

Escuela de Ingenierías Industrial, Informática y Aeroespacial

SISTEMAS DE INFORMACIÓN DE GESTIÓN Y BUSINESS INTELLIGENCE

FOOT4U

Autor: David Rosales Robles

(Diciembre, 2019)

Índice de contenidos.

1. Introducción	Pág. 04.
2. Objetivos	Pág. 05.
3. Herramientas/tecnologías utilizadas	Pág. 06.
4. Núcleo del trabajo	Pág. 10.
5. Análisis crítico	Pág. 24.
6. Líneas de futuro	Pág. 26.
7. Bibliografía	Pág. 27.

Índice de figuras.

- *Figura 1. Grafo de ejemplo.*
- *Figura 2. Estructura básica del lenguaje HTML.*
- *Figura 3. Información básica Neo4j.*
- *Figura 4. Esquema del grafo.*
- *Figura 5. Correspondencia entre identificadores y nombres de equipos.*
- *Figura 6. Instrucción LOAD CSV en Cypher.*
- *Figura 7. Instrucción para crear los equipos en Cypher.*
- *Figura 8. Instrucción para crear relaciones en Cypher.*
- *Figura 9. Importar librería GraphDatabase del paquete neo4j.*
- *Figura 10. Driver para establecer la conexión entre Python y Neo4j.*
- *Figura 11. Importar librería numpy.*
- *Figura 12. Definir e inicializar variables necesarias.*
- *Figura 13. Función charge_database.*
- *Figura 14. Función update_database.*
- *Figura 15. Sesión de conexión con Neo4j para cargar datos.*
- *Figura 16. Sesión de conexión con Neo4j para actualizar datos.*
- *Figura 17. Importar librerías para la regresión logística.*
- *Figura 18. Creación de los conjuntos train y test.*
- *Figura 19. Asignación de las clases.*
- *Figura 20. Creación del modelo de regresión logística.*
- *Figura 21. Predecir resultados con el conjunto de test.*
- *Figura 22. Mostrar métricas.*
- *Figura 23. Métricas y matriz de confusión.*
- *Figura 24. Interfaz (Primera parte).*
- *Figura 25. Código fuente JavaScript.*
- *Figura 26. Interfaz.*
- *Figura 27. Onclick botón Obtener Predicción.*
- *Figura 28. Importar librería eel.*
- *Figura 29. Seleccionar directorio eel.*
- *Figura 30. Cabecera función predict.*
- *Figura 31. Programar predicción.*
- *Figura 32. Predicción.*
- *Figura 33. Formatear probabilidades.*
- *Figura 34. Llamada JavaScript desde Python.*
- *Figura 35. eel start HTML.*
- *Figura 36. Resultado final.*
- *Figura 37. Script Aplicación.*
- *Figura 38. Pyinstaller. Crear .exe.*
- *Figura 39. Imagen Corporativa.*
- *Figura 40. Interfaz terminada.*

1. Introducción.

Ante el crecimiento exponencial de la cantidad de datos disponibles en el mundo de la informática y la creciente necesidad de hacer un uso inteligente de ellos, las formas tradicionales de gestionar bases de datos se están quedando obsoletas.

Por ello en este trabajo exploraremos una forma alternativa de almacenar y dar uso a nuestros datos. Esta nueva forma de trabajar consiste en una base de datos de grafos, que se caracteriza por una mayor sencillez y versatilidad así como por otorgar una mayor importancia a como están conectados los datos que en un modelo relacional.

Por otra parte, mi afición por el deporte, especialmente por el fútbol, y el auge de los mercados de apuestas deportivas me han llevado a plantearme si es posible pronosticar con precisión el resultado de un partido o en su defecto cómo se puede minimizar el error.

Teniendo en cuenta lo expuesto anteriormente me he decidido a realizar una aplicación que estime las probabilidades de producirse un determinado resultado en un partido. Debido a mi ubicación geográfica he elegido emplear resultados y equipos de la liga de Primera División de España, por la proximidad en el tiempo se han empleado las jornadas iniciales de la temporada 2019-2020.

Basándome en un sistema de aprendizaje automático se realizarán una serie de predicciones que se espera mejoren sensiblemente las probabilidades de acertar un resultado bajo el formato Quiniela (1x2).

Aunque en principio esto es sólo un trabajo académico, la continuación y el desarrollo de esta idea a lo largo del tiempo podría derivar en una aplicación que consiguiera enfrentarse y vencer a una casa de apuestas.

2. Objetivos.

El objetivo principal de este trabajo es desarrollar una aplicación que permita al usuario, de una manera sencilla, obtener una estimación de la probabilidad que tiene un resultado de fútbol de producirse.

Esta probabilidad será presentada como un porcentaje, existiendo tres opciones de resultado para cada partido a analizar: victoria local, empate o victoria visitante.

Para ello se contará con una base de datos, que almacene información sobre diferentes partidos; un código fuente, capaz de interpretar estos datos; y una interfaz, que facilite el uso del programa al usuario.

La base de datos contará con un número de partidos suficiente para poder aplicar algún algoritmo que nos permita generalizar de manera efectiva. Estos partidos pertenecerán exclusivamente a los disputados por los equipos participantes en la Primera División de España, conocida como LaLiga Santander por motivos de patrocinio, durante el transcurso de la temporada 2019-2020.

El código fuente deberá ser capaz de conectarse a la base de datos para extraer los datos relativos a los partidos. Posteriormente empleará estos datos para entrenar un algoritmo de aprendizaje automático que será el encargado de determinar los posibles resultados. Además deberá de ser capaz de mostrar estos resultados en una interfaz.

La interfaz será la parte del desarrollo en la que se invertirán menos recursos, debido al limitado tiempo del proyecto. Bastará con una interfaz clara y minimalista que permita realizar varias predicciones de forma consecutiva, hasta que el usuario decida dar por finalizada la sesión.

Teniendo en cuenta que existe una probabilidad de $\frac{1}{3}$ de acertar el resultado eligiéndolo de manera aleatoria consideraremos exitoso, para este proyecto, un algoritmo capaz de duplicar las probabilidades de acierto, colocándose en torno al 60% de precisión.

El proyecto deberá contar con un prototipo presentable a 18 de Diciembre de 2019, de manera que cualquier persona que asista a la presentación pueda hacerse una idea clara de en que consiste el trabajo, el porque se han escogido estas herramientas para desarrollarlo, las posibles mejoras que se podrían implementar y en general, del funcionamiento global de la aplicación.

A nivel personal el objetivo es familiarizarme con nuevas tecnologías de desarrollo y bases de datos.

3. Herramientas/tecnologías utilizadas.

- **Base de datos**

Para la construcción de la base de datos se ha empleado Neo4j; un software libre de bases de datos de grafos que salió al mercado en Febrero de 2010. Está implementado en Java y su principal diferencia con un sistema de bases de datos convencional es que la información se guarda en grafos, en lugar de tablas.

