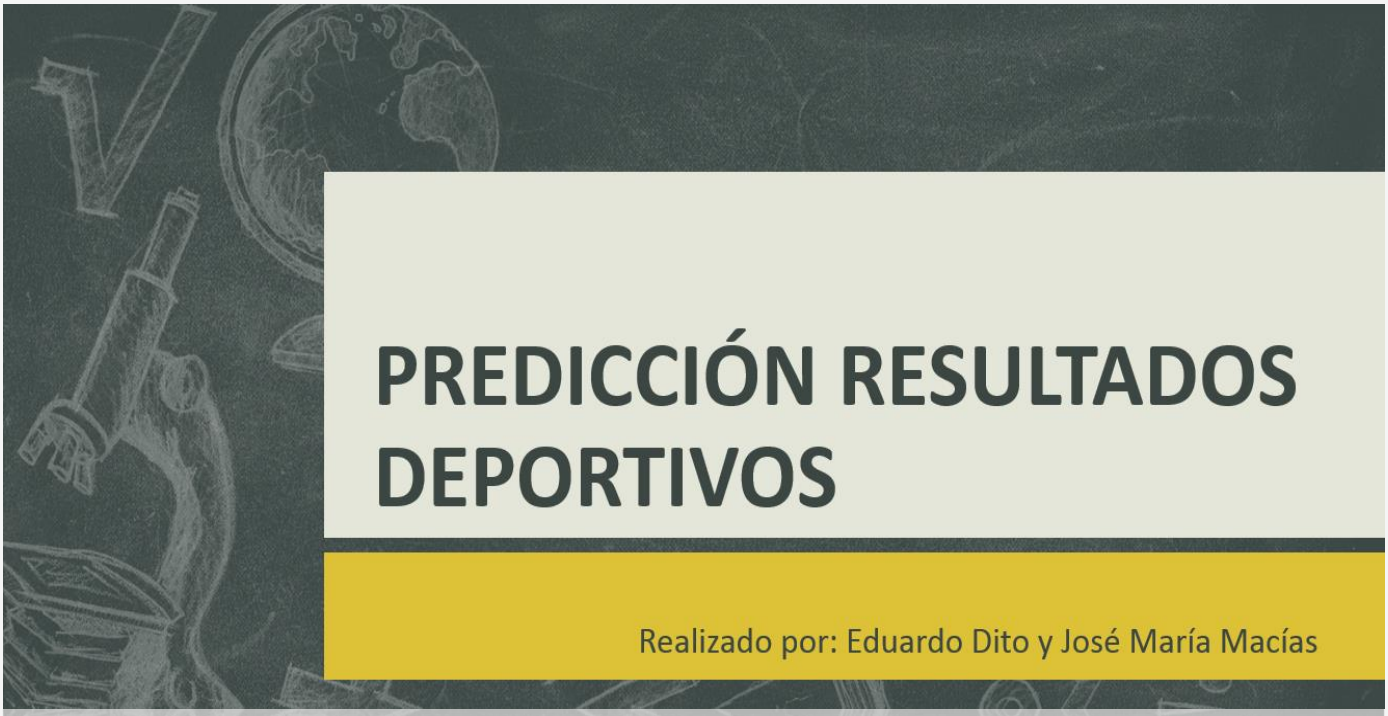
MEMORIA DEL PROYECTO



NEOLAND DATA SCIENCE BOOTCAMP

Enero-Marzo 2020

RESUMEN

En lo que respecta al fútbol, las predicciones sobre victorias o derrotas suelen ser complicadas y tienen un nivel de acierto poco significativo.

Sin embargo, en los últimos años, el auge de las casas de apuestas y el largo historial de análisis de datos y estadísticas en los deportes profesionales de E.E.U.U. (NBA, NFL…), han despertado el interés en un amplio abanico de variables que nos permiten trabajar con datos mucho más concretos (número de córners, disparos en un partido, cantidad de tarjetas, porcentaje de precisión en los pases, centros al área, ratios de remates, etc.).

La profesionalización y el auge económico que vive el fútbol, han llevado a un enorme nivel de estudio cualquier característica relacionada con este, por lo que no es raro encontrar hoy en día profesionales en los grandes equipos dedicados a la Psicología Deportiva o Nutrición para alto rendimiento, por ejemplo. Este nivel de estudio se lleva también al campo de la Ciencia de Datos y el Big Data.

La intención de nuestro trabajo es predecir el resultado de una de estas variables, el **número de córners**, que generará cada equipo en su último encuentro de liga.

MOTIVACIÓN Y OBJETIVO

Cuando te apasiona el fútbol y los datos, aunque a priori parezcan dos campos totalmente distintos, buscas la manera de unir las ambas cosas.

El objetivo de este proyecto, es encontrar un modelo que sirva de ayuda para predecir los datos que genera cada equipo en un partido, y que en manos de una persona con conocimientos de fútbol pueda generar rendimientos económicos mediante apuestas deportivas.

