Predicting Premier League wins using Machine Learning

Introduction

The Premier League is one of the most widely followed football competitions globally, captivating audiences with its intense matches and huge fan bases. With the huge following of the league, the ability to predict match outcomes has garnered considerable attention. Using Machine Learning methods, we will work to identify the factors that determine success in Premier League games.

This project revolves around the application of the Machine Learning technique known as the Random Forest Classifier. By utilising key metrics such as team performance, match scheduling, venue, opponents, and tactical formations, we aim to make informed predictions about match winners.

The report consists of six essential parts, each contributing to a comprehensive understanding of the project. In 'Problem Formulation', we define the predictive framework for the match outcomes and outline key dataset components. 'Methods' dives into dataset specifics, outlining preprocessing steps, feature selection rationale, and justifications for employing Random Forest and Linear Regression models. This section also introduces the chosen loss functions and the model validation process. 'Results' presents the performance metrics of both models, highlighting their respective strengths and outcomes. The 'Conclusion' provides a concise summary, discussing the chosen method's superior accuracy and outlining potential areas for improvement. The 'Bibliography/References' section acknowledges sources, while 'Appendices' include the code.

This report aims to provide a comprehensive overview of our methodology and the steps we go through along the way. We will begin by establishing a clear understanding of the problem, including exploration of the data points and their relevance. Then, in order to ensure the dataset's quality and applicability, we will explain where it came from and how it was created.

In this project, we are not only attempting to forecast Premier League match winners, but we also hope to learn more about the complex factors that influence these results. By mixing our passion for football with machine learning techniques, we hope to explain the subtle aspects of the "beautiful game."

Problem Formulation

To approach the task of predicting Premier League game winners as a Machine Learning problem, we first need to define the key components

We are formulating this as a supervised binary classification problem. Each instance in our dataset represents a Premier League match. The goal is to predict whether a specific team will win or not based on a set of features and compare it to the actual results.
This is a supervised learning task, since the model learns from labelled examples, where it's provided with both the input features and the corresponding target labels (in this case, whether a team wins or not).

Data Points, Features, and Labels:

Data Points:

Each data point in our dataset corresponds to a single Premier League match. It contains information about both competing teams, such as their performance statistics, the date and time of the match, the

round of play, the venue, the opponent, and the tactical formation utilised. The data is categorical eg. team names and also numerical eg. match date and time and round numbers.

Features:

These are the attributes or characteristics of a match that we use to make predictions. Our features encompass a range of statistics and match details, including team performance metrics, match timing, round number, venue, opponent identity, and tactical formations employed by the teams.

Labels:

The labels indicate the outcome of each match. Specifically, it denotes whether the team we are interested in predicting (referred to as the "home" team) emerged as the winner (label = 1) or not (label = 0) for simplicity reasons a draw will also be labelled as 0.

Source of the Dataset:

The data we will use comes from FBref.com which is a reputable sports statistics source, including official Premier League records. It consists of a thorough collection of match data spanning numerous Premier League games, giving our machine learning model a large and varied set of cases to learn from. The data needs a lot of cleaning before being usable since we need different data from multiple datasets on the website. We have created our own file matches.csv containing the cleaned up data.

Methods

Number of Datapoints and Dataset Description:

The dataset contains a total of 1389 records. It includes a comprehensive collection of Premier League match data spanning two different seasons. Each record represents a single match and includes a range of features, such as team performance metrics, match details such as date, time, round, venue information, opponent identity, and tactical formations used by the teams.

Data Preprocessing:

Prior to utilising the dataset for modelling, we conducted necessary data preprocessing steps. We filtered out the needed features from different sets to form our "matches.csv" file. We also handled missing values, ensuring data consistency.

Feature Selection Process:

The feature selection process involved evaluating the relevance and importance of each feature for predicting Premier League match outcomes. As football fans we have selected features that have a visibly significant impact on match results, such as team performance metrics and venue details, and prioritised them.

Choice of ML Models:

We chose the Random Forest Classifier as the main machine learning model for our project. This is because of the model's capacity to handle complicated relationships in the data, robustness to noisy input, and ability to identify nonlinear patterns. The Random Forest Classifier was the most viable contender for the prediction job considering the variety and complexity of Premier League match data.

We chose Linear Regression as the secondary model to provide a different perspective on the problem. While it may seem counterintuitive to apply a linear model to a problem with complex, non-linear relationships, Linear Regression can still offer valuable insights as it provides a clear and interpretable view of the relationship between individual features and the outcome variable.