Un grafo está formado por nodos, los cuales representan una entidad y relaciones, que representan como se asocian dos nodos. Este tipo de estructura nos permite modelar cualquier tipo de escenario imaginable definido por relaciones.

A diferencia de otras bases de datos, en una base de datos de grafos se da prioridad a las relaciones, lo que permite no tener que definir “foreign-keys”. Además, el modelo de datos es significativamente más simple y visual que el de las bases de datos relacionales.

Actualmente, la necesidad de generar información a partir de un gran volumen de datos almacenados hace que las bases de datos de grafos tengan un gran futuro, pues dan más importancia a como se relacionan los distintos nodos.

Estas bases de datos están diseñadas específicamente para manejar datos altamente conectados y ofrecen una clara ventaja en cuanto a escalabilidad de las aplicaciones, así como un mejor rendimiento cuando se hace un manejo intensivo de las relaciones de datos.

El lenguaje que utiliza esta plataforma para crear, leer, actualizar y eliminar nodos, relaciones y propiedades recibe el nombre de Cypher.

Cypher es un lenguaje de consulta que permite interactuar de forma eficiente con un grafo de propiedades. Este lenguaje que en un principio estaba destinado a ser utilizado junto a Neo4j se abrió a través del proyecto openCypher en Octubre de 2015.

Este lenguaje de consulta ha sido diseñado para ser simple, pero poderoso, de manera que se pueden expresar fácilmente consultas de bases de datos altamente complejas; permitiendo al desarrollador centrarse en otras tareas y no perder recursos en el acceso a los datos.

Se muestra a continuación un ejemplo sencillo de grafo proporcionado por Neo4j en su manual de uso de Cypher.

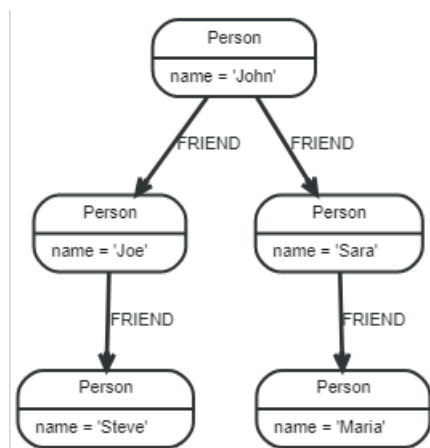


Figura 1. Grafo de ejemplo.

Como cláusulas principales, y que más se han utilizado durante el desarrollo de este trabajo, cabría destacar:

- MATCH → Para buscar un patrón.
- RETURN → Permite definir que incluir en el resultado de la consulta.
- WHERE → Agrega restricciones a un patrón de búsqueda.
- CREATE → Para crear nodos o relaciones.
- SET → Actualiza etiquetas y propiedades en nodos y relaciones.
- MERGE → Comprueba si un determinado patrón existe en el grafo.
- LOAD CSV → Para importar datos desde archivos .csv.

Para profundizar en estas y otras cláusulas se recomienda al lector consultar el manual sobre el uso de Cypher que nos proporciona Neo4j en su [sitio web](#).

• Código fuente

La aplicación está desarrollada en Python, un lenguaje de programación interpretado, dinámico y multiplataforma que soporta orientación a objetos, programación imperativa y programación funcional.

Fue creado a finales de los ochenta y alcanza su versión 1.0 en Enero de 1994. Se diseñó para ser leído con facilidad por lo que usa palabras donde otros lenguajes emplean símbolos, además el contenido de los distintos bloques de código es delimitado mediante espacios o tabuladores, lo que se conoce como indentación.

Admite el uso de comentarios y las variables se definen de forma dinámica, por lo que no hay que especificar su tipo de antemano.

Este lenguaje ha sido elegido debido a la gran cantidad de librerías relacionadas con la inteligencia artificial y funciones matemáticas complejas que posee y su facilidad para implementarlas; a diferencia de otros lenguajes de programación, cuyo uso es más

común, pero que elevaría demasiado la complejidad de un trabajo de estas características, pues no han sido diseñados para manejar grandes cantidades de datos

con el enfoque que se le pretende dar en Fut4j. Además cuenta con un sistema de conexión con Neo4j simple y efectivo, con un gran rendimiento.

La librería que permite la conexión entre nuestra base de datos y Python recibe el nombre de “GraphDatabase” y se incluye en el paquete neo4j, el cuál podemos instalar desde la línea de comandos mediante la instrucción “*pip install neo4j*”. Para obtener más información acerca de la instalación y un sencillo ejemplo de uso, tanto en Python como en otros lenguajes, podemos consultar el [sitio web](#) que Neo4j dedica a la documentación de los drivers que tienen la capacidad de proporcionarnos conectividad entre la base de datos y nuestra aplicación.

Para poder construir y manipular matrices se hará uso de la librería “numpy”, realizando su instalación de manera análoga a la realizada anteriormente con el driver de conexión de Neo4j. Para aprender más sobre el funcionamiento y las utilidades de esta librería podemos visitar el [manual de su página web](#), el cuál proporciona información detallada sobre los objetos, funciones y módulos que incluye.

Para la implementación del algoritmo de aprendizaje automático instalaremos el paquete “sklearn” haciendo uso de la instrucción “*pip install -U scikit-learn*” tal y como se nos indica en su [sitio web](#), en el que podemos consultar una extensa guía de uso y visualizar algunos ejemplos de lo que podemos hacer con esta utilidad. De entre todas las posibilidades facilitadas por sklearn, en este proyecto vamos a hacer uso de la librería “metrics” y “LogisticRegression”, perteneciente al paquete “sklearn.linear_model”.

Además, aunque para el resultado final no es necesario, emplearemos la librería “panda”; lo que nos permitirá mostrar mostrar por consola los resultados del clasificador entrenado, mientras se desarrolla la aplicación. De esta forma podemos valorar los resultados, comprender donde falla nuestro algoritmo y realizar los ajustes necesarios.

- **Interfaz**

Para desarrollar la interfaz se ha empleado HTML (HyperText Markup Language), un lenguaje de marcado que se usa para crear y representar visualmente páginas web. HTML emplea etiquetas, breves instrucciones de inicio y fin, las cuales permiten definir la forma en la que aparecerá en el navegador nuestro texto, imágenes y demás elementos.

```
<!doctype html>
<html>

  <head>
    <meta charset="utf-8"/>
    <title>Título de la web</title>
  </head>

  <body>
    Contenido de la web
  </body>

</html>
```

Figura 2. Estructura básica del lenguaje HTML.

Se ha personalizado la apariencia de los botones y cajas de selección, así como los colores de nuestra aplicación haciendo uso de CSS. Estas siglas significan Hojas de Estilo en Cascada (del inglés Cascading Style Sheets) y hacen referencia al lenguaje utilizado para describir la presentación de documentos HTML.

CSS ha sido diseñado para separar el contenido del documento y su forma de presentación, lo que se traduce en una mejora de la accesibilidad, la flexibilidad y el control; permitiendo que varios documentos HTML puedan compartir el mismo estilo haciendo uso de una sola hoja de estilos, que se encuentra separada en un archivo .css, reduciendo así la repetición de código de forma innecesaria.