ANTECEDENTES

Era cuestión de tiempo que la Ciencia de Datos terminara por llegar al fútbol, a fin de cuentas, es algo que se lleva haciendo desde hace años en otros deportes profesionales, e incluso décadas, de una forma más rudimentaria.

Prácticamente todos los modelos existentes están centrados en predecir las victorias de los equipos y/o los ganadores de diferentes torneos y no en las estadísticas concretas de cada juego.

Para valorar los modelos y técnicas para predecir eventos, el ML puede ser de gran ayuda, pero hay que saber interpretar los resultados y valorar los modelos. Aquí un breve repaso:

- Modelo **FiveThirtyEight**: Creado en 2009, es el método más popular, consiste en usar una puntuación media para cada equipo que se define como el resultado (ofensivo y defensivo) que obtendría un equipo jugando contra otro equipo de nivel medio.

  Muchos modelos se basan en FiveThirtyEight, ya sea modificando su forma de puntuación media, aplicándola a los jugadores, etc.

- **Microsoft**, a través de su buscador Bing, realiza análisis basados en BigData y M.Learning para predecir resultados de partidos de fútbol. Una de sus principales peculiaridades es que tiene en cuenta la percepción de los usuarios a través de las búsquedas en sus plataformas., aunque de alguna forma ponderada, si no, el equipo con mayor número de aficionados, tendría los mejores números independientemente de su nivel de juego. Tiene el mayor % de acierto en fases eliminatorias (formatos play-offs, 1vs1, etc.). Su código no está liberado.

- **Google** creo para el Mundial de 2014 un modelo basado en la Reg. Logística para formatos de eliminatoria (no hay empates). Con un acierto de 14 sobre 16, posteriormente, liberaron el código en Github.

MATERIAL Y HERRAMIENTAS

I. Herramientas de IA y Machine Learning

Por un lado, los métodos de **Machine Learning** usados han sido los siguientes:

- KNN (K-Nearest Neighbors)
- Naive-Bayes
- SVM (Support Vector Machines)
- Decission Tree
- Random Forest
- Adaboost Algorithm

Todos los métodos de Machine Learning han sido utilizados junto al método de validación Leave One Out, además, tanto al método Random Forest como al Adaboost se les ha aplicado Feature Selection para mejorar sus resultados, dado que estos algoritmos permiten su utilización.

Por otro lado, los métodos de **IA** han sido:

- Perceptron
- Red Neuronal Simple

II. Descripción de los Datos:

Trabajaremos con un dataset donde la mayoría de datos son numéricos y está formado por columnas que registran los datos de cada jugador, por equipo y en cada partido de cada jornada desde la primera hasta la 38. Los datos recogen las estadísticas respecto a Tiros, Tiros a Puerta, Centros al área, Centros Precisos… y nuestra **variable, CF** (Córners Forzados).

DataFrame Jugadores3:

	Titulares	P	CA	T	TaP	C	CP	CF	FR	FC	PI	BR	Reg	PA	P%	partido	equipo	localizacion	Jornada
0	Yassine Bono	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0	0.0	0	0	86.7	1	girona	local	1
1	Bernardo Espinosa	NaN	NaN	0.0	0.0	0.0	0.0	0.0	1.0	1	1	2.0	0	0	92.2	1	girona	local	1
2	Juanpe Ramírez	NaN	NaN	0.0	0.0	0.0	0.0	0.0	1.0	3	4	1.0	0	1	90.0	1	girona	local	1
3	Marc Muniesa	NaN	NaN	1.0	0.0	2.0	0.0	0.0	1.0	1	2	0.0	1	2	87.3	1	girona	local	1
4	Pedro Porro	NaN	NaN	0.0	0.0	6.0	1.0	1.0	1.0	0	0	3.0	2	0	86.4	1	girona	local	1

Nuestro dataset registra los datos de la temporada pasada **(2018-2019)**, la intención es comparar con los resultados reales nuestro modelo, y en caso de que sea un modelo preciso, utilizarlo para los datos de la próxima temporada con el fin de predecir los datos de futuras jornadas.

Contamos con **14400 filas y 17 columnas** antes de la limpieza del dataset, de las cuales, todas excepto 3 son numéricas (jugador, equipo, local/visitante). Tras la limpieza lo reducimos a **8360 filas y 11 columnas**.

El trabajo ha sido realizado enteramente en **Python**. Las librerías utilizadas para el manejo del Dataset han sido **Pandas** y **Numpy** esencialmente, aunque también nos hemos ayudado de otras como **Seaborn** e **Itertools**. También hemos utilizado herramientas básicas de Python como replace, groupby, iloc, bucles y funciones, etc.

