NBA Data WareHouse

Bases de Datos Avanzadas
Escuela Superior Informática (UCLM)
Mario Pérez Sánchez-Montañez, David Camuñas Sánchez, Francesco Zingariello
27 de febrero de 2020

Resumen

Introducción

Un almacén de datos o *Data Warehouse* es el instrumento que las organizaciones encontraron centralizar toda la información relevante, el objetivo de tomar decisiones informadas con más facilidad. Es decir, facilitar el almacenaje y análisis de los datos de la empresa.

En nuestro caso, hemos construido un almacén de datos para almacenar las estadísticas mas relevantes de la temporada 2018-2019 de la competición de baloncesto más popular, la NBA [4]. El trabajo consiste en la extracción de datos de la competición disponibles en [11] y almacenaje de estos datos en una base datos relacional para su posterior estudio.

Para el análisis y la extracción de datos, se llevará a cabo el proyecto con *Python* [16] y como motor de base de datos se utilizará *SQLite* [17]. Nuestra base de datos estará formada por cuatro tablas, que se describirán posteriormente.

Obtención de los datos

Los datos se han obtenido aplicando técnicas de *web scraping* [6] gracias a la librería *Beautiful Soup* [14] de *Python*. Con esta librería se obtiene el código HTML de una pagina web y se pueden realizar diversas operaciones con esa información, en nuestro caso hemos construido un *dataframe* [9] para poder construir posteriormente la base de datos.

Datos de jugadores

Los datos relacionados con los jugadores se han obtenido gracias al siguiente código [10] en *Python*:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import pandas as pd


def get_player_stats(year):
    # URL page we will scraping (see image above)
    url = "https://www.basketball-reference.com/" \
          "leagues/NBA_{}_totals.html".format(year)
    # this is the HTML from the given URL
    html = urlopen(url)
    soup = BeautifulSoup(html, features="html.parser")

    # use findALL() to get the column headers
    soup.findAll('tr', limit=2)
    # use getText()to extract the text we need into a list
    headers = [th.getText() for th in
               soup.findAll('tr', limit=2)[0].findAll('th')]
    # exclude the first column as we will not need the
    # ranking order from Basketball Reference for the analysis
    headers = headers[1:]

    # avoid the first header row
    rows = soup.findAll('tr')[1:]

    player_stats = [[td.getText() for td
                     in rows[i].findAll('td')]
                    for i in range(len(rows))]

    # Create DataFrame
    return pd.DataFrame(player_stats, columns=headers)


def get_rookie_stats(year):
    # URL page we will scraping (see image above)
    url = "https://www.basketball-reference.com/" \
          "leagues/NBA_{}_rookies-season-stats.html".format(year)
    # this is the HTML from the given URL
    html = urlopen(url)
    soup = BeautifulSoup(html, features="html.parser")

    # use findALL() to get the column headers
    soup.findAll('tr', limit=2)
```

```
    # use getText()to extract the text we need into a list
    headers = [th.getText() for th in
               soup.findAll('tr', limit=2)[1].findAll('th')]
    # the first row is not the header
    # exclude the first column as we will not need the
    # ranking order from Basketball Reference for the analysis
    headers = headers[1:]

    # avoid the first header row
    rows = soup.findAll('tr')[1:]

    rookie_stats = [[td.getText() for td
                     in rows[i].findAll('td')]
                    for i in range(len(rows))]

