Predicting how many goals a striker scores during a season based on their shooting (Stage 2)

September 2022

1 Introduction

Football being the worlds most popular sport [1] the money involved in player transfers are huge. Especially when talking about strikers. That is why it is important to see how a striker performs during a season before a team will decide if they want to buy that striker or not. And the main way to see that is through how many goals he makes. So the goal of this project is to predict the number of goals a striker will score during one season based of the striker's shooting statistics.

Section 2 of this project will go through the formulation of the problem and explain how the problem will be transformed to make use of the different machine learning methods. In section 3 we will explain in more detail about the dataset, how we selected our features and used linear and polynomial regression on our features. Finally we will explain our reasoning on how we split the data used in this project. In section 4 we will compare the different machine learning methods used in this project. And finally summarizing the project and discussing possible improvements will be in section 5.

2 Problem formulation

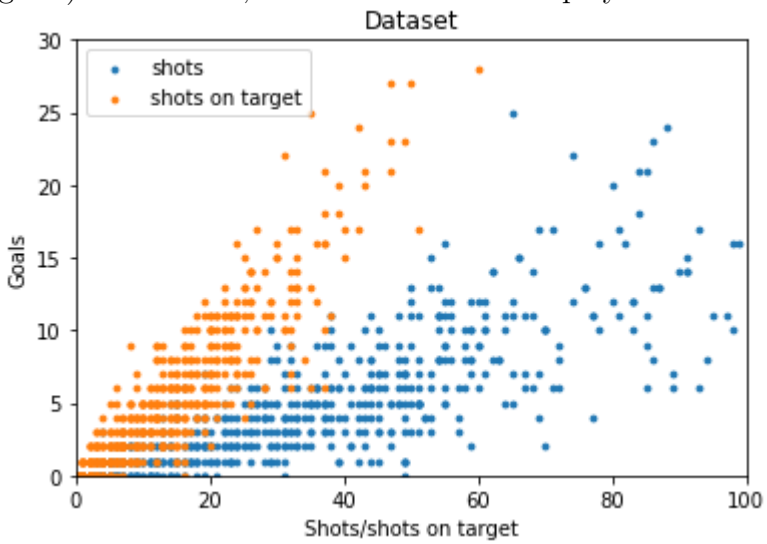
In this project our goal is to predict how many goals a striker scores in one football season, based on how many total shots and shots on target they shoot during that season. We decided to focus on the season 2021-2022 and chose to look at data from major European football leagues. We obtained our dataset from *FBref.com: Football Statistics and History*[2]. The dataset has statistics for 2921 players and it has a lot of data related to a player's shooting, like shots on target, goals, goals per shot, average of the distance of the shots taken etc. In total, there are 25 columns from which we will focus on shots taken, shots on target and goals.

As our methods will be supervised, we will use shots taken and shots on target as our features. To achieve our goal of predicting goals scored based on our features, we will naturally use goals as our labels.

3 Methods

3.1 Dataset

We slimmed down the original dataset quite a bit. We chose to take only players who mainly play as strikers (forwards), as they score most of the goals in football, players who have played at least 90 minutes, i.e. a full game during the season, and players who have taken at least a shot during the season. We did this to get rid of most of the data points, which resulted in 0 goals to stop them from skewing the result. We also dropped all columns which didn't help us visualize data or those which weren't the features (shots taken or shots on target) or the label (goals). In the end, we were left with 615 players in our dataset.



3.2 Feature selection

We chose taken shots and shots on target as our features, because as we see from the plot above, although the datapoints are quite dispersed, we see some linearity between the shots taken and goals scored, especially when looking at shots on target. Because strikers are usually in a good goal-scoring position when taking a shot, we don't have to worry about opportunistic 30 meter shots by midfielders or weak rollers from defenders.

3.3 Linear regression

Due to our features having linearistic features, it felt natural to choose linear regression as the first model. It is also a typical choice when the labels are real-valued[3], like ours are. For the loss function, we chose least squared error's cost function mean squared error. It is used to evaluate the performance of most regression algorithms and is the most commonly used loss

function for linear regression[3]. These choices allowed us to use sklearn's ready-made library for the linear regression model[4] and the mean squared error function[5].