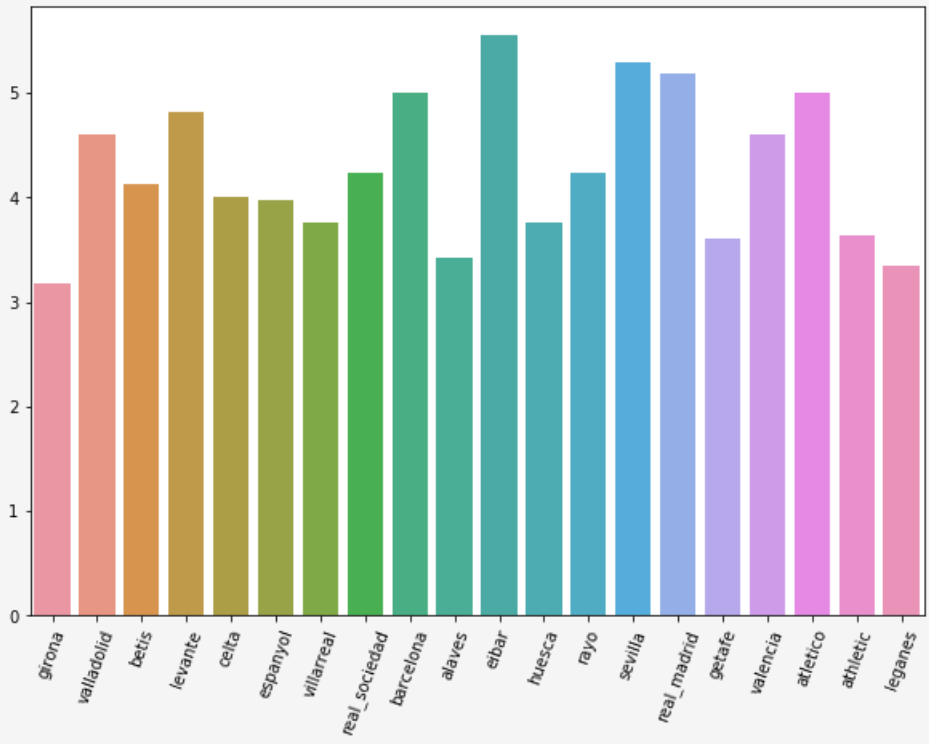
Para la obtención de datos hicimos uso de web scrapping, utilizando las librerías **Selenium** y **BeautifulSoup**.

Por último, para los métodos de **Machine Learning** e **IA** utilizamos las librerías **sklearn** (Scikit-Learn) y **keras** respectivamente.

III. Gráfica de los Datos iniciales

Cantidad de córners que genera cada equipo por partido:

EQUIPO	CF/P
Girona	3,18
Valladolid	4,61
Betis	4,13
Levante	4,82
Celta	4,00
Espanyol	3,97
Villarreal	3,76
Real_Sociedad	4,24
Barcelona	5,00
Alaves	3,42
Eibar	5,55
Huesca	3,76
Rayo	4,24
Sevilla	5,29
Real_Madrid	5,18
Getafe	3,61
Valencia	4,61
Atletico	5,00
Athletic	3,63
Leganes	3,64



MÉTODOS PROPUESTOS

En este apartado, se describirán por separado los métodos propuestos y utilizados en cada bloque del proyecto.

- **1º Parte: Webscrapping mediante Beautifulsoup y Selenium**

Paralelamente, probamos un método basado en Selenium y BeautifulSoup y otro basado solamente en BeautifulSoup, debido a que la página imposibilitaba el acceso prolongado con Selenium, escogimos un método que solo usaba BeautifulSoup, en el cual generábamos una lista con la url de todos los encuentros que se produjeron durante la temporada y posteriormente un bucle que nos permitía descargar en forma de dataframe las tablas de estadísticas de cada encuentro (tanto la del equipo local como la del visitante) de la lista de urls. **(Código adjunto en el Anexo).**

- **2º Parte: Tratamiento y montaje del primer dataset**

Con los datos obtenidos mediante webscrapping, armamos un primer dataset que recoge todos los datos que vamos a utilizar a posteriori. Para esto nos servimos de las librerías **numpy** y **pandas** principalmente, bucles, y otras herramientas por defecto de Python (groupby, replace…).

Se añaden columnas necesarias que no figuraban en el dataset inicial como número de partido y número de jornada, se convierten a numéricos todos los datos (tras hacer webscrapping se descargan como strings). Después, se asigna un id numérico a cada equipo y otro a cada jugador.