    return pd.DataFrame(rookie_stats, columns=headers)
```

Como se puede ver, el programa esta formado por dos funciones. La primera para obtener las estadísticas globales de todos los jugadores [13] y con la segundo se obtienen los datos relativos a los *rookies* (jugadores nóveles) [12]. Ambas funciones devolverán un objeto *dataframe* [9], porque en el programa principal se llamará a estas funciones para construir la base de datos, como se ha comentado anteriormente. Para el tratamiento de los datos en forma de *dataframe* utilizaremos la librería *pandas* [8].

Para obtener los datos que nos interesan de la web es necesario que sepamos cuáles son las etiquetas HTML que debemos de obtener del objeto *soup*. Esto normalmente se realiza inspeccionando la página web con cualquier navegador.

Datos de equipos

Los datos relativos a los equipos de la *nba* se han obtenido de forma similar a los datos de los jugadores:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import pandas as pd


def get_team_stats(year):
    # URL page we will scraping (see image above)
    url = "https://www.basketball-reference.com/" \
          "leagues/NBA_{}_ratings.html".format(year)
    # this is the HTML from the given URL
    html = urlopen(url)
    soup = BeautifulSoup(html, features="html.parser")

    # use findALL() to get the column headers
```

```
    soup.findAll('tr', limit=2)
    # use getText()to extract the text we need into a list
    headers = [th.getText() for th in
               soup.findAll('tr', limit=2)[1].findAll('th')]
    # the first row is not the header
    # exclude the first column as we will not need the ranking
    # order from Basketball Reference for the analysis
    headers = headers[1:]

    # avoid the first header row
    rows = soup.findAll('tr')[1:]

    team_stats = [[td.getText() for td in rows[i].findAll('td')]
                  for i in range(len(rows))]

    team_df = pd.DataFrame(team_stats, columns=headers)

    return team_df.sort_values('Team')


def get_entire_team_name():
    # URL page we will scraping (see image above)
    url = "https://en.wikipedia.org/wiki/Wikipedia:" \
          "WikiProject_National_Basketball_Association/" \
          "National_Basketball_Association_team_abbreviations "
    # this is the HTML from the given URL
    html = urlopen(url)
    soup = BeautifulSoup(html, features="html.parser")

    # avoid the first header row
    rows = soup.findAll('tr')[1:]

    team_names = [[td.getText().replace('\n', '') for
                   td in rows[i].findAll('td')]
                  for i in range(len(rows))]

    team_names_df = pd.DataFrame(team_names,
                                 columns=['Key', 'Name'])
    return team_names_df.sort_values('Name')
```

La función para obtener los datos de los equipos es muy similar a las funciones mencionadas anteriormente, con la diferencia que el *dataframe* que retornamos lo ordenamos por el nombre del equipo. La segunda función en un principio no la desarrollamos, pero en la construcción de la base de datos apreciamos que los datos de los jugadores indican el equipo mediante abreviaturas. Por lo tanto esta función la utilizaremos para establecer las equivalencias.

