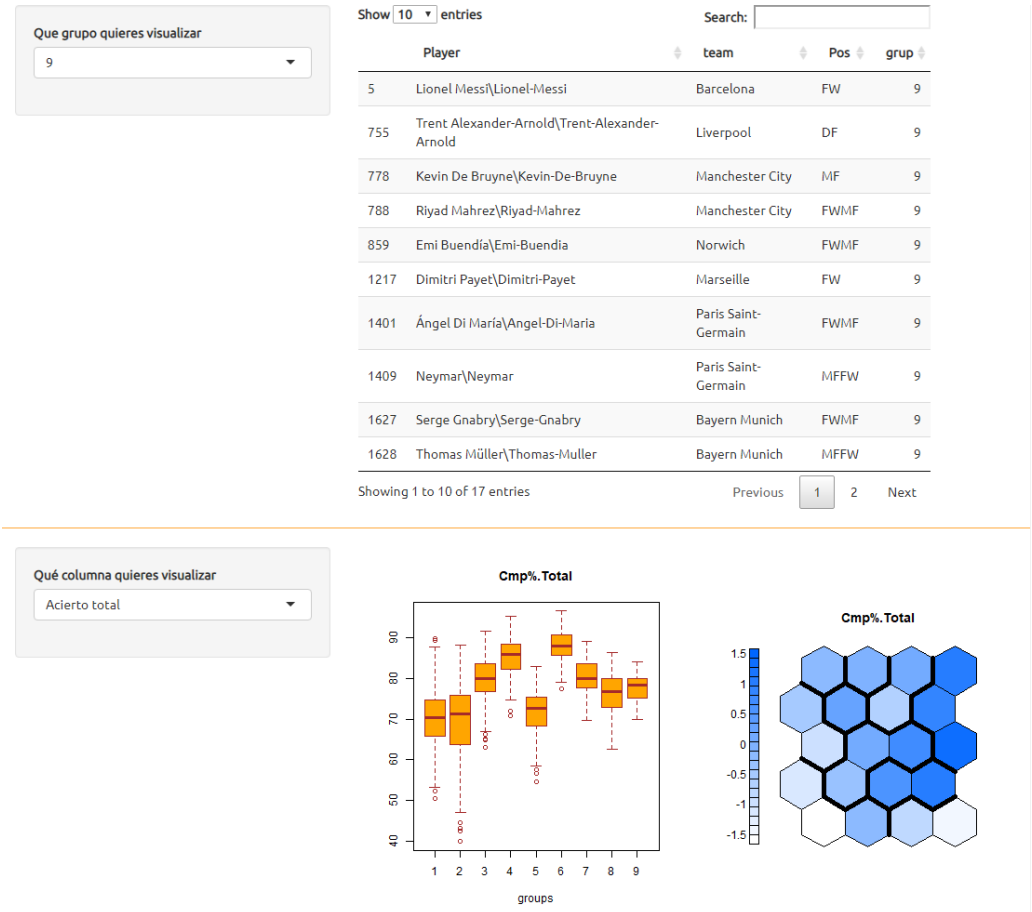


Figura 11 – Pestaña Passing, Cluster.