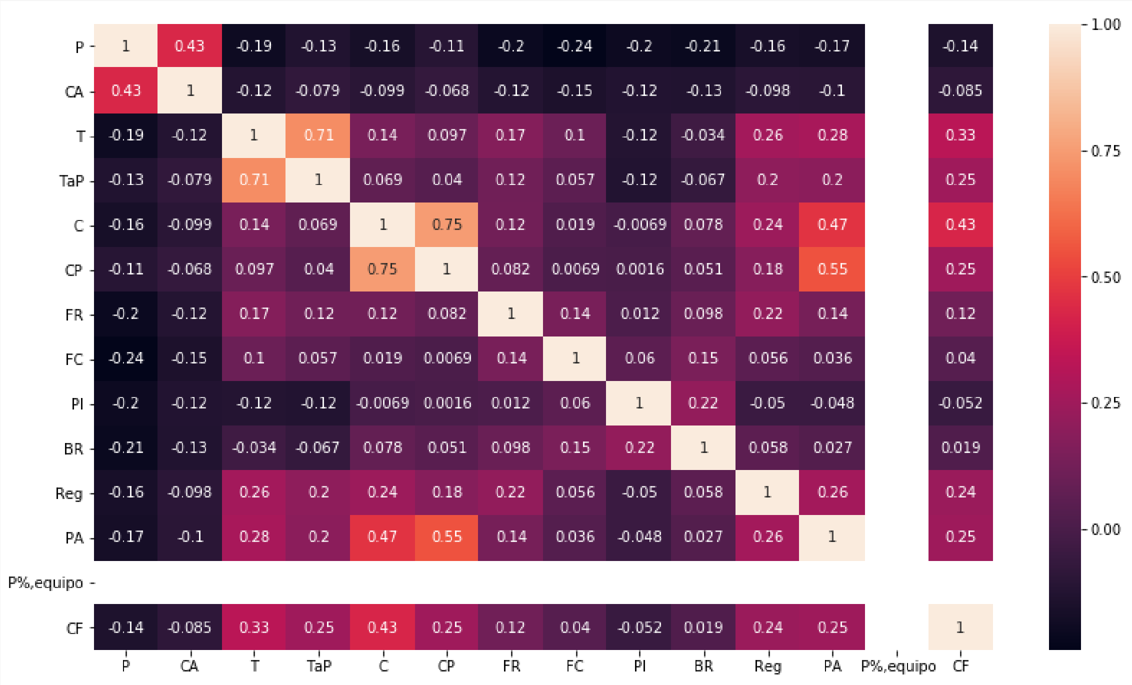
Agrupamos nuestra **variable objetivo, de forma categórica por rangos** (0 para 0-3 córners, 1 para 4-6 córners y 3 para más de 7 córners). Esta agrupación se hace en base a los tipos de rangos que ofrecen las casas de apuestas a la hora de establecer cuotas y a los resultados más frecuentes de nº de córners.

- **3º Parte: Estudio y análisis de los datos**

Mediante el mapa de correlaciones (**seaborn**), vemos las columnas con mayor correlación respecto a nuestra variable objetivo (Córners Forzados), junto a la tabla de correlaciones, nos muestra qué columnas son innecesarias, y pueden ser eliminadas. También decidimos eliminar los

datos de jugadores suplentes, y los minutos en los que se efectúan los cambios, pues son generadores de ruido.

Imagen del mapa de correlaciones:



- **4º parte: Transformación del dataset a forma vectorial**

El dataset será limpiado y tratado para ser trabajado en forma de líneas vectoriales, donde cada vector representa los datos de una jornada en función de las dos anteriores, por lo que pasa a ser un DataFrame de **36 filas y 365 columnas.**

Cada vector recoge los datos de nuestra variable por jornada en función de los datos particulares que arroja cada jugador/equipo para cada 3 jornadas (jornada 1, 2 y 3; 2, 3 y 4… 36, 37 y 38). Para elaborar este data set, usamos funciones y bucles. **(*Código adjunto en el Anexo).**

Para comprender este cambio, nos ayudaremos de una imagen que ilustra la forma en la que quedaría cada vector (o fila) de este nuevo dataset:

<table>
<tr><td colspan="23">Jornada 1</td><td colspan="16">Jornada 2</td><td colspan="6">Jornada 3: Datos Conocidos</td><td>…</td><td>Y</td></tr>
<tr><td colspan="4">Partido</td><td colspan="6">Portero</td><td colspan="6">Defensa 1</td><td colspan="2">…</td><td colspan="5">Delantero 2</td><td colspan="5">Partido</td><td colspan="6">Portero</td><td>…</td><td colspan="4">Del 2</td><td colspan="2">Partido</td><td colspan="4">Jugadores</td><td>…</td><td></td></tr>
<tr><td>Partido</td><td>Equipo</td><td>Local</td><td>Total Corners</td><td>Jugador</td><td>T</td><td>TP</td><td>C</td><td>CP</td><td>CF</td><td>Nombre</td><td>T</td><td>TP</td><td>C</td><td>CP</td><td>CF</td><td>…</td><td>Nombre</td><td>T</td><td>TP</td><td>C</td><td>CP</td><td>CF</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>Equipo</td><td>Local</td><td>Portero</td><td>Def 1</td><td>…</td><td>Del 2</td><td>…</td><td>N.º Corners</td></tr>
<tr><td>1</td><td>Girona</td><td>SI</td><td>5</td><td>Yasinne Bono</td><td>2</td><td>2</td><td>-</td><td>-</td><td>-</td><td>B. Espinosa</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td></td><td>Estuani</td><td>3</td><td>1</td><td>0</td><td>0</td><td>1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>Girona</td><td>Si</td><td>Bono</td><td>Pepito</td><td>…</td><td>Stuani</td><td></td><td>4</td></tr>
</table>

