Predicting Football Match Goals with Polynomial Regression and Decision Tree Classifier

1. Introduction

Football is a game that is enjoyed worldwide and is loved by billions of people. It is also a very competitive sport and a ruthless one at that. Sometimes the outcome of matches can be determined by pure luck and not a quality performance. Hence, it is important to be able to analyze the team's performance in order to see whether they are losing matches, because they are playing badly or if they are doing everything right, but have had some bad luck. Since the traditional prediction approach based on intuition is not reliable, there has been an increase of data-driven approaches in analyzing performance in football.

The aim of this project is to predict the number of goals scored in a football match with supervised Machine Learning methods based on in-game statistics, including, but not limited to home and away shots, chances for and against as well as possession. The dataset used will be from the English Premier League during the 2022-2023 season, provided by Thamer Sekhri. [1] Using these statistics teams could predict how many goals they and their opponents should have scored or conceded and compare that value to the actual value. This can help determine whether the teams overperformed or underperformed in any game. To achieve this aim, polynomial regression with degree 2 was used with mean squared error as the loss function. As an alternative, decision tree regression with a maximum depth of 4 was applied with mean squared error as loss function. These different models were compared on a test set by using mean squared error.

Section 1 introduces the topic, while section 2 discusses how to formulate the problem, in a way that would be suitable for applying different machine learning algorithms. Section 3 then discusses the details of the dataset, data processing, application of polynomial and decision tree regression models. Section 4 compares the results of different methods, evaluates them and lastly section 5 summarizes the report and discusses possible improvements that could be made in the future.

2. Problem Formulation

In any given competitive football game there is an away team and a home team. They both attempt to score more goals than the other. In this project the aim is to predict the number of goals by the home team in the match based on in-game statistics. To do this, the English Premier League season 2022-2023 was split into three parts with ratios 0.6, 0.2, and 0.2. First part was used to train the model and the other two were for testing and validation data

The training set, comprising 60% of the dataset, is constructed by randomly selecting samples, which ensures that the model has enough data to learn from while avoiding overfitting. The validation set, representing 20% of the data, is also chosen randomly. This set is used for assessing the model's performance without the risk of overfitting. The test set, composed of 20% of the data, is randomly selected from the remaining samples. The test set is reserved for evaluating the best model's generalization performance and assessing how well it performs on unseen data, ensuring a reliable

measure of model effectiveness.

These decisions are motivated by several factors. The model should have enough training data while maintaining effective validation and testing. The 60-20-20 ratio balances between these objectives.

To limit the variability of data and increase the accuracy of the model only one league during one season was considered. Some of the data statistic types that are present in the data are Date, Clock, Stadium, Attendance, Goals Home, Away Goals.

In this project feature values are home and away blocked shots, shots on and off target, possession, corners, where all these numbers are integer or float values. All of the variables are based on in-game statistics. These were selected based on domain knowledge, because they are closely linked to the result of the game. For example, the reason why the number of shots is a feature is that, if a team has many shots, they are likely to score more goals from them.

The label value is the number of goals in the match as a float value.

3. Methods

3.1. Dataset

In the dataset, 14820 data points with 39 different variables are available to use, without missing data in any fields. The values are either string, being team names, or integers and float, being the statistics themselves. There are many variables that are irrelevant to this research question. For example, the stadium, clock and attendance are not useful in determining the number of goals in a football match. Hence, there was a need to clean the data by removing the unnecessary variables.

3.2. Polynomial regression model

It can be seen by analyzing the data that there is no linear relationship between the feature values and the label value. That is why it was acceptable to use polynomial regression to make predictions about the label value. The formula for polynomial regression with degree k is:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_k x^k \quad [2]$$

The loss function chosen is mean squared error (MSE) as it is typically used with polynomial regression. It quantifies the quality of the polynomial regression model by measuring the average squared difference between the model's predictions and the actual target values or the training data. The most important feature of the MSE is the fact that its rate of punishment increases exponentially. When the dataset contains outliers it could be a problem, but this dataset does not have outliers. Because of this reason, MSE is a reasonable choice of loss function. The formula for the mean squared error is:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\widehat{y_i} - y_i)^2 \quad [2]$$

where $\widehat{y}$ is the prediction and y is the actual value.

Answering the problem using this regression model required scikit-learn's LinearRegression [3], PolynomialFeatures [4], and mean_squared_error [5] methods.

