ML_7_KNN

March 24, 2019

In [3]:
```
# Read the dataset

import numpy as np
import pandas as pd
with open("nba_2013.csv", "r") as csvfile:
    nba_raw = pd.read_csv(csvfile)
nba_raw.head()
```

Out[3]:

	player	pos	age	bref_team_id	g	gs	mp	fg	fga	fg.	...	drb	trb	ast	stl	blk	tov	pf	pts	season	season_end
0	Quincy Acy	SF	23	TOT	63	0	847	66	141	0.468	...	144	216	28	23	26	30	122	171	2013-2014	2013
1	Steven Adams	C	20	OKC	81	20	1197	93	185	0.503	...	190	332	43	40	57	71	203	265	2013-2014	2013
2	Jeff Adrien	PF	27	TOT	53	12	961	143	275	0.520	...	204	306	38	24	36	39	108	362	2013-2014	2013
3	Arron Afflalo	SG	28	ORL	73	73	2552	464	1011	0.459	...	230	262	248	35	3	146	136	1330	2013-2014	2013
4	Alexis Ajinca	C	25	NOP	56	30	951	136	249	0.546	...	183	277	40	23	46	63	187	328	2013-2014	2013

[5 rows x 31 columns]

In [4]:
```
# See the columns
print(nba_raw.columns.values)
```

```
['player' 'pos' 'age' 'bref_team_id' 'g' 'gs' 'mp' 'fg' 'fga' 'fg.' 'x3p'
 'x3pa' 'x3p.' 'x2p' 'x2pa' 'x2p.' 'efg.' 'ft' 'fta' 'ft.' 'orb' 'drb'
 'trb' 'ast' 'stl' 'blk' 'tov' 'pf' 'pts' 'season' 'season_end']
```

In [7]:
```
# Information about the data

nba_raw.describe()
```

Out[7]:

	age	g	gs	mp	fg	...
count	481.000000	481.000000	481.000000	481.000000	481.000000	

```
mean    26.509356   53.253638   25.571726  1237.386694  192.881497
std      4.198265   25.322711   29.658465   897.258840  171.832793
min     19.000000    1.000000    0.000000     1.000000    0.000000
25%     23.000000   32.000000    0.000000   388.000000   47.000000
50%     26.000000   61.000000   10.000000  1141.000000  146.000000
75%     29.000000   76.000000   54.000000  2016.000000  307.000000
max     39.000000   83.000000   82.000000  3122.000000  849.000000

                fga         fg.         x3p         x3pa         x3p.  \
count    481.000000  479.000000  481.000000  481.000000  414.000000
mean     424.463617    0.436436   39.613306  110.130977    0.285111
std      368.850833    0.098672   50.855639  132.751732    0.157633
min        0.000000    0.000000    0.000000    0.000000    0.000000
25%      110.000000    0.400500    0.000000    3.000000    0.234355
50%      332.000000    0.438000   16.000000   48.000000    0.330976
75%      672.000000    0.479500   68.000000  193.000000    0.375000
max     1688.000000    1.000000  261.000000  615.000000    1.000000

           ...          orb         drb         trb         ast  \
count      ...   481.000000  481.000000  481.000000  481.000000
mean       ...    55.810811  162.817048  218.627859  112.536383
std        ...    62.101191  145.348116  200.356507  131.019557
min        ...     0.000000    0.000000    0.000000    0.000000
25%        ...    12.000000   43.000000   55.000000   20.000000
50%        ...    35.000000  135.000000  168.000000   65.000000
75%        ...    73.000000  230.000000  310.000000  152.000000
max        ...   440.000000  783.000000 1114.000000  721.000000

              stl         blk         tov          pf          pts  season_end
count  481.000000  481.000000  481.000000  481.000000   481.000000       481.0
mean    39.280665   24.103950   71.862786  105.869023   516.582121      2013.0
std     34.783590   30.875381   62.701690   71.213627   470.422228         0.0
min      0.000000    0.000000    0.000000    0.000000     0.000000      2013.0
25%      9.000000    4.000000   21.000000   44.000000   115.000000      2013.0
50%     32.000000   14.000000   58.000000  104.000000   401.000000      2013.0
75%     60.000000   32.000000  108.000000  158.000000   821.000000      2013.0
max    191.000000  219.000000  295.000000  273.000000  2593.000000      2013.0

[8 rows x 27 columns]
```

In [10]:
```
nba_raw.isnull().any().any()
```

Out[10]:
```
True
```

In [11]:
```
# We can replace the NaN values with mean of the columns.But if we normalize the data
#the the mean will be 0 and the standard Deviation=1
# Hence we can fill the NaN values with 0
```

```
        # Replace NaN values with zeros.
        nba = nba_raw.fillna(0)

        # Convert strings to NaN and drop.

        nba = nba.convert_objects(convert_numeric=True).dropna()

        nba.head()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: FutureWarning: convert_obj
For all other conversions use the data-type specific converters pd.to_datetime, pd.to_timedelta
  # Remove the CWD from sys.path while we load stuff.
```