En la imagen se puede apreciar cómo se forma cada **vector de datos (en gris**), con los datos por partido y jugador para la primera y segunda jornada (naranja), y los datos conocidos del tercer partido (verde) y la **variable objetivo**, nº de córners **(azul)**. Así, cada vector, nos da los resultados de los córners en un partido, en función de los registros de los dos partidos anteriores y los datos conocidos de ese mismo partido, antes de ser jugado (local/visitante, equipo y nombres de los jugadores). Los nombres de las columnas en la imagen, son una ayuda meramente visual, en el dataset con que trabajamos quedarían con un índice numérico para poder trabajar con ellos más fácilmente a posteriori.

- **5º parte: Aplicación de Machine Learning al dataset vectorial**

Utilizamos los métodos KNN, AdaBoost, DecissionTree, SVM y RandomForest, Naive-Bayes combinados con LeaveOneOut y Features Selection (este último solamente es posible en Random Forest y Adaboost).

En las pruebas de IA utilizamos los métodos de Perceptrón y Red Neuronal Simple, aunque ambos son descartados ya que el Preceptron arroja peor resultado que los métodos de Machine Learning y la Red Neuronal nos da resultados muy dispares y cambiantes.

Cabe destacar que de las posibles métricas que arrojan los métodos de Machine Learning escogemos la métrica Accuracy, que es la que más nos interesa para nuestro modelo. Cada clasificador nos proporciona el accuracy por equipo, posteriormente, mediante un bucle, generamos los datos en forma de diccionario y calculamos la media de todos los equipos por lo que efectuamos una comparativa en función de la precisión media por equipo. **(*Código adjunto en el Anexo).**

RESULTADOS Y EXPERIMENTACIÓN

Tras realizar las primeras pruebas de Machine Learning mendiante **Regresión Múltiple y Polinomial** con nuestra **variable numérica** (inicialmente era así) y ver los bajos resultados que arrojaban la mayoría de clasificadores (**entre un 12 y un 15% de precisión**), decidimos volver al principio y replantear cada parte del código. Primero, comprobamos los resultados de la tabla de correlaciones respecto a nuestra variable "CF":

	P	CA	T	TaP	C	CP	FR	FC	PI	BR	Reg	PA	P%,equipo	CF
P	1.000000	0.432615	-0.193568	-0.128866	-0.161747	-0.111789	-0.203732	-0.238933	-0.195456	-0.208216	-0.160751	-0.169049	NaN	-0.137863
CA	0.432615	1.000000	-0.118241	-0.078942	-0.099084	-0.068481	-0.121898	-0.146374	-0.124395	-0.128543	-0.097667	-0.101887	NaN	-0.084692
T	-0.193568	-0.118241	1.000000	0.706801	0.141222	0.097431	0.174820	0.102712	-0.121361	-0.034448	0.263195	0.278838	NaN	0.328370
TaP	-0.128866	-0.078942	0.706801	1.000000	0.068854	0.039809	0.115544	0.056988	-0.123328	-0.067042	0.202321	0.196615	NaN	0.253655
C	-0.161747	-0.099084	0.141222	0.068854	1.000000	0.749172	0.121115	0.019261	-0.006873	0.078390	0.242660	0.470318	NaN	0.426939
CP	-0.111789	-0.068481	0.097431	0.039809	0.749172	1.000000	0.081869	0.006868	0.001596	0.050914	0.177967	0.551681	NaN	0.249957
FR	-0.203732	-0.121898	0.174820	0.115544	0.121115	0.081869	1.000000	0.141309	0.011835	0.098223	0.215165	0.137125	NaN	0.115475
FC	-0.238933	-0.146374	0.102712	0.056988	0.019261	0.006868	0.141309	1.000000	0.060091	0.146324	0.055705	0.035656	NaN	0.040422
PI	-0.195456	-0.124395	-0.121361	-0.123328	-0.006873	0.001596	0.011835	0.060091	1.000000	0.217620	-0.050150	-0.048360	NaN	-0.051594
BR	-0.208216	-0.128543	-0.034448	-0.067042	0.078390	0.050914	0.098223	0.146324	0.217620	1.000000	0.057838	0.027364	NaN	0.019294
Reg	-0.160751	-0.097667	0.263195	0.202321	0.242660	0.177967	0.215165	0.055705	-0.050150	0.057838	1.000000	0.259824	NaN	0.239687
PA	-0.169049	-0.101887	0.278838	0.196615	0.470318	0.551681	0.137125	0.035656	-0.048360	0.027364	0.259824	1.000000	NaN	0.246252
P%,equipo	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
CF	-0.137863	-0.084692	0.328370	0.253655	0.426939	0.249957	0.115475	0.040422	-0.051594	0.019294	0.239687	0.246252	NaN	1.000000

