Grady Lynch

English Premier League Financial Analysis 2

Subject:

The financial landscape of soccer in the English Premier League is very complicated, and very volatile. Unlike popular American sports leagues like the NFL or the NBA, the Premier League does not have a salary cap. This means that Premier League teams are not limited by the league in how much money they spend on their players' salaries. Another important financial aspect of the Premier League is the presence of player transfer fees. The most common way that a soccer player moves to a new team is when one team pays a transfer fee to the player's current team in exchange for that player. So when an ambitious team with financial muscle decides to buy all the best players and spend boatloads of money, this team should be dominant. However, this hasn't always been the case historically.

I am interested in finding out what the actual value of the money spent by Premier League teams is in terms of their performance.

Data:

In the first edition of this project, my final dataset had only 67 rows. This was because I couldn't find a way to use long term financial data of teams that had been relegated or promoted. In order to have more data, I needed to be able to use the data from these teams. What I did was I normalized my financial features by dividing them by the corresponding clubs 'Market Value' for that year. This also did the job of adjusting for inflation. What I hoped was that this would improve the quality of my financial data because the Premier League is known to be very top heavy financially.

The data that I used was the data of teams that had been in the Premier League the previous season from 2015 to 2023. The transfer, points, and market value data I got from Transfermarkt.com. The salary data I got from Capology.com. For the Transfermarkt data, I used the Transfermarkt scraper from Apify to scrape the data into csv format. Next I loaded my dataset into a jupyter notebook to get it ready for WEKA. My final dataset had 150 rows. Here is the code from the notebook.

```python
def add_columns(current_df, previous_df):

    df = current_df.copy()

    # SalaryDelta = Difference in salaries from previous year
    df['SalaryDelta'] = df['SalaryTotal'] - previous_df['SalaryTotal'].values

    # Add normalized values for 1yrNetSpend and SalaryDelta by dividing them by MarketValue
    df['1yrNetSpendNormalized'] = df['1yrNetSpend'] / df['MarketValue']
    df['SalaryDeltaNormalized'] = df['SalaryDelta'] / df['MarketValue']

    # 2yrNetSpendNormalized = Sum of current and previous year's normalized spending
    df['2yrNetSpend'] = df['1yrNetSpendNormalized'] + previous_df['1yrNetSpendNormalized'].values

    # Keep only required columns
    df = df[['1yrNetSpendNormalized', '2yrNetSpend', 'SalaryDeltaNormalized', 'improved', 'SalaryTotal']]

    return df

def get_1yrNetSpendNormalized_2014(current_df):

    df = current_df.copy()

    # Normalize the 1yrNetSpend value for the first dataframe
    df['1yrNetSpendNormalized'] = df['1yrNetSpend'] / df['MarketValue']

    return df
```

```python
df2014NEW = get_1yrNetSpendNormalized_2014(df2014)
df2015NEW = add_columns(df2015, df2014NEW)
df2016NEW = add_columns(df2016, df2015NEW)
df2017NEW = add_columns(df2017, df2016NEW)
df2018NEW = add_columns(df2018, df2017NEW)
df2019NEW = add_columns(df2019, df2018NEW)
df2020NEW = add_columns(df2020, df2019NEW)
df2021NEW = add_columns(df2021, df2020NEW)
df2022NEW = add_columns(df2022, df2021NEW)
df2023NEW = add_columns(df2023, df2022NEW)

# Combine all DataFrames
final_df = pd.concat([df2015NEW, df2016NEW, df2017NEW, df2018NEW, df2019NEW, df2020NEW, df2021NEW, df2022NEW, df2023NEW], ignore_index=True)

# Remove the rows that have 'unkown' in the improved column
final_df_filtered = final_df[final_df['improved'] != 'unknown']

final_df_filteredB = final_df_filtered[final_df_filtered['improved'] != 'same']

# Keep only required columns
final_df_filteredB = final_df_filteredB[['1yrNetSpendNormalized', '2yrNetSpend', 'SalaryDeltaNormalized', 'improved']]

final_df_filteredB.to_csv('PremierLeagueFinancesV2_FinalData2.csv', index=False)
```

Features:

In the first version of this project, the features that I used were the one and two year net transfer expenditure, as well as the change in total salaries year by year. I called these 1yrNetSpend, 2yrNetSpend, and SalaryDelta, and all of these values were adjusted for

inflation and converted to USD. I also had PointsDelta, which was the change in points earned by a team year to year.

PointsDelta was my outcome on which I tried to use regression models to predict. I was disappointed by the results of this, and I finished my first report with a list of a few things that I wanted to do differently next time.