Llegados a este punto, estábamos trabajando con demasiada información y nos planteamos si sería posible aplicar un método de reducción de dimensión que fuese compatible con las agrupaciones. Por este motivo aplicamos el método de *PCA* (Análisis de Componentes Principales, del inglés *Principal Components Analysis*). El objetivo del PCA es transformar un conjunto de vectores de n componentes en un conjunto de vectores de m componentes con $m \leq n$. Es decir, proyectar los datos iniciales en un espacio de dimensión menor, de forma que se pierda la menor información posible. Hay varias formas equivalentes de deducir matemáticamente las componentes principales. La más simple es encontrar las proyecciones que maximizan la varianza: la primera componente principal es la dirección del espacio de partida tal que la proyección de los vectores de entrada sobre ella tiene la mayor varianza. La segunda componente principal es la dirección que maximiza la varianza (de la proyección de los vectores de entrada sobre ella) entre todas las direcciones ortogonales a la primera componente principal. La i-ésima componente es la dirección que maximiza la varianza entre todas las direcciones ortogonales a las primeras $(i-1)$ componentes. De esta forma, las componentes principales de una colección de vectores constituyen una base ortonormal en la que las diferentes dimensiones individuales de los datos no están correlacionadas linealmente. El PCA se usa comúnmente para reducir la dimensionalidad de los datos, ya que son las direcciones que más respetan la estructura de los datos (varianza) a la hora de proyectarlos sobre un subespacio de menor dimensión. En particular, por la construcción descrita, son las primeras componentes principales aquellas que contienen la mayor información sobre los datos de entrada. Se puede calcular el porcentaje que aporta cada dirección principal a la varianza total, y así elegir el grado de error que se comete al reducir dimensiones, que se conocería como *tolerancia*.

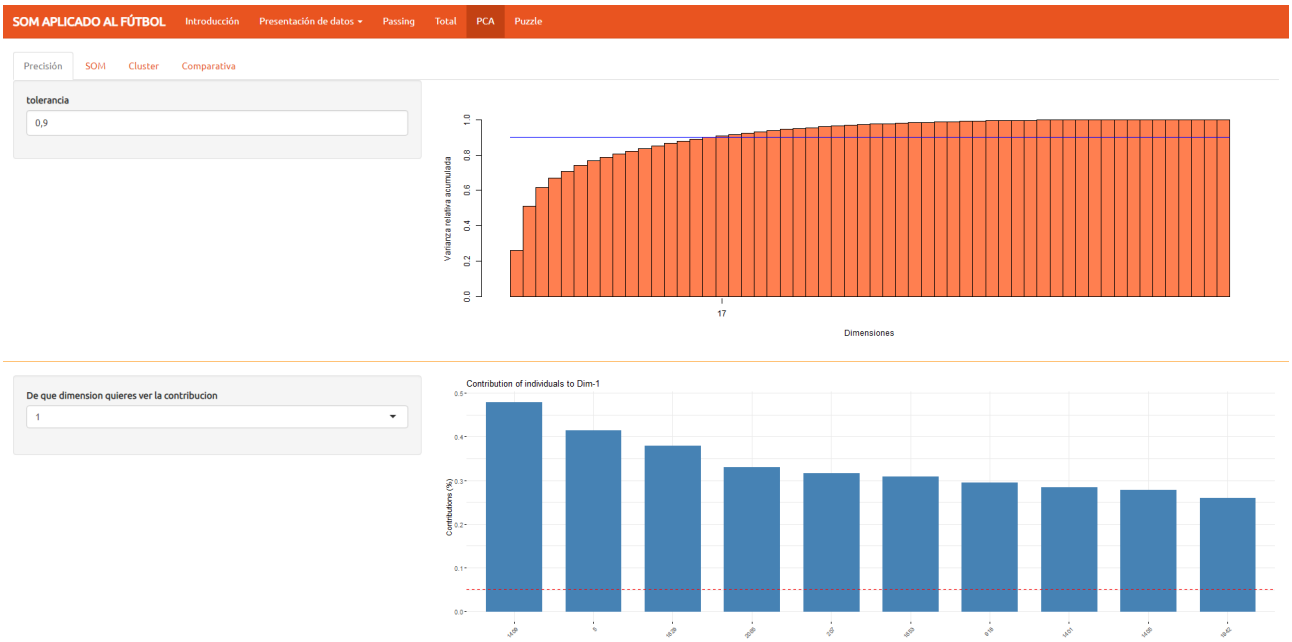


Figura 12 – Pestaña PCA

El uso del PCA se implementó en la plataforma en la pestaña *PCA*. Inicialmente, la plataforma nos da la opción de elegir la tolerancia, que nos indica cuánta información estamos dispuestos a perder a cambio de reducir nuestros datos. Puede verse en la Figura 12. La gráfica en azul nos da información sobre la aportación de cada variable original a una de las direcciones principales, a elegir por el usuario.