Decidimos eliminar las columnas con menor nivel de correlación con la variable, estas son P y CA (paradas y centros atajados en el caso de los porteros), FC, PI, y BR (faltas cometidas, pases interceptados y balones robados respectivamente). Tras esto, los resultados mejoraron (entorno al **22% de precisión**), aunque seguían siendo insuficientes.

En ese momento decidimos convertir la variable (inicialmente numérica) a categórica, lo cual no sólo aumentó la precisión del modelo, si no que abrió la posibilidad a usar otros tipos de clasificadores más precisos que los usados inicialmente. Aquí fue cuando definimos los rangos en los que se dividiría la variable. Para este momento la precisión de los clasificadores estaba entre el **33% y el 42%.**

Finalmente, teníamos pensado volver a aplicar el método de las correlaciones que usamos en el primer dataset a nuestro dataset vectorial,

tras consultarlo con el tutor, nos recomendó usar Feature Selection en lugar del método de las correlaciones, además del método de validación Leave One Out. Esto nos arrojó los resultados finales:

Método	Accuracy Media
AdaBoost	43,75%
Random Forest	51,81%
SVM	44,00%
KNN	45,00%
Decission Tree	42,00%
Naive-Bayes	42,92%

El método que mejor resultado nos otorga es el Random Forest, que puede acertar en 1 de cada 2 partidos.

Posteriormente, también analizamos si había algún tipo de tendencia temporal en una tabla que muestra los aciertos del modelo por jornada y equipo:

Jornada	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
Nº Aciertos	10	10	10	5	9	9	10	7	12	12	10	10	10	11	11	7	10	10	8	7	7	10	8	8	11	15	12	10	13	11	8	11	10	7	9	12

Aunque es poco significativa, se puede ver como los datos tienen mayor acierto en las jornadas anteriores al parón invernal y a los últimos partidos, donde los equipos ya han hecho el "*rodaje*" y son más regulares.

ANÁLISIS DEL PROCESO Y CONCLUSIONES

Durante el desarrollo del trabajo nos encontramos con diversas dificultades que pudimos ir solventando:

- El proyecto nació con la idea de ser un análisis multi-variable (corners, nº de faltas, tarjetas, % posesión…). Tras ver los bajos resultados y la diferencia de la importancia de unas y otras columnas en los diferentes targets, decidimos centrar en estudio en una sola variable para reducir el ruido que causaba el exceso de datos. Sin embargo, la gran cantidad de datos del dataset inicial, permite ampliar el proyecto más adelante para conseguir la idea inicial de predecir múltiples variables.
- La conversión de un dataset normal a uno en formato vectorial es algo que no habíamos hecho anteriormente y tuvimos que aplicar un método novedoso y desconocido para nosotros. Teóricamente en un concepto bastante abstracto y complicado de entender, sin embargo, una vez llevado a la práctica, permite tener en cuenta el factor tiempo incluso en series de datos con bajas iteraciones temporales.

Los modelos considerados buenos, tienen una métrica cercano o superior al 90% (según el campo de estudio), sin embargo, cuando se trata de predecir eventos que aún no han sucedido y que dependen de muchas variables (algunas imprevisibles), un modelo con un 50% de precisión, en manos de un experto, puede suponer una ayuda enorme a la hora de acertar en una apuesta.

En el mundo del fútbol, el análisis de los datos, está extendido en dos aspectos, el informativo (periodismo principalmente), y el físico (entrenamientos, predicción de lesiones, control de la forma física, etc.). Sin embargo, está poco extendido en la táctica, lo cual puede ser una gran oportunidad de negocio.

Este proyecto podría ser el punto de partida para otros proyectos como la creación de una **app de recomendación de apuestas**. También, se podría modificar para realizar ratings de jugadores y **analizar los mercados de fichajes** en busca de mejores jugadores, encontrar jugadores cuyo precio de mercado esté por debajo de su precio real o evaluar el potencial y localizar a **jóvenes promesas** que puedan ser futuras estrellas.