Creación de la base de datos

La creación de la base de datos la hemos automatizado gracias al siguiente *script* que llama a las funciones explicadas en la sección anterior:

```
from team_stats import *
from player_stats import *
import pandas as pd
import sqlite3 as sqlite
import os

players_df, teams_df = None, None
rookies_df, central_df = None, None

def load_data():
    global players_df, teams_df, rookies_df
    # Players DataFrame
    players_df = get_player_stats(2019).reset_index()
    players_df = players_df.rename(columns={'index': 'id'})

    # Teams Abbreviation DataFrame
    teams_abbreviation_df = get_entire_team_name()
    teams_abbreviation_df.at[23,'Key']= 'PHO'
    teams_abbreviation_df.at[1,'Key']= 'BOS'
    teams_abbreviation_df.at[2,'Key']= 'BRK'
    teams_abbreviation_df.at[3,'Key']= 'CHO'



    # Teams DataFrame
    teams_df = get_team_stats(2019).reset_index()
    teams_df = teams_df.rename(columns={'index': 'id'})
    teams_df.insert(1, "Key", teams_abbreviation_df['Key'], False)

    # Rookie DataFrame
    rookies_df = get_rookie_stats(2019).reset_index()
    rookies_df = rookies_df.rename(columns={'index': 'id'})


def clean_data():
    global players_df, teams_df, rookies_df
    # Remove columns that we don't use
    players_df = players_df.drop('eFG%', 1)
    teams_df = teams_df.drop(
        ['Div', 'MOV/A', 'ORtg/A', 'DRtg/A', 'NRtg/A'], 1)
    rookies_df = rookies_df.drop(
        ['Yrs', 'MP', 'PTS', 'TRB', 'AST'], 1)

    # Remove NaN rows
```

```
    players_df = players_df.dropna()
    teams_df = teams_df.dropna()
    rookies_df = rookies_df.dropna()

    # Remove players with invalid team
    for i, row in players_df.iterrows():
        if row['Tm'] == 'TOT':
            continue

        if row['Tm'] not in list(teams_df['Key']):
            players_df = players_df.drop(index=i)

    # Reboot index, because it need start with 0
    rookies_df = rookies_df.reset_index().drop('index', 1)



def build_central_df():
    global players_df, teams_df, rookies_df, central_df
    central_df = pd.DataFrame(
        columns=['id_player', 'id_team',
        'Player', 'Team', 'Rookie'])

    # build id_player
    central_df['id_player'] = players_df['id']
    # build player names
    central_df['Player'] = players_df['Player']

    for i, row in central_df.iterrows():
        # build id team
        if players_df.loc[i]['Tm'] == 'TOT':
            # id for total statistics, be cause he
            # played in more one team
            row['id_team'] = -1
        else:
            row['id_team'] = int(teams_df.loc[list(
                teams_df['Key']).index(
                    players_df.loc[i]['Tm'])]['id'])

        # build team name
        if row['id_team'] == -1:
            row['Team'] = 'Total'
        else:
            row['Team'] = teams_df.loc[list(
                teams_df['id']).index(row['id_team'])]['Team']

        # build is rookie
        if row['Player'] in list(rookies_df['Player']):
```

```
                row['Rookie'] = int(rookies_df.loc[list(
                    rookies_df['Player']).index(row['Player'])]['id'])

            central_df.loc[i] = row

        # Remove team column, because team info will be in the central
        players_df = players_df.drop('Tm', 1)

    def build_database():
        global players_df, teams_df, rookies_df, central_df

        sql_data = 'nba_data.db'
        connection = sqlite.connect(sql_data)

        # load db creation script
        file = open('creation.sql', 'r')
        query = ''
        for line in file:
            query += line

        # db tables creation
        connection.cursor().executescript(query)

        # Load players_df in db
        players_df.to_sql('PLAYERS', connection,
            if_exists='append', index=False)

        # Load teams_df in db
        teams_df.to_sql('TEAMS', connection,
            if_exists='append', index=False)

        # Load rookies_df in db
        rookies_df.to_sql('ROOKIES', connection,
            if_exists='append', index=False)

        # Load central_df in db
        central_df.to_sql('PLAYER_TEAM', connection,
            if_exists='append', index=False)

        # Close the connection with the db
        connection.close()


    if __name__ == '__main__':
        if os.path.isfile('nba_data.db'):
            print('The db file exists, remove to build newly')
        else:
            load_data()
```

```
clean_data()
build_central_df()
build_database()
```

El programa lleva a cabo las siguientes operaciones:

- **Cargar los datos:** se llama a las funciones explicadas anteriormente para cargar la información relativas a jugadores, *rookies* y equipos. Además, en esta parte del programa también se establece la equivalencia de los equipos con sus abreviaturas.
- **Transformación de datos:** se realiza una limpieza de los datos, que consiste en eliminar las columnas que no vamos a utilizar, eliminar las filas vacías y eliminación de jugadores que no juegan en un equipo de la nba.
- **Construcción del *dataframe* central**: estos datos se usarán para construir una tabla central que relacione los datos de jugadores, *rookies* y equipos, siguiendo el modelo de estrella [2].
- **Construcción de la base datos**: en este último paso se construirán las cuatro tablas de la base datos y se introducirán los datos guardados en los cuatro *dataframes* que hemos comentado. Las operaciones que se ejecutan para crear las tabla y sus respectivas relaciones se pueden ver en el fichero ***creation.sql***.

Diagrama relacional

El diagrama relacional de nuestra base de datos, estará formado por cuatro tablas relacionadas en forma de estrella, de las cuales tres de ellas las crearemos a partir de los datos obtenidos en el punto anterior (*Equipos*, *Jugadores* y *Rookies*). La cuarta tabla (*Jugadores-Equipos*) es una tabla central, la cual está relacionada con cada una de las anteriores tablas.