- Change PointsDelta to a nominal value.
- Normalize financial data using teams' market value
- Curate more data, including data for teams that have been promoted for 2 years.

My outcome for the second version of this experiment was a nominal value, 'improved', that tracked whether or not the team's points total had improved as either yes, no, or same. As I mentioned already, I was able to use the market value data from Transfermarkt to normalize my data. Normalizing the data also allowed me to use data from newly promoted teams even though the financial size of those clubs is a fraction of that of some of the top end clubs. This allowed me to increase the size of my dataset from 67 rows to 150 rows.

Experiments & Results:

Like the first iteration of this experiment, I loaded my csv file into Weka and converted it to an arff file in the preprocessing tab. Then I used the Weka Experiment Environment to run all of my classifiers and algorithms at once and get easy to interpret output. This time I was able to use classifiers because of my nominal outcome. I ran the experiment first with a train test split of 75/25 and 22 classifiers and algorithms.

75/25 Train Test Split Results

Dataset	(1) rules.Ze	(2) bayes	(3) bayes	(4) bayes	(5) funct	(6) funct	(7) lazy.	(8) lazy.	(9) meta.
PremierLeagueFinancesV2_F (10)	50.68	50.68	43.48	43.48	45.60	45.85	49.31	48.26	43.17 *
(v/ /*)		(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/0/1)

(10) meta	(11) meta	(12) meta	(13) meta	(14) meta	(15) rule	(16) rule	(17) rule	(18) rule	(19) tree	(20) tree	(21) tree	(22) tree
44.24	39.44 *	45.60	47.73	47.17	50.68	47.19	46.96	50.68	46.69	45.86	45.33	47.19
(0/1/0)	(0/0/1)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)

	Percent Correct
1. Zero R	50.68
2. BayesNet	50.68
3. NaiveBayes	43.48
4. NaiveBayesUpdateable	43.48
5. Logistic	45.60
6. MultilayerPerceptron	45.85
7. Lazy Ibk	49.31
8. lazy Kstar	48.26
9. AdaBoostM1	43.17
10. Bagging	44.24
11. LogitBoost	39.44
12. MultiClassClassifier	45.60
13. RandomCommittee	47.73
14. RandomizableFilteredClassifier	47.17
15. DecisionTable	50.68
16. Jrip	47.19
17. OneR	46.96
18. PART	50.68
19. LMT	46.69
20. RandomForrest	45.86
21. RandomTree	45.33
22. REPTree	47.19

I also Ran the same classifiers and algorithms using cross validation. In my last project, my results improved when I used cross validation.

Cross Validation Results

Dataset	(1) rules.Ze	(2) bayes	(3) bayes	(4) bayes	(5) funct	(6) funct	(7) lazy.	(8) lazy.	(9) meta.
PremierLeagueFinancesV2_F(100)	50.67	50.67	47.67	47.67	46.40	47.67	50.87	48.53	43.00
	(v/ /*)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)

(10) meta	(11) meta	(12) meta	(13) meta	(14) meta	(15) rule	(16) rule	(17) rule	(18) rule	(19) tree	(20) tree	(21) tree	(22) tree
43.07	39.67 *	46.40	45.27	49.80	50.67	44.87	48.40	50.67	48.20	43.80	45.07	46.27
(0/1/0)	(0/0/1)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)	(0/1/0)

	Percent Correct
1. Zero R	50.67
2. BayesNet	50.67
3. NaiveBayes	47.67
4. NaiveBayesUpdateable	47.67
5. Logistic	46.40
6. MultilayerPerceptron	47.67
7. Lazy Ibk	50.87
8. lazy Kstar	48.53
9. AdaBoostM1	43.00
10. Bagging	43.07
11. LogitBoost	39.67
12. MultiClassClassifier	46.40
13. RandomCommittee	45.27
14. RandomizableFilteredClassifier	49.80
15. DecisionTable	50.67
16. Jrip	44.87
17. OneR	48.40
18. PART	50.67
19. LMT	48.20
20. RandomForrest	43.80
21. RandomTree	45.07
22. REPTree	46.27

Conclusions & Next Steps

Once again, the results of the experiment were somewhat disappointing. None of the results significantly exceed the baseline of 50%. Despite using a nominal outcome, having more data, and normalizing my financial data, I still wasn't able to make accurate predictions of a team's performance based on financial data.

I am still determined to expand on this experiment and continue to try and accurately predict Premier League clubs performance. I think that the features that I have are strong, and that they are intuitive and comparative financial metrics. I am considering adding more features, maybe even some non-financial ones.