ANEXO: CÓDIGO

I. Código del WebScrapping usado con BeautifulSoup:

1ª Parte:

```
#1 Importar las librerías necesarias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import requests
from bs4 import BeautifulSoup
```

In [ ]:

```
#2 Obtener las urls de los partidos en forma de lista
res = requests.get("http://www.futbolfantasy.com/laliga/calendario/2019")
soup = BeautifulSoup(res.content,'lxml')
listado_urls = []
for a in soup.find_all('a', href=True):
    listado_urls.append(a['href'])

# En listado_urls aparece un listado de todas las urls de la pagina, creamos un
# listado nuevo manualmente con las urls que nos interesan: listado_partidos
```

In [ ]:

```
# Se realiza otra lista con el nombre de los equipos que jugan cada partido
# y los nombres de los equipos con un guión en medio '-' por '_'
# para que al hacer split por salga el nombre completo
listado_equipos = ['girona-valladolid',
'betis-levante',
'celta-espanyol',
'villarreal-real_sociedad',
'barcelona-alaves',
'eibar-huesca',
'rayo-sevilla',
    ......]
```

2ª Parte:

```
for i in range(len(listado_equipos)):
    listado_equipos[i] = listado_equipos[i].split('-')
```

In [12]:

```
# Obtenemos los datos de jugador de todos los partidos de la temporada
data = []
n = 1
y = 0
for i in listado_partidos:

    res = requests.get(i)
    soup = BeautifulSoup(res.content,'lxml')

    table = soup.find_all('table')[0]
    df = pd.read_html(str(table))[0]
    df = df[:11]
    df['partido'] = n
    df['equipo'] = listado_equipos[y][0]
    df['localizacion'] = 'local'
    data.append(df)

    table = soup.find_all('table')[1]
    df = pd.read_html(str(table))[0]
    df = df[:11]
    df['partido'] = n
    df['equipo'] = listado_equipos[y][1]
    df['localizacion'] = 'visitante'
    data.append(df)
    n+=1
    y+=1


dfdata = pd.concat(data)
```

In [14]:

```
# Guardamos este dataframe en un archivo csv
dfdata.to_csv('C:/Users/eduds/Desktop/laliga_limpio/jugadores.csv',ind
ex=False)
```

II. Transformación del dataset en un dataset vectorial:

```
#Donde E sería el nombre del equipo al que queremos transformar en Dat
aset vectorial

def Team(E):
    partidos = df.groupby('equipo').get_group(E)
    x = 1 #jornada a empezar
    lista = []
    for i in range(36): #numero de filas
        datos_a = pd.DataFrame(np.concatenate(np.asarray(partidos[part
idos.jornada == i+x].iloc[0:12,0:7]))).T
        datos_a = datos_a.reset_index()
        corners_a = partidos[partidos.jornada == i+x].iloc[0:12,-1].su
m()#suma de los corners
        datos_a.insert(6,'corners', corners_a)


        datos_b = pd.DataFrame(np.concatenate(np.asarray(partidos[part
idos.jornada == i+x+1].iloc[0:12,0:7]))).T
        datos_b = datos_b.reset_index()
        corners_b = partidos[partidos.jornada == i+x+1].iloc[0:12,-1].
sum()#suma de los corners
        datos_b.insert(6,'corners', corners_b)

        jugadores_c = pd.DataFrame(np.concatenate(np.asarray(partidos[
partidos.jornada == i+x+2].iloc[0:12,0:1]))).T
        corners_c = partidos[partidos.jornada == i+x+2].iloc[0:12,-1].
sum()
        jugadores_c.insert(11,'corners', corners_c)

        total_v1 = pd.concat([datos_a,datos_b,jugadores_c],axis=1,igno
re_index=True)
        total_v1 = total_v1.drop([79,86,87,88,89,91],axis=1)
        lista.append(total_v1)

    df_final = pd.concat(lista)
    df_final = df_final.reset_index()

    df_final = df_final.drop(0,axis=1)
    df_final = df_final.drop('index',axis=1)
    df_final.replace(np.nan,0)
    # df_final = df_final.replace(['local','visitante'],[0,1])
    return df_final
```

III. Aplicación del Machine Learning:

Veremos un ejemplo (Random Forest) de los muchos clasificadores aplicados.

```
# 1 - Importamos las librerías necesarias:

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV
```

```
# 2 - Aplicamos GridSearch para encontrar los parámetros más
#eficientes para nuestro clasificador:
df_final = Team('betis')
```

In [9]:

```
df_final[169] = df_final[169].replace((0,1,2,3),0)
df_final[169] = df_final[169].replace((4,5,6),1)
df_final[169] = df_final[169].replace((7,8,9,10,11,12,13),2)
```

In [16]:

```
clf = RandomForestClassifier()
X = df_final.iloc[:,:162]
y = df_final.iloc[:,162:163]
```

In [17]:

```
params={'criterion':['gini', 'entropy'],
        'max_depth': [2,4,8],# Maxima pofundidad del arbol
        'max_features': [2,3,5], # numero de features a considerar en
cada split
        'max_leaf_nodes': [3,5,8], # maximo de nodos del arbol
        'min_impurity_decrease' : [0.02], # un nuevo nodo se hará si a
l hacerse se decrece la impureza
        'min_samples_split': [3], # The minimum number of samples requ
ired to split an internal node
        'n_estimators':[1, 2, 5, 10,20]
        }
```