Con la finalidad de poder establecer un canal de comunicación con nuestra aplicación en Python, se han incluido en el código HTML algunas funciones escritas en JavaScript, un lenguaje ligero e interpretado, orientado a objetos y dinámico, conocido como el lenguaje de script para páginas web.

Para conseguir esta comunicación empleamos una librería llamada eel, la cuál deberá ser incluida tanto en nuestro archivo .py como en nuestro .html, permitiendo la comunicación, a través de JavaScript, entre el núcleo de la aplicación y la interfaz. Para comprender mejor el funcionamiento de esta librería es recomendable acceder a la página de GitHub de [samuelhwiliams](https://github.com/samuelhwiliams); aunque de momento, para este proyecto, bastará con comprender que se basa en exponer funciones desde Python para que puedan ser usadas desde JavaScript y viceversa.

4. Núcleo del trabajo.

La base de datos ha sido construida en Neo4j haciendo uso del lenguaje de consulta Cypher y cuenta con la siguiente configuración:




Version	3.5.6 Enterprise
Status	RUNNING
Nodes	5060
Labels	2
Relationships	10080
Relationship types	1
IP address	localhost
Bolt port	7687 
HTTP port	7474 
HTTPS port	7473 

Figura 3. Información básica Neo4j.

Tras varias configuraciones y grafos de prueba, finalmente se ha optado por una estructura sencilla que podemos visualizar en cualquier momento desde Neo4j con la instrucción “*CALL db.schema()*”. Lo que en nuestro caso produce la siguiente salida:

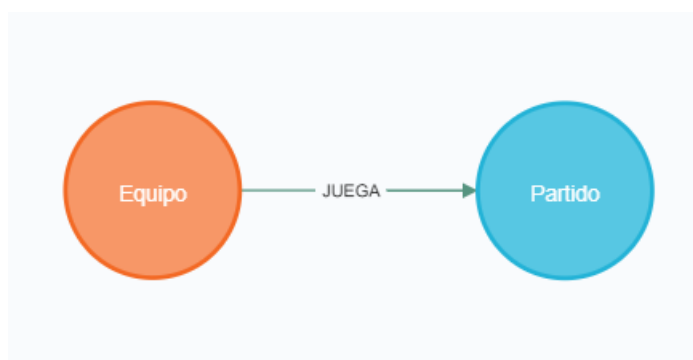


Figura 4. Esquema del grafo.

Para implementar esta base de datos se ha importado un archivo .csv, el cuál se proporciona junto a la aplicación y está compuesto de una fila para cada partido y 12 columnas en las que se representa la información relativa a cada encuentro. Estas columnas se corresponden con el equipo local, el equipo visitante, los goles del local, los goles del visitante, la posición en la clasificación del equipo local, la posición en la

clasificación del equipo visitante, los tiros a puerta realizados por el equipo local, los tiros a puerta realizados por el equipo visitante, las paradas efectuadas por el portero local, las paradas efectuadas por el equipo visitante, el resultado y la jornada.

El resultado se representa con 1, 0 ó 2; los cuales se corresponden con una victoria local, un empate y una victoria visitante respectivamente. Los equipos se representan con un identificador que se basa en el orden en el que debutaron en la temporada 2019-2020 en LaLiga Santander y se corresponde con la numeración que se proporciona a continuación.

```
1 -> Athletic Club
2 -> FC Barcelona
3 -> Celta de Vigo
4 -> Real Madrid
5 -> Valencia
6 -> Real Sociedad
7 -> Mallorca
8 -> Eibar
9 -> Leganés
10 -> Osasuna
11 -> Villarreal
12 -> Granada
13 -> Alavés
14 -> Levante
15 -> RCD Espanyol
16 -> Sevilla FC
17 -> Real Betis
18 -> Real Valladolid
19 -> Atlético de Madrid
20 -> Getafe
```

Figura 5. Correspondencia entre identificadores y nombres de equipos.

Para cargar este archivo en Neo4j empleamos la siguiente instrucción en Cypher, Lo que nos generará 90 nodos con la etiqueta partido y 12 propiedades por nodo, que se corresponden con cada una de las columnas de la tabla en el archivo .csv.

```
LOAD CSV WITH HEADERS FROM "file:///bd.csv" AS line CREATE (:Partido { Local:
line.Local, Visitante: line.Visitante, GolesLocal: line.GolesLocal, GolesVisitante:
line.GolesVisitante, PosicionLocal: line.PosicionLocal, PosicionVisitante:
line.PosicionVisitante, TirosLocal: line.TirosLocal, TirosVisitante:
line.TirosVisitante, ParadasLocal: line.ParadasLocal, ParadasVisitante:
line.ParadasVisitante, Resultado: line.Resultado, Jornada: line.Jornada })
```

Figura 6. Instrucción LOAD CSV en Cypher.

Lo ideal sería crear un script de Python que obtenga de forma automática todos estos datos de la web, pero debido al inexistente dominio de este lenguaje que tenía al comienzo de este proyecto, la complejidad de las páginas en las que se almacenan estos datos y la imposibilidad de encontrar una base de datos, que representara los datos que me interesaban, en plataformas como [Kaggle](#); me he visto obligado a introducir manualmente todos los datos, lo que explica el reducido número inicial de nodos.

Tras este pequeño inciso procederemos a explicar como se crean los nodos correspondientes a cada equipo y las relaciones en nuestra base de datos (es necesario hacerlo en este orden), para lo que se han incluido unas imágenes capturadas directamente desde la línea de comandos de Neo4j.

```
MATCH (n:Partido)
MERGE (e:Equipo { Nombre: n.Local})
RETURN e
```

Figura 7. Instrucción para crear los equipos en Cypher.

```
MATCH (n:Partido),(e:Equipo)
WHERE (n.Local=e.Nombre) OR (n.Visitante=e.Nombre)
CREATE (e)-[r:JUEGA]→(n)
RETURN r
```

Figura 8. Instrucción para crear relaciones en Cypher.

Una vez creados estos nodos y relaciones, tenemos la estructura de grafo que se muestra en la [Figura 4](#), al comienzo de este capítulo.

Con este escenario comenzamos a desarrollar nuestro código fuente en Python, que guardaremos en un documento que recibe el nombre de fut4j.py.

En primer lugar importamos la librería necesaria para conectarnos a nuestra base de datos y cargamos el driver.

```
from neo4j import GraphDatabase
```

Figura 9. Importar librería GraphDatabase del paquete neo4j.

```
driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "SISTGBI2019"))
```

Figura 10. Driver para establecer la conexión entre Python y Neo4j.