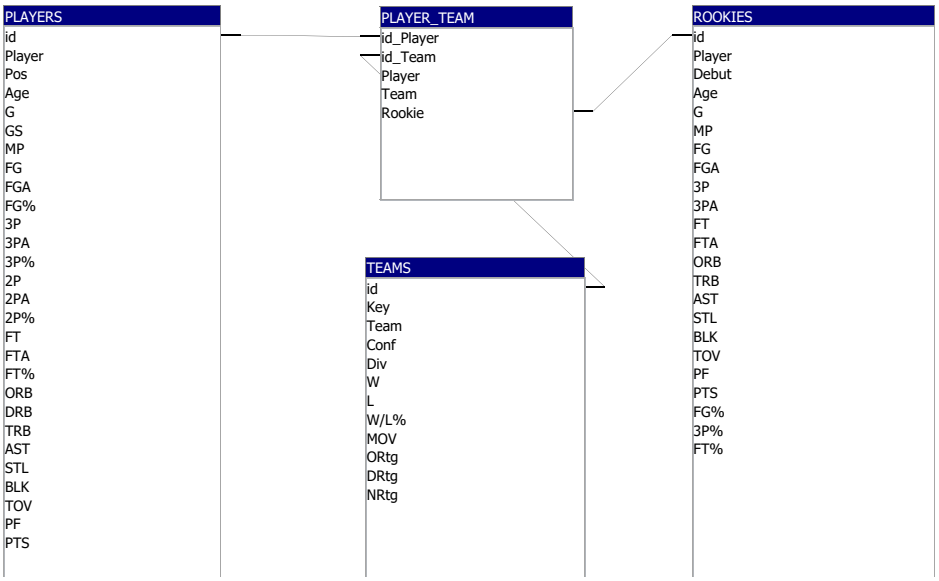

Figura 1: Diagrama relacional de nuestra base de datos

A continuación, se explicara la definición de cada parametro por los que está formada cada tabla.

La tabla **Jugadores** (PLayers), esta formada por:

- **id**: Player id.
- **Player**: Player full name.
- **Pos**: Position.
- **Age**: Player age.
- **G**: Games.
- **GS**: Games started
- **MP**: Minutes played.
- **FG (3P + 2P)**: Field goals.
- **FGA (3PA + 2PA)**: Field goal attempts.
- **FG %**: Field goal percentage.
- **3P**: 3 points field goals.
- **3PA**: 3 point field goal attempts.
- **3P %**: FG % on 3 points FGA's (field goal attempts).
- **2P**: 2 points field goals.
- **2PA**: 2 point field goal attempts.
- **2P %**: FG % on 2 points FGA's (field goal attempts).

Modelo de regresión lineal

El modelo construido está basado en la regresión lineal simple de dos variables [5], los puntos totales que anota un jugador con sus intentos de tiro. El objetivo con este modelo será predecir cuántos puntos anotaría un jugador indicando los intentos de tiro.

Los modelos predictivos o de regresión consisten en la representación de la relación entre dos (o más) variables a través de un modelo formal supone contar con una expresión lógico-matemática que resume cómo es esa relación. Además, este modelo permite realizar predicciones de los valores que tomará una de las dos variables (la que se asuma como variable de respuesta o dependiente, y) a partir de los valores de la otra (la que se asuma como variable explicativa, independiente o predictora, x) [1].

Cálculo del modelo

En nuestro caso, como vamos a realizar una regresión lineal simple la formula sería la siguiente:

$$y = \alpha + \beta * x \tag{1}$$

La variable que queremos predecir (y) serán los puntos que anotará un jugador y para ello utilizaremos la variable explicativa de los tiros a canasta de un

jugador (x), contando los tiros libres, los tiros de dos puntos y los triples.

$$PTS = \alpha + \beta * TA \tag{2}$$

Los parámetros utilizados en la predicción se van a calcular utilizando la librería *statsmodels* [18] de Python. Los pasos seguidos para calcular este modelo se pueden ver con detalle en el *notebook* de *Jupyter* [7] que hemos creado. La fórmula resultante con los parámetros calculados en base a los datos almacenados en nuestro *data warehouse* es la siguiente:

$$PTS = -5{,}209429 + 1{,}000137 * TA \tag{3}$$

Al dibujar la recta de regresión [15] podemos observar como de bueno es el modelo creado para predecir los puntos que anotará un jugador introduciendo los intentos a canasta (ver Figura 2). Cuanto menos dispersos y más cercanos a la recta estén los puntos mejor será nuestro modelo.