This can help identify which specific factors have a direct and linear impact on match outcomes. For instance, it may reveal if metrics like team performance or match timing have a more straightforward, linear influence.

By incorporating both Random Forest and Linear Regression, we aim to gain a comprehensive understanding of the predictive power of different models and potentially uncover unique insights into the factors that affect Premier League match outcomes.

Choice of Loss Function

For this particular project we have decided to use both Cross Entropy and Mean Squared Error as the loss functions. For our Random Forest Classifier, we utilised Cross Entropy as the primary loss function since it is well-suited for multi-class classification tasks like predicting Premier League match outcomes. It effectively penalises misclassifications which is important for accurately predicting results.

We chose Mean Squared Error as the principal loss function in the context of Linear Regression because while our primary goal is classification, including MSE is a useful indicator for evaluating regression performance. This option is especially useful for analysis using match statistics or performance measures, since it gives a clear metric of prediction errors, improving our ability to analyse and assess the regression-based approach's outcomes.

Model Validation Process

To evaluate the model's performance, we will use a splitting approach. The dataset will be divided into three subsets. The training set, used for training the model, which will be 60% of the data, the validation set, 20% of the data, to finetune the hyperparameters and model evaluation and the testing set also 20% of the data, reserved for final evaluation and simulating real scenarios with unseen data.

This approach achieves a balance between the necessity for sufficient training data and the need for different sets for fine-tuning and final evaluation, which results in a reliable and adaptable predictor of Premier League match outcomes.

Results

In this analysis, two machine learning models were evaluated: Random Forest and Linear Regression.

Random Forest:

- Validation Accuracy using Entropy: 0.64
- Test Accuracy using Entropy: 0.57
- Precision when predicting wins: 38%

Linear Regression:

- Training Set MSE: 0.24

- Test Set Accuracy: 0.75
- Test Set Precision: 0.33
- Test Set MSE: 0.21
- Validation Set MSE: 0.25

Final Chosen Method:

After comparing the results, it appears that the Linear Regression model performs better in terms of accuracy and precision. It has a higher test set accuracy (75%) compared to the Random Forest model (57%). Additionally, the precision of the Linear Regression model for predicting wins is 33%, which is higher than the Random Forest model's precision of 38%.

Test Error of the Final Chosen Method:

The test error for the final chosen method, which is Linear Regression, is measured by the Mean Squared Error (MSE) and is found to be 0.21.

In conclusion, based on the analysis and comparison of both models, the Linear Regression model is chosen as the final method for predicting match outcomes. It demonstrates better performance in terms of accuracy, precision, and MSE on the test set. The test error, measured by MSE, is 0.21 for the Linear Regression model. This indicates that, on average, the predicted outcomes are within a reasonable range of the actual outcomes.

Conclusion

In conclusion, this report evaluated how well two machine learning models—Linear Regression and Random Forest—performed at predicting match results for a particular dataset. To choose the most useful model, the analysis compared training and validation errors.

The Linear Regression model demonstrated promising results, with a lower test set Mean Squared Error (MSE) of 0.21, indicating that its predictions were closer to the actual outcomes. However, it's important to note that while the Linear Regression model showed good results, there may still be room for improvement.

The possibility of overfitting should be considered, as the training error was noticably smaller than the validation error. This suggests that the model might be too complex for the given data, and regularisation techniques or alternative models could be explored to address this issue.

To enhance the performance of the ML method, several strategies can be considered like leveraging additional features of data points, experimenting with different models, and exploring alternative loss functions for training. Additionally, collecting a larger and more diverse dataset could provide the models with a broader range of patterns to learn from, potentially leading to more accurate predictions.

Overall, while the Linear Regression model shows promise, there is still room for refinement. By addressing potential overfitting and exploring alternative modelling techniques, we can work towards a more accurate prediction system for match outcomes. Additionally, the use of more data can be important for achieving even more accurate and reliable results in the future.