También será necesario definir las estructuras de datos que vamos a emplear más adelante para lo cuál, en este proyecto, necesitaremos la librería Numpy. Estas variables nos servirán para almacenar los datos que extraigamos de nuestra base de datos así como datos nuevos obtenidos a través de la ejecución de la aplicación.

```
import numpy as np
```

Figura 11. Importar librería numpy.

```
data_base = np.zeros(shape=(5040,12))
data_base = np.dtype('int64').type(data_base)
data = np.zeros(shape=(5030,16))
data = np.dtype('int64').type(data)
teams = np.zeros(shape=(20,8))
teams = np.dtype('int64').type(teams)

for i in range(20):
    teams[i][0] = i + 1
```

Figura 12. Definir e inicializar variables necesarias.

A continuación definiremos las funciones que nos permitirán, al ser llamadas, recuperar datos y actualizarlos en caso de ser necesario. En nuestro caso reciben el nombre de `charge_database` y `update_database` y se encargarán de recuperar la información de los partidos y actualizar la de los equipos, respectivamente.

```
#Cargar los datos desde neo4j
#-----
def charge_database(tx):
    print('Cargando base de datos...')
    aux=0
    for record in tx.run("MATCH (p:Partido)"
        "RETURN p.Local, p.Visitante, p.GolesLocal, p.GolesVisitante,"
        "p.PosicionLocal, p.PosicionVisitante, p.TirosLocal,"
        "p.TirosVisitante, p.ParadasLocal, p.ParadasVisitante, p.Resultado, p.Jornada"):
        data_base[aux]=[record["p.Local"],record["p.Visitante"],record["p.GolesLocal"],
            record["p.GolesVisitante"],record["p.PosicionLocal"],record["p.PosicionVisitante"],
            record["p.TirosLocal"],record["p.TirosVisitante"],record["p.ParadasLocal"],
            record["p.ParadasVisitante"],record["p.Resultado"],record["p.Jornada"]]
        aux=aux+1
    aux=0
```

Figura 13. Función `charge_database`.

```
#Actualizar datos de neo4j
#-----
def update_database(tx, id):
    tx.run("MATCH (n:Equipo { Identificador: $name })"
        "SET n.golesFavor=$golesFavor, n.golesContra=$golesContra,"
        "n.posicion=$posicion, n.tirosRealizados=$tirosRealizados,"
        "n.tirosRecibidos=$tirosRecibidos, n.paradasRealizadas=$paradasRealizadas,"
        "n.paradasRecibidas=$paradasRecibidas"
        , name=str(id), golesFavor=str(teams[id-1][1]), golesContra=str(teams[id-1][2]),
        posicion=str(teams[id-1][3]), tirosRealizados=str(teams[id-1][4]),
        tirosRecibidos=str(teams[id-1][5]), paradasRealizadas=str(teams[id-1][6]),
        paradasRecibidas=str(teams[id-1][7]))
```

Figura 14. Función `update_database`.

Ahora estamos preparados para establecer la conexión con nuestra base de datos y llamar a la función `charge_database` con el objetivo de cargar en la variable `data_base`, creada anteriormente, la información que tenemos de los partidos. Algo que realizaremos con las líneas de código expuestas en la siguiente figura.

```
#Sesion de conexion con nuestra base de datos
#-----
with driver.session() as session:
    session.read_transaction(charge_database)
driver.close()
```

Figura 15. Sesión de conexión con Neo4j para cargar datos.

La variable teams consta de 20 filas, una por cada equipo, y 8 columnas, que almacenan el identificador del equipo, los goles a favor, los goles en contra, su posición en la tabla, los tiros a puerta realizados, los tiros a puerta recibidos, las paradas efectuadas por su guardameta y las paradas efectuadas por los guardametas de los equipos rivales. Esto se consigue sumando los datos de todos los partidos que han disputado.

Nuestra matriz de entrenamiento recibe el nombre de data y esta compuesta por tantas filas como partidos tenemos y 17 columnas que harán referencia a los datos acumulados de los dos equipos que disputan el partido, de ahí que tengamos el doble de columnas que en la variable teams y una columna extra para el resultado.

Se evitará incluir en este documento el código con el que se generan estas matrices, con la finalidad de no alargarlo de manera innecesaria, puesto que es un desarrollo puramente matemático que se puede consultar de manera sencilla en el archivo fut4j.py que acompaña a esta memoria, dentro del bloque “#Definir matriz de entrenamiento”.

Ahora que tenemos los datos globales de cada equipo actualizados en la matriz teams crearemos una nueva sesión de conexión con nuestra base de datos para actualizar las propiedades de los equipos llamando a la función update_database.

```
#Cargar driver neo4j
#-----
driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "SISTGBI2019"))
#Sesion de conexion con nuestra base de datos
#-----
with driver.session() as session:
    print('Actualizando base de datos...')
    session.write_transaction(update_database, 1)
    session.write_transaction(update_database, 2)
    session.write_transaction(update_database, 3)
    session.write_transaction(update_database, 4)
    session.write_transaction(update_database, 5)
    session.write_transaction(update_database, 6)
    session.write_transaction(update_database, 7)
    session.write_transaction(update_database, 8)
    session.write_transaction(update_database, 9)
    session.write_transaction(update_database, 10)
    session.write_transaction(update_database, 11)
    session.write_transaction(update_database, 12)
    session.write_transaction(update_database, 13)
    session.write_transaction(update_database, 14)
    session.write_transaction(update_database, 15)
    session.write_transaction(update_database, 16)
    session.write_transaction(update_database, 17)
    session.write_transaction(update_database, 18)
    session.write_transaction(update_database, 19)
    session.write_transaction(update_database, 20)
driver.close()
```

Figura 16. Sesión de conexión con Neo4j para actualizar datos.

Es necesario volver a cargar el driver, tal y como apreciamos en la figura, puesto que fue cerrado, al cargar los datos, con el fin de optimizar el rendimiento.

Llegados a este punto, y tras cerrar de nuevo la conexión, no necesitaremos volver a la base de datos; por lo que comenzaremos a desarrollar nuestro algoritmo de predicción. Para ello se han valorado varias opciones y al final se ha elegido una regresión logística, un tipo de análisis utilizado para predecir el resultado de una variable que puede adoptar un número limitado de categorías.

Antes de comenzar a programar este sistema de aprendizaje automático debemos añadir a nuestro proyecto una librería que nos permita implementarla eficientemente y dos librerías más que nos permitirán mostrar la matriz de confusión, así como otra información relevante.

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import pandas as pd
```

Figura 17. Importar librerías para la regresión logística.

Ahora es el momento de separar nuestros datos en conjuntos de entrenamiento y de test, asignando de forma aleatoria el 80% y el 20% de los casos, respectivamente.

```
np.random.seed(123)
m_train = np.random.rand(len(data)) < 0.8
data_train = data[m_train,]
data_test = data[~m_train,]
```

Figura 18. Creación de los conjuntos train y test.

También debemos establecer la clase a las que pertenece cada caso, tanto en el conjunto de test, como en el de entrenamiento. Pudiendo, en este caso tomar los valores 1, 0 ó 2.