Seguidamente, se repite el proceso de aplicar el SOM y el algoritmo clúster, tal y como se hizo en las pestañas *passing* y *total*, pero esta vez sobre los datos proporcionados del PCA. La implementación se hizo de forma idéntica, con la excepción de una pestaña adicional en la que poder comparar las agrupaciones dadas aplicando el algoritmo sobre todos los datos o sobre los datos reducidos por el PCA. Dicha pestaña puede verse en la Figura 13.

Para más información sobre el paquete *Shiny* se puede consultar la dirección shiny.rstudio.com, donde además de documentación puede encontrarse plantillas, estilos y otras cosas de utilidad.

3.4. Resultados

Los resultados fueron, a nuestro modo de ver, satisfactorios. Nuestra idea inicial era que la clasificación obtenida coincidiera con la de las posiciones de los jugadores, ya que los jugadores en cada posición, suponíamos, deberían jugar de forma similar. Esto se obtuvo en los casos de las posiciones más singulares, como porteros, defensas o delanteros centros, donde la mayoría de los jugadores eran clasificados en un mismo grupo. En posiciones más flexibles, como los laterales o centrocampistas, que pueden tener papeles defensivos y ofensivos simultáneamente, se separa en varios grupos. Podemos encontrar un grupo de pasadores con centrocampistas como Busquets, Kroos y Modric pero a la vez laterales como Jordi Alba o Carvajal. Luego tenemos dos grupos: uno de centrocampistas y laterales más defensivo y otro más ofensivo. La aparición de estos grupos mixtos, unos con más sentido desde nuestro conocimiento del deporte, otros con menos, fue lo que nos animó a implementar otra pestaña con más datos para hacer una segunda clasificación de los jugadores. Las conclusiones que extraemos de esta segunda