3.3. Decision tree regression model

In addition to a polynomial model approach a second non-linear regression model needed to be chosen and one effective with nonlinear relationships is decision tree regression. It was chosen for this reason and its adaptability to various feature values, as well as its relatively lower demand for data preprocessing.
The fundamental working principle of decision trees involves dividing the dataset into smaller subsets by posing binary, practically true or false, questions. The initial training data is input at the root node, leading to its division based on true/false questions. Then, the system proceeds by posing questions to the data at decision or interior nodes, further branching out. At some point though, the system decides to stop branching out at particular points, which become leaf nodes. These leaf nodes constitute the terminal points of the model, and predictions are generated by assessing these leaf nodes. Decision tree regression can be used for both data classification and continuous value prediction.

The loss function was chosen to be the same as with the polynomial regression model - mean squared error loss for the same reasons as earlier. It is common to use mean squared error with decision tree regressors as it is the default loss function and since the dataset doesn't contain so many outliers, the mean squared error could give a reasonable value.

Answering the problem using this regression model required scikit-learn's DecisionTreeRegressor [6], and mean_squared_error [7] methods in addition to basic Python commands.

4. Results

4.1 Polynomial regression model results

One of the most important aspects when employing polynomial regression is about the degree selection process. This choice has a direct impact on the relationship between the features and the target variable. In order to not fall victim to underfitting or overfitting, a widely accepted model validation technique was employed. This involved assessing polynomial regression models with degrees ranging from 2 to 10 by calculating the mean squared error for both training and validation datasets as can be seen in Table 1.

Table 1. Training and Validation Errors for Polynomial Regression degrees 2 to 10

Degree	2	3	4	5	6	7	8	9	10
Validation Err.	0.73980	6.47872	6.37817	7.01795	7.53061	39.4059	41.7085	44.1794	46.8642
Training Err.	0.56956	1.2e-23	4.6e-25	5.3e-25	9.7e-25	1.6e-25	1.4e-24	1.4e-24	1.4e-24

Considering these training and validation errors, the optimal degree choice for polynomial regression is degree 2 since it gives the smallest validation error with an acceptable training error. Based on the provided values, we can reasonably conclude that degree 2 represents the point just before the model performance drops significantly and it overfits. This could suggest that a linear model would be better.

4.2 Decision tree regression model results

Similar to a polynomial regression model, there is an important choice to be made when working with decision trees. That choice is about the tree's depth. The deeper the model is, the more complex it gets, increasing the chance of overfitting. However, a model that is too simple will not be very accurate and

balancing these two aspects is difficult. To solve this problem a similar approach as in polynomial regression was used. This means that mean squared error was used to compute the validation and training dataset errors across a depth range spanning from 1 to 9 as illustrated in Table 2.

Table 2. Training and Validation Errors for Decision Tree Regression from 1 to 9

Max. Depth	1	2	3	4	5	6	7	8	9
Validation Err.	1.96632	1.53848	1.46427	0.97823	0.94904	0.75233	0.77998	0.93146	0.91611
Training Err.	1.28451	0.99973	0.66273	0.45384	0.30516	0.14380	0.05756	0.02444	0.00548

Considering these training and validation errors, the optimal depth choice for the decision tree regressor was chosen to be 6. It gives the lowest validation error along with an acceptable training error. After a maximum depth of 6, the model begins to overfit.

4.3 Comparing the models

When comparing the validation errors of both models, namely polynomial regression with degree 2 (0.73980) and decision tree regression with a maximum depth of 6 (0.75233) it can be seen that one of them is performing slightly better. Based on this observation, polynomial regression with degree 2 was chosen as the final model.

The last step was to analyze the performance of the model based on the test set. The test set is a set of elements that the model has not seen before and outputs a useful value that can be used to measure the model's success. In this project, the test set is 20% of the whole dataset and contains 2280 data points. The test set accuracy was calculated by using mean squared error and for the final chosen model it was 0.85902.

5. Conclusion

The fact that the best polynomial regression outperformed decision tree models was expected, despite the latter's success in classification tasks. The reason why is the potential loss of information when decision trees answer binary questions, a known limitation in their application. Higher degrees of the polynomial had significantly smaller training errors, which means that the models overfit. This could suggest that a linear model is the most accurate, which would contradict the assumption that was made about no linear relationship between the data. Testing a linear model should be the next step. In general, the performance of the models was not optimal. Since the dataset used was relatively small, one potential solution is increasing the dataset, for example using several seasons of the Premier League, which could improve the accuracy.