```
clase_train = data_train[:, -1]
clase_test = data_test[:, -1]
```

Figura 19. Asignación de las clases.

Con estos valores establecidos, y haciendo uso de las librerías de sklearn, crearemos el modelo de regresión logística en base a los datos del conjunto de entrenamiento.

```
modelo_lr = LogisticRegression(solver='lbfgs', multi_class='auto', max_iter=999999)
modelo_lr.fit(X=data_train[:, :-1], y=clase_train)
```

Figura 20. Creación del modelo de regresión logística.

A continuación, aplicamos nuestro modelo al conjunto de test y mostramos los resultados.

```
# PREDICCION
#-----
prediccion = modelo_lr.predict(data_test[:, :-1])
```

Figura 21. Predecir resultados con el conjunto de test.


```
# METRICAS
#-----
print(metrics.classification_report(y_true=clase_test, y_pred=prediccion))
print(pd.crosstab(data_test[:, -1], prediccion, rownames=['REAL'], colnames=['PREDICCION']))
```

Figura 22. Mostrar métricas

Estas instrucciones producen el siguiente resultado por consola:

	precision	recall	f1-score	support
0	0.64	0.54	0.58	282
1	0.65	0.79	0.71	452
2	0.84	0.68	0.75	248
accuracy			0.69	982
macro avg	0.71	0.67	0.68	982
weighted avg	0.69	0.69	0.68	982
PREDICCION	0	1	2	
REAL				
0	151	121	10	
1	75	355	22	
2	11	69	168	

Figura 23. Métricas y matriz de confusión.

Podemos apreciar como se mejoran ampliamente las probabilidades de acierto respecto a una predicción aleatoria. Destacando, por mucho, la precisión obtenida en victorias visitantes, lo que podría obedecer a la dificultad que entraña ganar fuera de casa en el fútbol y la necesidad de que el equipo visitante sea muy superior a su rival.

Además, podemos observar que la mayoría de equivocaciones, cuando se predice una victoria de algún equipo, suelen producirse debido a que la probabilidad de victoria, aunque superior al resto, no era demasiado alta, lo que se traduce en que algunas veces el otro equipo logre empatar. Estas probabilidades se verán más claramente en los resultados finales mostrados en la interfaz.

Las líneas correspondientes a mostrar las métricas por consola aparecerán comentadas en la versión final para minimizar los posibles elementos incomprensibles para el usuario. En caso de querer realizar alguna prueba bastará con eliminar los símbolos que indican comentarios para que se produzca la salida mencionada anteriormente.

Una vez construido el núcleo de la inteligencia artificial hay que proporcionar una manera sencilla y clara de utilizarla, para ello se implementará una interfaz en HTML. Desde ella nos comunicaremos con nuestro archivo .py por medio de JavaScript y haciendo uso de la librería eel.

El desarrollo de la interfaz web comienza con la creación de un archivo que recibirá el nombre de app.html en el que colocaremos los diferentes elementos que vayamos necesitando.

Comenzamos añadiendo dos etiquetas `<select>`, una para que el usuario pueda escoger el equipo local, y otra para el visitante. A continuación incorporaremos una etiqueta `<input>` de tipo `button`, que estará representada por el texto *Obtener Predicción* y que nos permitirá ejecutar nuestra consulta.

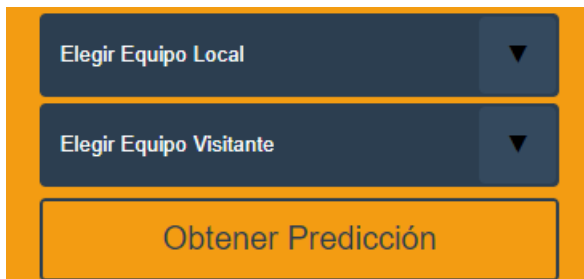


Figura 24. Interfaz (Primera parte).

Posteriormente creamos otro `<input>` de tipo `button` que contenga el texto *Salir*, para poder cerrar nuestra aplicación. Dejaremos entre ambos botones tres etiquetas `<div>`, inicialmente vacías y ocultas, en las que mostraremos luego las probabilidades de cada resultado; de esta manera el botón que cierra la aplicación se mantendrá siempre en la parte inferior de la misma.

En este momento debemos crear una etiqueta `<script>`, dentro de la cuál ubicaremos nuestro código JavaScript, esta es la parte más compleja del documento `.html`, motivo por el cual se muestra en este documento su código fuente.

```
<!-- Include eel.js - note this file doesn't exist in the 'web' directory -->
<script type="text/javascript" src="/eel.js"></script>
<script type="text/javascript">
    //document.getElementById('res').style.display = 'none';
    document.getElementById('pr1').style.display = 'none';
    document.getElementById('prx').style.display = 'none';
    document.getElementById('pr2').style.display = 'none';

    eel.expose(say_result_js); // Expose this function to Python
    function say_result_js(r,s,t,u) {
        //document.getElementById('res').style.display = 'block';
        document.getElementById('pr1').style.display = 'block';
        document.getElementById('prx').style.display = 'block';
        document.getElementById('pr2').style.display = 'block';
        //document.getElementById("res").innerHTML = r;
        document.getElementById("prx").innerHTML = s;
        document.getElementById("pr1").innerHTML = t;
        document.getElementById("pr2").innerHTML = u;
    }
    function fin() {
        window.close();
    }
</script>
```

Figura 25. Código fuente JavaScript.

En este script lo primero que hacemos es ocultar los contenedores, antes mencionados, en los que se ubicarán las probabilidades resultantes. Seguidamente elaboramos una función que será llamada desde Python y que en nuestro caso recibe el nombre de `say_result_js` y la exponemos para posibilitar su llamada desde un archivo externo mediante la instrucción `“eel.expose()”`. La tarea de esta función será hacer visibles los `<div>` en los que se guardarán los resultados y asignarles los valores que se han recibido como argumentos.

Por último se incluirá en el script una función `fin` que cierre la ventana, la cuál será llamada al pulsar el botón `Salir`.

A este archivo `.html` le daremos una mejor apariencia haciendo uso de una hoja de estilos que recibirá el nombre de `index.css` y que conseguirá que nuestra web luzca de la siguiente manera:

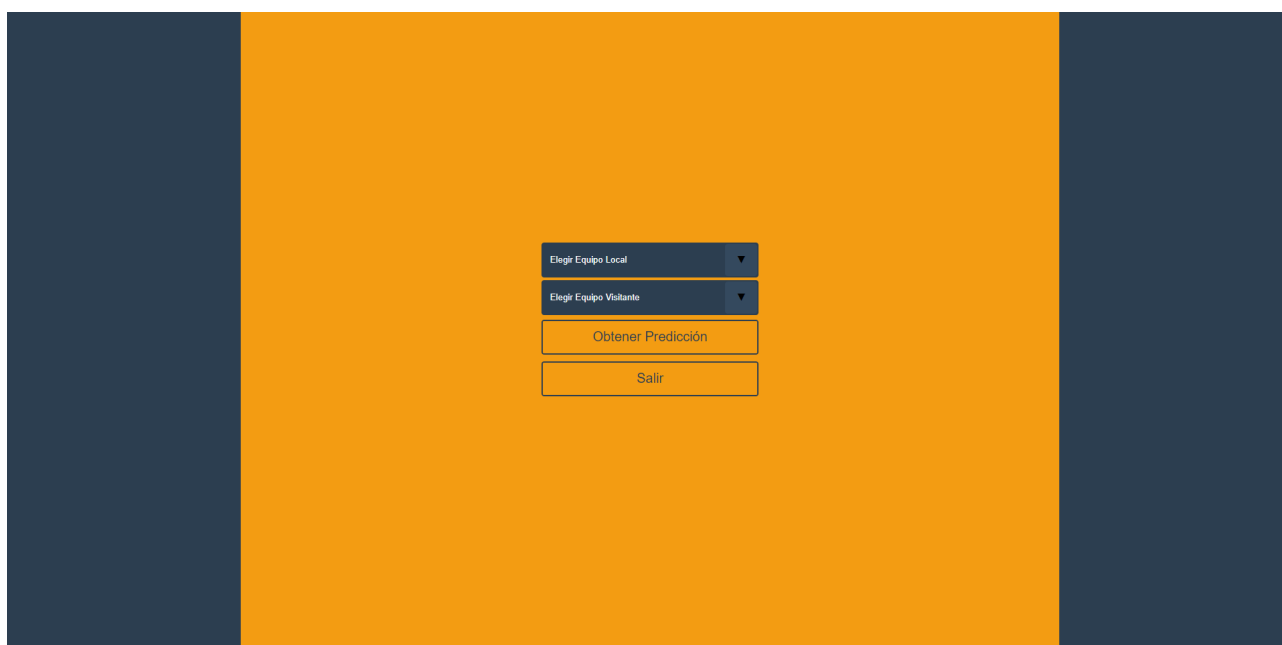


Figura 26. Interfaz.

Las franjas verticales han sido incluidas como forma de estilizar el diseño y teniendo en cuenta un diseño estándar que en un futuro, si el proyecto creciera y se convirtiera en una aplicación web accesible globalmente, permitiría incorporar publicidad en ellas si fuera necesario.

Llegados a este punto añadiremos al botón `Obtener Predicción` el siguiente parámetro y abandonaremos el documento HTML.