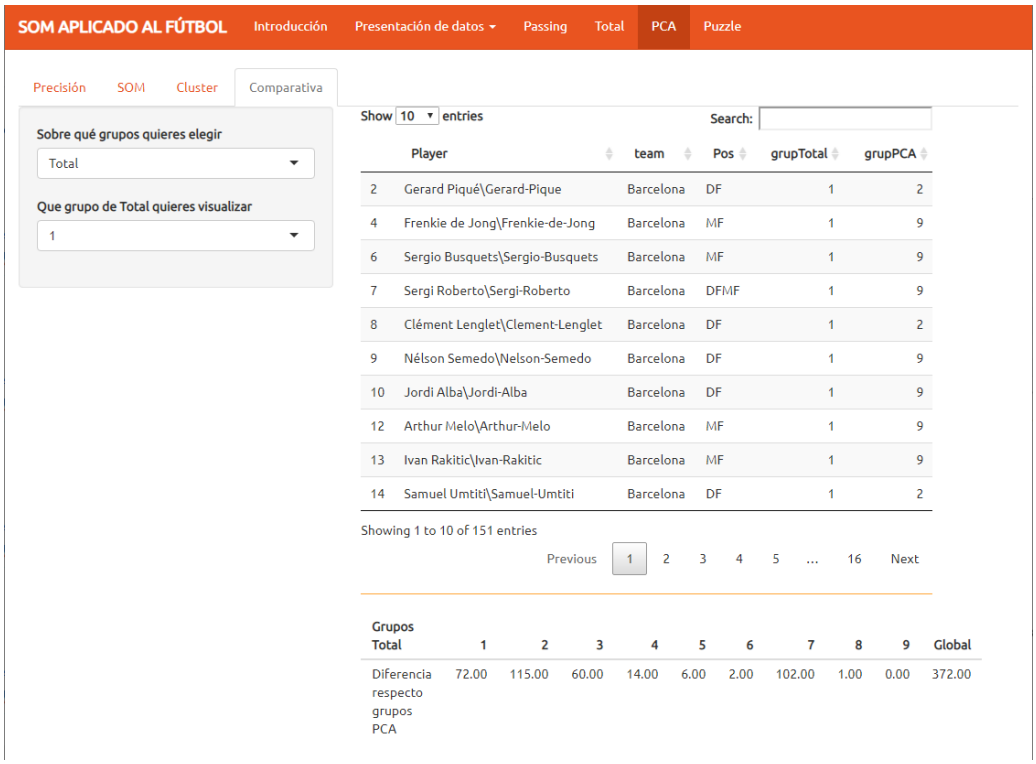


Figura 13 – Pestaña PCA, Comparativa

clasificación es que no hay mucha variabilidad con la primera ya que la mayoría de jugadores los agrupa con los mismos jugadores. Los grupos donde se agrupaban todos los jugadores de una misma posición continúan estando, pero esta vez los jugadores que se clasificaban en grupos mixtos aparecen agrupados con más criterio, en el sentido de sus posiciones.

En el caso de la pestaña donde se hace uso del PCA los resultados también fueron satisfactorios, ya que los resultados de la clasificación fueron casi idénticos a los realizados sin la reducción de dimensión de los datos, para experimentos hechos con un valor de tolerancia de 0.9. Como era de esperar, a menor tolerancia los resultados divergían más que con una tolerancia mayor. Después de esta experiencia, creemos que este tipo de herramientas pueden llegar a ser extremadamente útiles, sobretodo en casos donde los patrones de entrada tienen un dimensionalidad alta.

4. Conclusión

Este trabajo nos ha servido, principalmente, como breve introducción al vasto mundo de las redes neuronales. Por una parte, en la asignatura hemos estudiado desde el punto de vista teórico las redes neuronales, entendiendo no solo su filosofía como sistema sino también los conceptos matemáticos ocultos que las configuran.

Por otra parte, el proyecto final nos ha preparado para afrontar situaciones típicas que podría encontrar un científico de datos: hallar cuestiones interesantes y útiles sobre algún tema del que disponemos datos, informarnos sobre el tema, limpiar y analizar los datos, aplicar nuestro conocimiento teórico y extraer conclusiones de los resultados.

Como apunte final, queremos destacar el papel importantísimo que juega la computación en las matemáticas de hoy en día. No solo en este campo, donde la teoría matemática ya

va ligada a su aplicación de forma directa, sino también en otras áreas tradicionalmente más "puras"de las matemáticas, como análisis, álgebra o topología. Por ello, creemos firmemente que el aprendizaje de técnicas de computación es imprescindible hoy en día en la mochila de un matemático, y es importante contar con una formación sólida también en este aspecto.

Material suplementario

El código fuente de la plataforma se puede encontrar en

https://github.com/llosaba/SOM_Project.

Los datos concernientes a las estadísticas de los futbolistas están ubicados en la carpeta BBDD. Estos se distribuyen en cuatro carpetas, según si son relativos a los pases, tiros, posesión o acciones defensivas. Dichos datos se obtuvieron de la página web fbref.com.

Referencias

Calabuig, J. M. and García Raffi, L. M. and Sánchez Pérez, E. A. (2021).
Aprender como una máquina: Introduciendo la inteligencia artificial en la enseñanza secundaria.
Modelling in Science, Education and Learning, v. 14, n. 1, p. 45–52.

Fausett, L. (1994).
Fundamentals of Neural Networks: Architectures, Algorithms, and Applications.
Prentice-Hall international editions. Prentice-Hall.

Haykin, S. (1999).
Neural Networks: A Comprehensive Foundation.
International Edition. Prentice Hall.

Kohonen, T. (1982).
Self-organized formation of topologically correct featuremaps.
Biological Cybernetics 43, 59–69.

Murtagh, F.
hclust package.
https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/hclust

Wehrens, R. and Kruisselbrink, J.
Supervised and unsupervised self-organising maps.
https://github.com/cran/kohonen