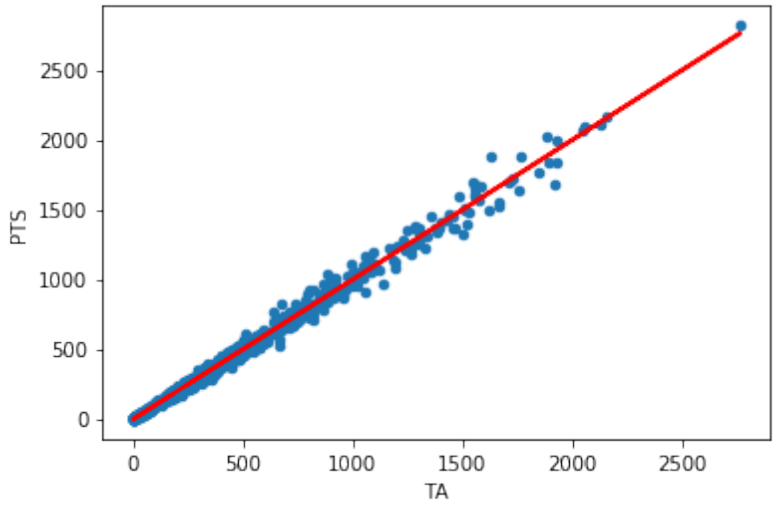

Figura 2: Recta de regresión de PTS TA

Realizar predicción

En este punto, se va a realizar una serie de predicciones utilizando la expresión calculada anteriormente. En el Cuadro 1 se pueden ver cinco jugadores con los puntos reales anotados en la temporada y los puntos que predice nuestro modelo. Se puede apreciar que la predicción se asemeja bastante a los puntos que anota un jugador.

Cabe destacar que los parámetros han sido calculados con los mismos datos que luego hemos usado para predecir, por lo tanto, se puede dar el fenómeno de la multicolinealidad [3]. Esto sucede cuando el modelo se ajusta demasiado a un conjunto de datos y cuando utilizamos otros datos para hacer

Player	PTS	PTS_pred
Álex Abrines	165	164.813847
Quincy Acy	17.0	22.794404
Jaylen Adams	108.0	113.806864
Steven Adams	1108.0	1095.941316
Bam Adebayo	729.0	706.888055

Cuadro 1: Predicciones del modelo

una predicción el modelo falla.

En nuestro caso, al usar datos de una temporada de una competición deportiva se presupone que los datos no van a ser muy desiguales de una temporada a otra. Con lo cual, nuestro modelo se debe comportar correctamente cuando introducimos un dato que no forma parte del conjunto utilizado para crear el modelo.

Referencias

1. http://ocw.uv.es/ciencias-de-la-salud/pruebas-1/1-3/t_09nuevo.pdf. Modelo de regresión lineal.
2. https://es.wikipedia.org/wiki/Esquema_en_estrella. Esquema en estrella.
3. https://es.wikipedia.org/wiki/Multicolinealidad. Multicolinealidad.
4. https://es.wikipedia.org/wiki/National_Basketball_Association. National basketball association.
5. https://es.wikipedia.org/wiki/RegresiÃşn_lineal. Regresión lineal.
6. https://es.wikipedia.org/wiki/Web_scraping. Web scraping.
7. https://jupyter.org/. Jupyter.
8. https://pandas.pydata.org/. Pandas.
9. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html. Pandas dataframe.
10. https://towardsdatascience.com/web-scraping-nba-stats-4b4f8c525994. Web scraping nba stats.
11. https://www.basketball-reference.com. Basketball reference.
12. https://www.basketball-reference.com/leagues/NBA_2019_rookies-season-stats.html. Rookies stats.
13. https://www.basketball-reference.com/leagues/NBA_2019_totals.html. Player stats.
14. https://www.crummy.com/software/BeautifulSoup/bs4/doc/. Beautiful soup.
15. https://www.hiru.eus/es/matematicas/recta-de-regresion-y-correlaciones. Recta de regresión.
16. https://www.python.org/. Python software fundation.
17. https://www.sqlite.org/index.html. Sqlite.
18. https://www.statsmodels.org/stable/index.html. Statsmodels.