```
onclick="javascript:eel.predict(document.getElementById('local').value,document.getElementById('visitante').value);"
```

Figura 27. Onclick botón Obtener Predicción.

De vuelta en Python tendremos que importar nuestra última librería, que nos permitirá llamar a la función anteriormente declarada en JavaScript, para mandarle los resultados, y recibir una petición para ejecutar la función predict que implementaremos más adelante.

```
import eel
```

Figura 28. Importar librería eel.

Establecemos cual será el directorio en el que se ubicará nuestra interfaz.

```
# Set web files folder  
eel.init('web')
```

Figura 29. Seleccionar directorio eel.

En este punto desarrollamos la función que se encargará de determinar las probabilidades de que ocurra uno u otro resultado y que recibirá el nombre de predict. Es necesario exponerla mediante eel, de manera similar a lo efectuado en JavaScript, haciendo uso de la siguiente instrucción “@eel.expose”. Esta función será llamada desde el documento .html y recibirá como parámetros los dos equipos seleccionados por el usuario en formato numérico.

```
@eel.expose | # Expose this function to Javascript  
def predict(a,b):  
    # PREDICCIÓN  
    #-----  
  
    local = int(a)  
  
    visitante = int(b)
```

Figura 30. Cabecera función predict.

Conociendo los equipos sobre los que se va a realizar la consulta se crea una matriz de dimensiones 1x16 que equivaldría a una fila de las utilizadas durante el proceso de entrenamiento y se cargan en ella los datos que se tienen de ambos equipos.

```
pred = np.zeros(shape=(1,16))
pred = np.dtype('int64').type(pred)

#equipo local
pred[0][0] = local
#equipo visitante
pred[0][1] = visitante
#goles a favor local
pred[0][2] = teams[local-1][1]
#goles en contra del equipo local
pred[0][3] = teams[local-1][2]
#goles a favor visitante
pred[0][4] = teams[visitante-1][1]
#goles en contra del equipo visitante
pred[0][5] = teams[visitante-1][2]
#posicion del local
pred[0][6] = teams[local-1][3]
#posicion del visitante
pred[0][7] = teams[visitante-1][3]
#tiros a puerta del local
pred[0][8] = teams[local-1][4]
#tiros a puerta recibidos por el local
pred[0][9] = teams[local-1][5]
#tiros a puerta del visitante
pred[0][10] = teams[visitante-1][4]
#tiros a puerta recibidos por el visitante
pred[0][11] = teams[visitante-1][5]
#paradas del local
pred[0][12] = teams[local-1][6]
#paradas recibidas por el local
pred[0][13] = teams[local-1][7]
#paradas del visitante
pred[0][14] = teams[visitante-1][6]
#paradas recibidas por el visitante
pred[0][15] = teams[visitante-1][7]
```

Figura 31. Preparar predicción.

Con todo preparado para hacer uso del modelo entrenado anteriormente ejecutamos las siguientes líneas, que nos permitirán obtener un resultado probable y las probabilidades de que ocurra cada uno de los tres resultados.

```
miprediccion = modelo_lr.predict(pred)
probabilidades_prediccion = modelo_lr.predict_proba(pred)
```

Figura 32. Predicción.

Le damos el formato deseado:

```
prx= int(round(probabilidades_prediccion[0][0]*100))
pr1= int(round(probabilidades_prediccion[0][1]*100))
pr2= int(round(probabilidades_prediccion[0][2]*100))

px= "Probabilidad de empate: " + str(prx) + "%"
p1= "Probabilidad de victoria local: " + str(pr1) + "%"
p2= "Probabilidad de victoria visitante: " + str(pr2) + "%"
```

Figura 33. Formatear probabilidades.

Y procedemos a llamar a la función de JavaScript que nos permitirá visualizar estos resultados en nuestra aplicación web.

```
eel.say_result_js(m, px, p1, p2)
```

Figura 34. Llamada JavaScript desde Python.

Por último añadiremos una línea final que permita a la librería eel saber que archivo es en el que están contenidas las especificaciones de la interfaz.