In [19]:

```
X = np.array(X)
y = np.array(y)
from sklearn.model_selection import GridSearchCV
grid_solver = GridSearchCV(estimator = clf, # model to train
                   param_grid = params, # param_grid
                   scoring = make_scorer(f1_score, average ="macro"),
                   cv = 5)
model_result = grid_solver.fit(X,y)
```

- Los seleccionados como mejores parámetros fueron:

```
model_result.best_params_
```

Out[22]:

```
{'criterion': 'entropy',
 'max_depth': 4,
 'max_features': 5,
 'max_leaf_nodes': 8,
 'min_impurity_decrease': 0.02,
 'min_samples_split': 3,
 'n_estimators': 2}
```

- Por último, Definimos la función que nos proporcionaría el accuracy final:

```
# Definimos una segunda función, que en función de la 1ª (que nos daba
# el dataset vectorial por equipo),
# nos calcule el Accuracy para el equipo 'X'. Esta función aplica
# Leave One Out y Feature Selection automáticamente.

def RFTeam(X):
    #df_final=X
    df_final = Team(X)
    df_final = df_final.replace(np.nan, 0.0)
    df_final[169] = df_final[169].replace((0,1,2,3),0)
    df_final[169] = df_final[169].replace((4,5,6),1)
    df_final[169] = df_final[169].replace((7,8,9,10,11,12,13),2)
    X = df_final.iloc[:,:162]
    y = df_final.iloc[:,162:163]
    X = np.array(X)
    y = np.array(y)
    #y = y.replace(np.nan,0.0)
    y = y[:, 0]
    # Build a forest and compute the feature importances
    forest = RandomForestClassifier(n_estimators=100, criterion="entro
py", max_depth=4, min_samples_split=3,
                                    max_features=2, max_leaf_nodes=8,
min_impurity_decrease=0.02)
 #Change for R.Forest
 #INICIO DEL FEATURE SELECTION -
    forest.fit(X, y)
    importances = forest.feature_importances_
    std = np.std([tree.feature_importances_ for tree in forest.estimat
ors_], axis=0)
    indices = np.argsort(importances)[::-1]

    # Print the feature ranking
    #print("Feature ranking:")

    #for f in range(X.shape[1]):
      #  print("%d. feature %d (%f)" % (f + 1, indices[f], importances
[indices[f]]))
    X = df_final.iloc[:,indices[:50]]
    X=X.replace(np.nan,0)
    X = np.array(X)
 # - FINAL DEL FEATURE SELECTION (estas líneas de código sólo se aplic
an en R. Forest y Adaboost).
```

```
    loo = LeaveOneOut()
    loo.get_n_splits(X)
    yhat=[]
    for train_index, test_index in loo.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        #CLASIFICADOR Random Forest
        clf = RandomForestClassifier(n_estimators=100, criterion="entr
opy", max_depth=4, min_samples_split=3, max_features=2, max_leaf_nodes
=8, min_impurity_decrease=0.02)
        clf.fit(X_train, y_train)  ##X_train, y_train
        yhat1 = clf.predict(X_test)
        yhat.append(yhat1)

    #print("Accuracy: ", accuracy_score(y,yhat))
    return accuracy_score(y,yhat)
```

- Esta función proporciona el accuracy para un equipo, sin embargo, definiendo una listacon el nombre de cada equipo ('list_equip'), podemos realizar un bucle que nos proporcione en forma de diccionario el valor del Accuracy de cada equipo:

```
L_Acc = []
for i in list_equip:
    L_Acc.append(RFTeam(i))
Dicc = dict(zip(list_equip,L_Acc))
Dicc
```

Out[13]:

```
{'girona': 0.6666666666666666,
 'valladolid': 0.3611111111111111,
 'betis': 0.4722222222222222,
 'levante': 0.3888888888888889,
 'celta': 0.6666666666666666,
 'espanyol': 0.6388888888888888,
 'villarreal': 0.5,
 'real_sociedad': 0.4722222222222222,
 'barcelona': 0.4444444444444444,
 'alaves': 0.6111111111111112,
 'eibar': 0.3888888888888889,
 'huesca': 0.5555555555555556,
 'rayo': 0.7222222222222222,
 'sevilla': 0.5555555555555556,
 'real_madrid': 0.4166666666666667,
 'getafe': 0.4166666666666667,
 'valencia': 0.4444444444444444,
 'atletico': 0.3333333333333333,
 'athletic': 0.6944444444444444,
 'leganes': 0.6111111111111112}
print('Media_Random_Forest = ', sum(L_Acc)/20)
0.518055555555555
```