Out[11]:

	player	pos	age	bref_team_id	g	gs	mp	fg	fga	fg.	...	drb	trb	ast	stl	blk	tov	pf	pts	season	season_end
0	Quincy Acy	SF	23	TOT	63	0	847	66	141	0.468	...	144	216	28	23	26	30	122	171	2013-2014	2013
1	Steven Adams	C	20	OKC	81	20	1197	93	185	0.503	...	190	332	43	40	57	71	203	265	2013-2014	2013
2	Jeff Adrien	PF	27	TOT	53	12	961	143	275	0.520	...	204	306	38	24	36	39	108	362	2013-2014	2013
3	Arron Afflalo	SG	28	ORL	73	73	2552	464	1011	0.459	...	230	262	248	35	3	146	136	1330	2013-2014	2013
4	Alexis Ajinca	C	25	NOP	56	30	951	136	249	0.546	...	183	277	40	23	46	63	187	328	2013-2014	2013

[5 rows x 31 columns]

In [25]:
```
# Selecting numeric Columns


distance_columns = ['age', 'g', 'gs', 'mp', 'fg', 'fga', 'fg.', 'x3p', 'x3pa',
 'x3p.', 'x2p', 'x2pa', 'x2p.', 'efg.', 'ft', 'fta', 'ft.', 'orb', 'drb', 'trb', 'ast
 'stl', 'blk', 'tov', 'pf', 'pts']

## Normalizing columns

nba_numeric = nba[distance_columns]
nba_numeric.head(5)

nba_normalized = (nba_numeric - nba_numeric.mean()) / nba_numeric.std()
nba_normalized.head(5)
```

Out[25]:

	age	g	gs	mp	fg	fga	fg.
0	-0.835906	0.384886	-0.862207	-0.435088	-0.738401	-0.768505	0.325957
1	-1.550487	1.095711	-0.187863	-0.045011	-0.581271	-0.649215	0.667749
2	0.116868	-0.010016	-0.457600	-0.308035	-0.290291	-0.405214	0.833763

```
3  0.355062  0.779789  1.599148  1.465144  1.577804  1.590172  0.238067
4 -0.359519  0.108454  0.149309 -0.319180 -0.331028 -0.475703  1.087666

        x3p      x3pa      x3p.     ...          ft.       orb       drb  \
0 -0.700282 -0.716608  0.120520     ...    -0.151926  0.260690 -0.129462
1 -0.778936 -0.829601 -1.390497     ...    -0.522588  1.387883  0.187020
2 -0.778936 -0.829601 -1.390497     ...    -0.250457  0.743773  0.283340
3  1.737992  1.430256  1.027130     ...     0.575320 -0.383420  0.462221
4 -0.778936 -0.822068 -1.390497     ...     0.673851  0.614951  0.138859

        trb       ast       stl       blk       tov        pf       pts
0 -0.013116 -0.645220 -0.468056  0.061410 -0.667650  0.226515 -0.734621
1  0.565852 -0.530733  0.020680  1.065446 -0.013760  1.363938 -0.534801
2  0.436083 -0.568895 -0.439307  0.385292 -0.524113  0.029924 -0.328603
3  0.216475  1.033919 -0.123066 -0.683520  1.182380  0.423107  1.729123
4  0.291341 -0.553630 -0.468056  0.709175 -0.141348  1.139262 -0.400878

[5 rows x 26 columns]
```

In [24]:
```
# Finding the nearest neighbor on the basis of Eucledian distance

from scipy.spatial import distance

# Fill in NA values in nba_normalized.

nba_normalized.fillna(0, inplace=True)

# Find the normalized vector for lebron james.

lebron_normalized = nba_normalized[nba["player"] == "LeBron James"]

# Find the distance between lebron james and everyone else.

euclidean_distances = nba_normalized.apply(lambda row: distance.euclidean(row, lebron_
distance_frame = pd.DataFrame(data={"dist": euclidean_distances, "idx": euclidean_dist
distance_frame.sort_values("dist", inplace=True)

second_smallest = distance_frame.iloc[1]["idx"]

most_similar_to_lebron = nba.loc[int(second_smallest)]["player"]
print("most_similar_to_lebron:", most_similar_to_lebron)
```

```
most_similar_to_lebron: Carmelo Anthony
```

In [30]:
```
# Dependent and independent variables

X = nba_numeric.drop(["pts"],axis = 1).values
y = nba_numeric["pts"].values
```

In [32]:
```
# Standard scaling and train_test_split

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.33,random_state =


from sklearn.preprocessing import StandardScaler

scX = StandardScaler()
scX.fit(X_train,X_test)
```

Out[32]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [35]:
```
# Applying KNN

from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)

from sklearn.metrics import r2_score

score = r2_score(y_test,y_pred)
score
```

Out[35]:
```
0.969751393363463
```