```
eel.start('app.html')
```

Figura 35. eel start HTML.

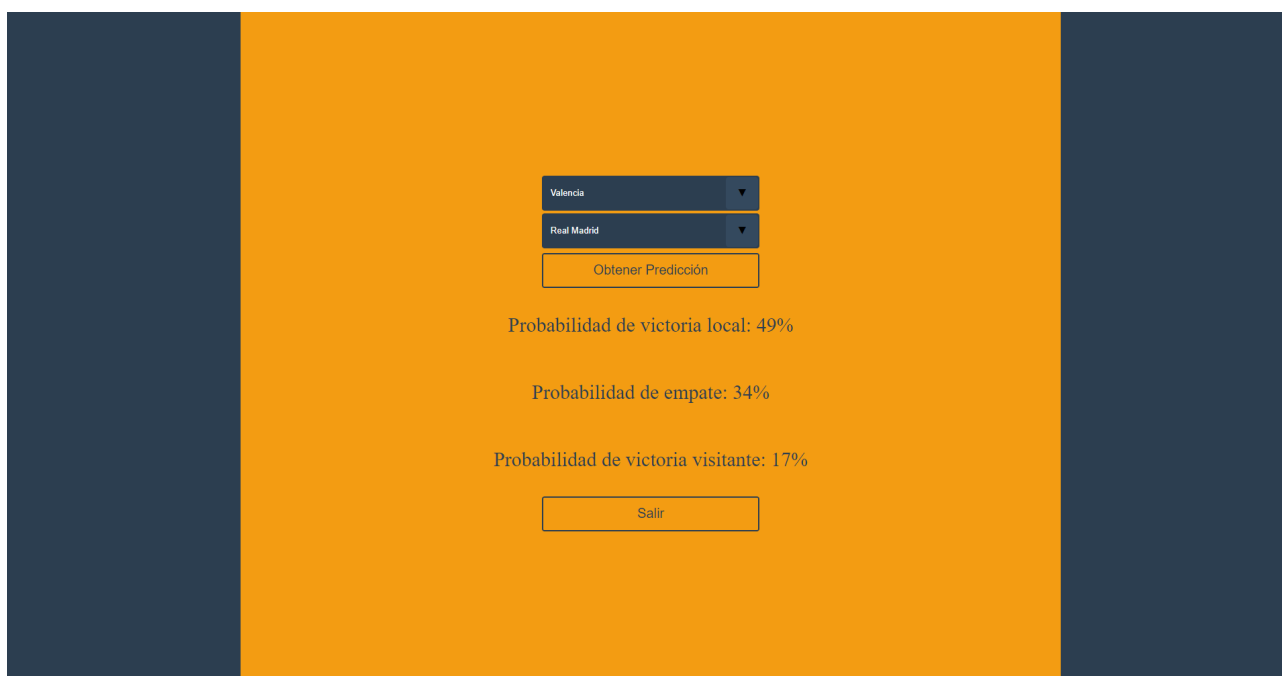


Figura 36. Resultado final.

La estructura de directorios proporcionada debe mantenerse para un correcto funcionamiento de la aplicación.

```
>Foot4u\  
|   >Aplicacion.exe  
|   >fut4j.py  
|   >web \  
|   |   >app.html  
|   |   >index.css  
|   |   >images  
|   |   |   >logo.png
```

Como forma de simplificar la ejecución creamos un script de Python que recibirá el nombre de Aplicacion.py y se limitará a iniciar todo. Para ello bastará con dos líneas de código.

```
import os  
os.system ("python fut4j.py")
```

Figura 37. Script Aplicacion.

Para mayor comodidad crearemos un ejecutable de formato .exe ejecutando la siguiente instrucción.