The final choice for the model was a polynomial regression model with a degree of 2, which yielded a test error of 0.85902. While this is not extremely accurate, it falls within acceptable tolerance limits for our objectives. Relevant features were selected based on domain knowledge, which seems to be sensible. However, there is a risk that this method overlooks subtle mathematical relationships between the target variable and certain feature values. Therefore, it could be interesting to consider increasing the range of feature values, which could affect the result. It's important to note, though, that expanding the feature space also results in a rise in computational complexity, which requires more analysis to identify the optimal balance.

6. Sources

[1] Sekhri, T. (2023, July). Football data : Top 5 leagues. Kaggle. Retrieved September 12, 2023, from https://www.kaggle.com/datasets/thamersekhri/premier-league-stats-2022-2023

[2] Abhigyan. (2020, August 2). Understanding polynomial regression. Medium. Retrieved March 10, 2022, from https://medium.com/analytics-vidhya/understanding-polynomial-regression-5ac25b970e18

[3] Sklearn.linear_model.linearregression. scikit-learn. Retrieved September 22, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[4] Sklearn.preprocessing.PolynomialFeatures. scikit-learn. Retrieved September 22, 2023, from https://scikit-learn.org/stable/modules/preprocessing.html#polynomial-features

[5] Sklearn.metrics.mean_squared_error. scikit-learn. Retrieved September 22, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

[6] Sklearn.metrics.mean_squared_error. scikit-learn. Retrieved October 4, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

[7] Sklearn.metrics.DecisionTreeRegressor. scikit-learn. Retrieved October 4, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

7. Code Appendix

Although the codes in this report are written from scratch for this report, they are inspired by the course assignments.

mlProject

October 11, 2023

[1]:
```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as pyplot
from matplotlib import style
import seaborn as sns
import sklearn as skl
from sklearn.metrics import accuracy_score


from sklearn.datasets import make_regression
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
from sklearn.tree import DecisionTreeRegressor
```

[ ]:
```
raw_data = pd.read_csv('Premier_League.csv', encoding = 'unicode_escape')
```

[ ]:
```
final_data = raw_data.drop(['date', 'links', 'clock', 'stadium', 'attendance',␣
 ↪'Home Team',
                            'Goals Home', 'Away Team', 'Away Goals'], axis=1)
predict = raw_data['Goals Home']

X = np.array(final_data)
y = np.array(predict)

X_train, X_val_test, y_train, y_val_test = train_test_split(X, y, test_size=0.
 ↪4, random_state=55)
X_val, X_test, y_val, y_test = train_test_split(X_val_test, y_val_test,␣
 ↪test_size=0.5, random_state=55)
```

[ ]:
```
dec_errors_tr = []
dec_errors_val = []
depths = []
for depth in range(1, 10):
    decst_regr = DecisionTreeRegressor(max_depth=depth)
```

```
    decst_regr.fit(X_train, y_train)

    y_pred_dec_train = decst_regr.predict(X_train)
    dec_error_tr = mean_squared_error(y_train, y_pred_dec_train)
    y_pred_dec_val = decst_regr.predict(X_val)
    dec_error_val = mean_squared_error(y_val, y_pred_dec_val)

    dec_errors_tr.append(dec_error_tr)
    dec_errors_val.append(dec_error_val)
    depths.append(depth)

print(dec_errors_val)
print(dec_errors_tr)
```

[ ]:
```
poly_tr_errors = []
poly_val_errors = []
poly_degrees = []

for degree in range(2, 10):
    poly = PolynomialFeatures(degree=degree)
    X_train_poly = poly.fit_transform(X_train)
    lin_regr = LinearRegression()
    lin_regr.fit(X_train_poly, y_train)

    y_pred_poly_train = lin_regr.predict(X_train_poly)
    poly_tr_error = mean_squared_error(y_train, y_pred_poly_train)

    X_val_poly = poly.fit_transform(X_val)
    y_pred_poly_val = lin_regr.predict(X_val_poly)
    poly_val_error = mean_squared_error(y_val, y_pred_poly_val)

    poly_tr_errors.append(poly_tr_error)
    poly_val_errors.append(poly_val_error)
    poly_degrees.append(degree)

print(poly_val_errors)
print(poly_tr_errors)
```

[ ]:
```
X_test_poly = poly.fit_transform(X_test)
y_poly_test_pred = lin_regr.predict(X_test_poly)
poly_test_error = mean_squared_error(y_test, y_poly_test_pred)
y_dec_test_pred = decst_regr.predict(X_test)
dec_test_error = mean_squared_error(y_test, y_dec_test_pred)
print(poly_test_error)
print(dec_test_error)
```