3.4 Polynomial regression

The second regression model is polynomial regression. It is the natural follow-up for linear regression due to it being just another type of linear regression. We wanted to try if using polynomial regression with degrees from 2 to 10 would give us a better fit than the linear regression model. As it is just a more complicated version of linear regression, the loss function is again mean squared error.

3.5 Splitting the data

Usually when the dataset is split into the training, validation and test sets, the training set will be 60% of the original set and the validation and test sets will both be 20%. We went with this typical split because with our quite large dataset, we can be sure that the set sizes won't be too small. We split the set using sklearn's method "train_test_split"[6].

4 Results

	Training error	Validation error
Linear regression	4.71525332599386	3.345374792479929

In the table above you can see the training and validations errors for the linear model and below you can see the polynomial model's errors from degrees 2 to 10.

Degree	Training errors	Validation errors
2	4.45205060249269	3.9790051410374483
3	4.260266136634818	4.000072380970007
4	4.005288741462984	4.333238427980481
5	3.88176013122262	4.474678341606459
6	3.7112802996527083	4.594430152896185
7	3.5517650443306183	46.46393455732747
8	4.789400648350709	40.72573185092203
9	22.734431276456487	28493.160482424963
10	114.99234495910846	55737.75426743803

As you can see, the training and validation errors for the polynomial model are quite good until the 7th degree. From the 7th degree onwards the errors start to expand due to overfitting. We would get the best polynomial fit with the second degree because its validation error (3.97900...) is the best with a good training error (4.45205...). Still the linear model clearly gives us the smallest validation error (3.34537...) with a reasonable training error (4.71525...).

This is why we chose linear regression as our final method.

The test set was 20% of the whole dataset as explained in section 3.5. The test set consists of data points that the model has not seen before and it is used to measure how good the model performs. Mean squared error is again used to calculate the average loss incurred on the test set. The test error for the linear model is 4.238926155410416.

5 Conclusion

This project predicted the number of goals a striker scores with satisfactory accuracy. With the chosen machine learning method, linear regression, the validation error was smaller than training error, which means that our data would need to be more worked on. While our data set is large with 615 players in it, there might be too many cases where it is too easy to predict or too hard to learn, as some strikers might have a few shots but most of them goals or many shots but only a few goals.

With mean squared error as the loss function, the performance of both methods were similar when the polynomial degree was small. When the degree was 7 or higher the validation error becomes much higher than the corresponding error with linear regression, as it overfits the data. Therefore further improvements could be having more features, having even more data and splitting the data differently, for example with k-fold cross validation as the edge cases may have bigger effect on the result than we thought.

6 References

[1] https://www.worldatlas.com/articles/what-are-the-most-popular-sports-in-the-world.html
[2] https://fbref.com/en/comps/Big5/2021-2022/shooting/players/2021-2022-Big-5-European-Leagues-Stats
[3] https://towardsdatascience.com/optimization-of-supervised-learning-loss-function-under-the-hood-df1791391c82
[4] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

7 Appendices

```
%config Completer.use_jedi = False  # enable code auto-completion
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv('Football_player_statistics.csv')

# dropping unnecessary columns from the dataset
df.drop(columns=['Nation', 'Rk', 'Comp', 'Age', 'Born', 'FK','PK','PKatt','xG','npxG','npxG/Sh','SoT%',
                 'G/Sh','G/SoT','Dist','G-xG','np:G-xG','Matches', '-9999'], inplace=True)

# renaming the columns
df.columns=['Name','Position','Team','90s','Goals','Shots','Shots on target',
            'Shots/90', 'Shots on target/90']

# drop player who aren't mainly strikers
df = df[df['Position'].astype(str).str[0] == 'F']
# drop strikers who haven't played a full 90 minutes during the season or taken a shot
df = df[df['90s'] >= 1]
df = df[df['Shots'] != 0]
```