```
pyinstaller --windowed --onefile --icon=./logo.ico Aplicacion.py
```

Figura 38. Pyinstaller. Crear .exe.

Logo.ico hace referencia a un archivo en formato .ico que ha sido creado como imagen corporativa de la aplicación, e incluido en la carpeta del proyecto.



Figura 39: Imagen Corporativa.

Posteriormente he incluido este icono en la página web para mejorar su aspecto visual, pasando a tener el aspecto que se mostrará a continuación.

En la estructura de directorios, por lo tanto, se incluirá una carpeta llamada images que contendrá el logo en formato .png.

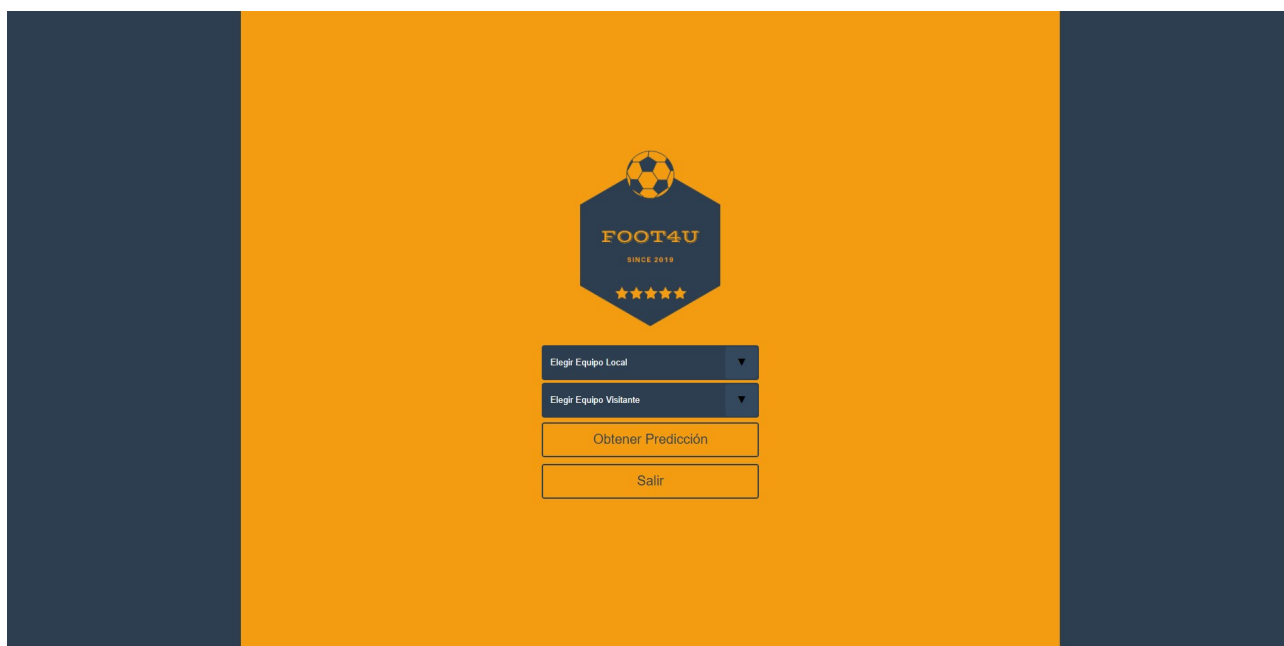


Figura 40. Interfaz terminada.

Se ha incluido también en el directorio una copia del archivo .csv, utilizado para construir la base de datos, bajo el nombre bd.csv, para facilitar la reproducción desde 0 de este proyecto.

Finalmente se ha elegido como nombre de la aplicación FOOT4U, en pequeño guiño a Neo4j, que quiere representar la idea de la aplicación de ser accesible y para todo tipo de públicos.

5. Análisis crítico.

Para comenzar este apartado se ha realizado un DAFO, que consiste en un análisis de las características internas y de la situación externa del proyecto. Es una herramienta de estudio de la situación de una empresa o proyecto con un amplio uso en los sectores financieros.

DEBILIDAD	AMENAZA
<ul style="list-style-type: none"> Necesidad constante de actualizarse para funcionar de manera óptima. Imposibilidad de medir la mentalidad de los deportistas y otras variables que tienen peso en el resultado. 	<ul style="list-style-type: none"> Conflicto de intereses con las casas de apuestas. Posibles cambios en la normativa FIFA.
FORTALEZA	OPORTUNIDAD
<ul style="list-style-type: none"> Escalabilidad del proyecto. Posibilidad de detectar patrones que se nos escaparían sin una inteligencia artificial. 	<ul style="list-style-type: none"> Crecimiento exponencial de los mercados de apuestas deportivas. Aumento de la cantidad de datos públicos que se recopilan sobre los partidos.

El resultado obtenido con este proyecto es en general satisfactorio, puesto que la aplicación, haciendo uso de una base de datos propia, es capaz de aumentar significativamente las probabilidades de acertar un resultado. Partiendo de esta base las posibilidades de mejora son casi infinitas debido al crecimiento de los mercados de apuestas deportivas y el aumento de los datos disponibles.

Considero que el trabajo es muy escalable, y que a partir de él se pueden montar aplicaciones más complejas, que incluyan mayor número de variables o que realicen predicciones relativas a otros deportes o ligas.

Si se quisiera comercializar o simplemente emplear una aplicación de estas características con el fin de ganar dinero apostando habría que tener en cuenta el poder que tienen las casas de apuestas y la posibilidad de que te cierren la cuenta antes de conseguir beneficios significativos.

Como punto negativo resaltar que me ha resultado imposible, a pesar de los esfuerzos, conseguir implementar un rastreador web usando Scrapy, debido al complejo formato que presentan las páginas en las que están alojados los datos de interés para este proyecto. Conseguir esto implicaría un aumento significativo de el número de partidos que se podrían analizar, lo que, a priori, aumentaría la precisión de las predicciones. Además quedaría solucionada una de las debilidades expuestas en el DAFO puesto que se podría

programar el script para que se ejecutara cada cierto tiempo, manteniendo la base de datos actualizada constantemente.

Además debido al limitado tiempo con el que se contaba para realizar el trabajo me he visto obligado a apostar por una interfaz sencilla, que si bien funciona correctamente, considero que podría ser más visual e incorporar elementos gráficos.

6. Líneas de futuro

En el futuro, incorporando un script de Python, que sea capaz de obtener datos en tiempo real de webs especializadas, para actualizar la base de datos la aplicación alcanzaría su funcionamiento óptimo.

Personalmente creo que podría ser interesante explorar como influyen otras variables en el resultado. Recomendando incluir el valor que tienen en el mercado los jugadores de cada equipo, lo que nos permitirá cuantificar de alguna manera lo determinantes que son, a la vez que se tienen en cuenta los traspasos de jugadores; y las dimensiones del campo en el que se va a jugar el partido, que nos proporcionarán información sobre el tipo de campo en el que mejor se desenvuelve un equipo.

Para mejorar aún más el algoritmo, si se cuenta con una muestra de partidos suficientemente amplia, incluir los árbitros puede suponer una mejora en los resultados. Además se deberían tener en cuenta las posibles lesiones y sanciones con las que llega un equipo al partido.

A nivel técnico sería interesante almacenar tanto la base de datos como la página HTML en un servidor, lo que permitirá a potenciales nuevos usuarios descargarse un único ejecutable para hacer funcionar la aplicación, aumentando la sencillez y velocidad de utilización del programa.

Por otra parte, aunque en este caso me he centrado en el fútbol por gustos personales, esta método de trabajo es aplicable a cualquier otro deporte y mercado de apuestas. Por lo que podría ser una buena opción explorar mercados en los que las posibilidades iniciales sean de $\frac{1}{2}$ y que además tengan un fuerte componente matemático. Un ejemplo claro de este tipo de apuesta sería el mercado over/under de NBA, que consiste en pronosticar si un determinado partido superará un número de puntos establecido.

Como cierre, me gustaría advertir al lector de que las apuestas deportivas siempre seguirán siendo un juego de azar en el que acotar las probabilidades no nos garantiza el éxito; por ello, aún haciendo las mejoras pertinentes a este proyecto y logrando unos números que en un entorno de laboratorio debieran dar beneficios económicos, se recomienda no apostar nunca dinero que no se esté dispuesto a perder.

7. Bibliografía

CSS. MDN. (2019). Recuperado 17 de Diciembre de 2019, de <https://developer.mozilla.org/es/docs/Web/CSS>

Curso Python. Pildoras Informaticas. (2017). Recuperado 17 de Diciembre de 2019, de <https://www.youtube.com/watch?v=G2FCfQj-9ig&list=PLU8oAIHdN5BlvPxziopYZRd55pdqFwkeS>

Estructura de una página HTML. (2015). Recuperado 17 de Diciembre de 2019, de https://cristiansystemf-cur.webnode.es/_files/200000011-e17d7e3715/HTML5%20estruc.PNG

HTML. MDN. (2019). Recuperado 17 de Diciembre de 2019, de <https://developer.mozilla.org/es/docs/Web/HTML>

JavaScript. MDN. (2019). Recuperado 17 de Diciembre de 2019, de <https://developer.mozilla.org/es/docs/Web/JavaScript>

NumPy Reference. NumPy v1.17 Manual. (2019). Recuperado 17 de Diciembre de 2019, de <https://docs.scipy.org/doc/numpy/reference/>

Python Graph Database. Neo4j. (2019). Recuperado 17 de Diciembre de 2019, de <https://neo4j.com/developer/python/>

Python GUI Using Chrome. Nitratine. (2018). Recuperado 17 de Diciembre de 2019, de <https://nitratine.net/blog/post/python-gui-using-chrome/>

Regresión Logística - Práctica con Python. Ligdi González. (2019). Recuperado 17 de Diciembre de 2019, de <http://ligdigonzalez.com/algoritmo-regresion-logistica-machine-learning-practica-con-python/>

Regresión Logística. Santiago de la Fuente Fernández. (2011). Recuperado 17 de Diciembre de 2019, de <http://www.estadistica.net/ECONOMETRIA/CUALITATIVAS/LOGISTICA/regresion-logistica.pdf>

Samuel H. Williams. EEL. (2019). Recuperado 17 de Diciembre de 2019, de <https://github.com/samuelhwilliams/Eel>

Scikit-learn. Machine Learning in Python. (2019). Recuperado 17 de Diciembre de 2019, de <https://scikit-learn.org/stable/index.html>

The Neo4j Cypher Manual v3.5. Neo4j. (2019). Recuperado 17 de Diciembre de 2019, de <https://neo4j.com/docs/cypher-manual/current/>

Welcome to Python.org. (2019). Recuperado 17 de Diciembre de 2019, de <https://www.python.org/>

Why Graph Databases?. Neo4j Graph Database Platform. (2019). Recuperado 17 de Diciembre de 2019, de <https://neo4j.com/why-graph-databases/?ref=footer>