Bibliography/References

https://www.geeksforgeeks.org/python-mean-squared-error/

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

https://madewithml.com/courses/foundations/pandas/

Machine_Learning

October 11, 2023

[1]:
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

#import requests


matches = pd.read_csv("matches.csv", index_col = 0)
# Drop the third column

column_to_drop = matches.columns[2] # This gets the label of the 3rd column
matches = matches.drop(column_to_drop, axis=1) #dropping the useless columns,
matches["date"] = pd.to_datetime(matches["date"])
matches["venue_code"] = matches["venue"].astype("category").cat.codes #Home␣
 ↪1,Away 0

matches["opp_code"] = matches["opponent"].astype("category").cat.codes # Code␣
 ↪for each team
matches["hour"] = matches["time"].str.replace(":.+", "", regex = True).
 ↪astype("int")

matches["day_code"] = matches["date"].dt.dayofweek # Number for each day ->␣
 ↪Monday = 0 , Tuesday = 1 ...
matches["target"] = ( matches["result"] == "W").astype("int") # 0 = loss/draw ..
 ↪. 1 = won


matches.head()
```

[1]:

	date	time	round	day	venue	result	gf	ga	opponent	\
1	2021-08-15	16:30	Matchweek 1	Sun	Away	L	0.0	1.0	Tottenham	
2	2021-08-21	15:00	Matchweek 2	Sat	Home	W	5.0	0.0	Norwich City	
3	2021-08-28	12:30	Matchweek 3	Sat	Home	W	5.0	0.0	Arsenal	

```
4 2021-09-11  15:00  Matchweek 4  Sat  Away      W  1.0  0.0  Leicester City
6 2021-09-18  15:00  Matchweek 5  Sat  Home      D  0.0  0.0      Southampton

    xg  …   fk   pk  pkatt season            team venue_code opp_code  \
1  1.9  …  1.0  0.0    0.0   2022  Manchester City          0       18
2  2.7  …  1.0  0.0    0.0   2022  Manchester City          1       15
3  3.8  …  0.0  0.0    0.0   2022  Manchester City          1        0
4  2.9  …  0.0  0.0    0.0   2022  Manchester City          0       10
6  1.1  …  1.0  0.0    0.0   2022  Manchester City          1       17

   hour  day_code  target
1    16         6       0
2    15         5       1
3    12         5       1
4    15         5       1
6    15         5       0

[5 rows x 31 columns]
```

[2]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, mean_squared_error


# Train RandomForest with entropy
rf_entropy = RandomForestClassifier(n_estimators = 50, min_samples_split = 10,␣
 ↪criterion = 'entropy', random_state=1)

# Define the predictors and split the data
predictors = ["venue_code", "opp_code", "hour", "day_code"]
data_before_2022 = matches[matches["date"] <='2021-12-31' ]
train, validation = train_test_split(data_before_2022, test_size=0.2,␣
 ↪random_state=100) # 80% training, 20% validation
test = matches[matches["date"] >'2022-01-01' ] # no games on a new years day

rf_entropy.fit(train[predictors], train["target"])
val_preds_entropy = rf_entropy.predict(validation[predictors])
val_acc_entropy = accuracy_score(validation["target"], val_preds_entropy)
print(f"Validation Accuracy using Entropy: {val_acc_entropy:.2f}")

# You can also test on the test set if needed
test_preds_entropy = rf_entropy.predict(test[predictors])
test_acc_entropy = accuracy_score(test["target"], test_preds_entropy)
print(f"Test Accuracy using Entropy: {test_acc_entropy:.2f}")
precision_rf_entropy = precision_score(test["target"], test_preds_entropy)
```

```
Validation Accuracy using Entropy: 0.64
Test Accuracy using Entropy: 0.57
```

[3]:
```
combined = pd.DataFrame(dict(actual = test["target"], prediction =␣
 ↪test_preds_entropy))
pd.crosstab(index = combined["actual"], columns = combined["prediction"])
# when we predicted win (most of the time we were wrong
```

[3]:
```
prediction    0   1
actual
0           135  37
1            81  23
```

[4]:
```
from sklearn.metrics import precision_score

precision_score(test["target"], test_preds_entropy) # CheckPoint
# when we predicted W, team only won 38% of the time
```

[4]:
```
0.38333333333333336
```

[5]:
```
grouped_matches = matches.groupby("team")
group = grouped_matches.get_group("Liverpool")
```

[6]:
```
#improve precision with rolling_averages
def rolling_averages(group, cols, new_cols):
    group = group.sort_values("date") # checking teams recent perfomance
    rolling_stats = group[cols].rolling(3, closed ='left').mean()
    # closed = 'left' prevents it getting knowledge from the future
    group[new_cols] = rolling_stats
    group = group.dropna(subset = new_cols) # removes all the rows that have␣
 ↪missing values
    return group
```

[7]:
```
cols = ["gf", "ga", "sh", "sot","dist", "fk", "pk", "pkatt"]

new_cols = []
for i in cols:
    new_cols.append(f"{i}_rolling")
new_cols
```