	Name	Position	Team	90s	Goals	Shots	Shots on target	Shots/90	Shots on target/90
5	Dickson Abiama	FW	Greuther Fürth	8.1	0	18	4	2.23	0.50
6	Matthis Abline	FW	Rennes	1.1	0	2	0	1.75	0.00
7	Tammy Abraham	FW	Roma	34.3	17	93	32	2.71	0.93
13	Che Adams	FW	Southampton	22.7	7	49	24	2.16	1.06
15	Sargis Adamyan	FWMF	Hoffenheim	3.7	1	3	2	0.82	0.54
...	...	...	...	...	...	...	...	...	...
2897	Simone Zaza	FWMF	Torino	1.4	0	9	3	6.28	2.09
2901	Arber Zeneli	FW	Reims	4.5	2	11	3	2.42	0.66
2903	Andi Zeqiri	FWMF	Augsburg	11.3	2	32	9	2.83	0.80
2911	Hakim Ziyech	FWMF	Chelsea	14.7	4	53	19	3.62	1.30
2913	Simon Zoller	FW	Bochum	6.6	3	11	7	1.66	1.06

615 rows × 9 columns

```
# features
X = df[['Shots', 'Shots on target']].to_numpy().reshape(-1, 2)
# label
y = df['Goals'].to_numpy()

X_train, X_val_train, y_train, y_val_train = train_test_split(X, y, test_size=0.4,
                                                              random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_val_train, y_val_train, test_size=0.5,
                                                              random_state=42)
```

```
# linear regression
regr = LinearRegression()

# training set
regr.fit(X_train, y_train)
y_train_pred = regr.predict(X_train)
tr_error = mean_squared_error(y_train, y_train_pred)

# validation set
regr.fit(X_val, y_val)
y_val_pred = regr.predict(X_val)
val_error = mean_squared_error(y_val, y_val_pred)
```

```
# table for linear

data =  [
            [                'Training error', 'Validation error'],
            ['Linear regression',  f"{tr_error}", f"{val_error}"]
        ]

# Pop the headers from the data array
column_headers = data.pop(0)
row_headers = [x.pop(0) for x in data]

# Add a table at the bottom of the axes
the_table = plt.table(cellText=data,
                      rowLabels=row_headers,
                      rowLoc='right',
                      colLabels=column_headers,
                      loc='center')

# Scaling is the only influence we have over top and bottom cell padding.
# Make the rows taller (i.e., make cell y scale larger).
the_table.scale(1, 2)

# Hide axes
ax = plt.gca()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)

# Hide axes border
plt.box(on=None)
```

```
# test error for linear
regr = LinearRegression()
regr.fit(X_test, y_test)
y_test_pred = regr.predict(X_test)
test_error = mean_squared_error(y_test, y_test_pred)

print(f"Test error: {test_error}")
```

```
Test error: 4.238926155410416
```

```
# data scatter plot
plt.scatter(X[:,0], y, s=8)
plt.scatter(X[:,1], y, s=8)
plt.xlim([0,100])
plt.ylim([0,30])
plt.legend(['shots', 'shots on target'], loc='upper left')
plt.title('Dataset')
plt.xlabel('Shots/shots on target')
plt.ylabel('Goals')
plt.show()
```

```
# polynomial regression
lin_regr = LinearRegression(fit_intercept=False)
tr_errors = []
val_errors = []

for i in range(2, 11):
    poly = PolynomialFeatures(i)

    # training set
    X_train_poly = poly.fit_transform(X_train)
    lin_regr.fit(X_train_poly, y_train)
    y_train_pred = lin_regr.predict(X_train_poly)
    tr_errors.append(mean_squared_error(y_train, y_train_pred))

    # validation set
    X_val_poly = poly.fit_transform(X_val)
    y_val_pred = lin_regr.predict(X_val_poly)
    val_errors.append(mean_squared_error(y_val, y_val_pred))
```

```
# table for polynomial

data = [['Degree', 'Training errors', 'Validation errors']]
for i in range(2,11):
    data.append([str(i), str(tr_errors[i-2]), str(val_errors[i-2])])

column_headers = data.pop(0)

the_table = plt.table(cellText=data,
                      rowLabels=None,
                      rowLoc='right',
                      colLabels=column_headers,
                      loc='center')

the_table.scale(1, 2)

ax = plt.gca()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)

plt.box(on=None)
```