[7]:
```
['gf_rolling',
 'ga_rolling',
 'sh_rolling',
 'sot_rolling',
 'dist_rolling',
 'fk_rolling',
 'pk_rolling',
 'pkatt_rolling']
```

[8]:
```
rolling_averages(group, cols, new_cols).head()
matches_rolling = matches.groupby("team").apply(lambda x:
 ↪rolling_averages(x,cols, new_cols))
matches_rolling = matches_rolling.droplevel('team')
matches_rolling.index = range(matches_rolling.shape[0])
matches_rolling.head()
```

[8]:

	date	time	round	day	venue	result	gf	ga	opponent	\
0	2020-10-04	14:00	Matchweek 4	Sun	Home	W	2.0	1.0	Sheffield Utd	
1	2020-10-17	17:30	Matchweek 5	Sat	Away	L	0.0	1.0	Manchester City	
2	2020-10-25	19:15	Matchweek 6	Sun	Home	L	0.0	1.0	Leicester City	
3	2020-11-01	16:30	Matchweek 7	Sun	Away	W	1.0	0.0	Manchester Utd	
4	2020-11-08	19:15	Matchweek 8	Sun	Home	L	0.0	3.0	Aston Villa	

	xg	…	day_code	target	gf_rolling	ga_rolling	sh_rolling	sot_rolling	\
0	0.4	…	6	1	2.000000	1.333333	7.666667	3.666667	
1	0.9	…	5	0	1.666667	1.666667	5.333333	3.666667	
2	0.9	…	6	0	1.000000	1.666667	7.000000	3.666667	
3	1.1	…	6	1	0.666667	1.000000	9.666667	4.000000	
4	1.5	…	6	0	0.333333	0.666667	9.666667	2.666667	

	dist_rolling	fk_rolling	pk_rolling	pkatt_rolling
0	14.733333	0.666667	0.000000	0.000000
1	15.766667	0.000000	0.000000	0.000000
2	16.733333	0.666667	0.000000	0.000000
3	16.033333	1.000000	0.000000	0.000000
4	18.033333	1.000000	0.333333	0.333333

[5 rows x 39 columns]

[9]:
```
#Retraining our model
from sklearn.metrics import precision_score
rf = rf_entropy
def make_predictions(data, predictors):
    train = data[data["date"] <='2021-12-31' ]
    test = data[data["date"] >'2022-01-01' ]
    rf.fit(train[predictors], train["target"])
    preds = rf.predict(test[predictors])
    combined = pd.DataFrame(dict(actual = test["target"], prediction = preds))
    precision = precision_score(test["target"], preds)
    return combined, precision

combined, precision = make_predictions(matches_rolling, predictors + new_cols)
combined = combined.merge(matches_rolling[["date", "team", "opponent",␣
 ↪"result"]], left_index = True, right_index = True )
# merging different information
print(precision)
```

```
combined # check miss predictions
```

```
0.5918367346938775
```

[9]:
```
      actual  prediction       date                    team       opponent  \
55         0           0 2022-01-23                 Arsenal        Burnley
56         1           0 2022-02-10                 Arsenal         Wolves
57         1           0 2022-02-19                 Arsenal      Brentford
58         1           0 2022-02-24                 Arsenal         Wolves
59         1           1 2022-03-06                 Arsenal        Watford
…        …          …         …                       …             …
1312       1           0 2022-03-13  Wolverhampton Wanderers        Everton
1313       0           0 2022-03-18  Wolverhampton Wanderers   Leeds United
1314       1           0 2022-04-02  Wolverhampton Wanderers    Aston Villa
1315       0           0 2022-04-08  Wolverhampton Wanderers  Newcastle Utd
1316       0           0 2022-04-24  Wolverhampton Wanderers        Burnley

     result
55        D
56        W
57        W
58        W
59        W
…      …
1312      W
1313      L
1314      W
1315      L
1316      L

[276 rows x 6 columns]
```

[10]:
```
class MissingDict(dict):
    __missing__ = lambda self, key:key
maps_values = {
    "Brighton and Hove Albion": "Brighton",
    "Manchester United" : "Manchester Utd",
    "Newcastle United" : "Newcastle Utd",
    "Totenham Hotspur" : "Tottenham",
    "West Ham United" : "West Ham",
    "Wolverhampton Wanderers" : "Wolves"
}
mapping = MissingDict(**maps_values)
combined["new_team"] = combined["team"].map(mapping)
```

```
merged = combined.merge(combined, left_on=["date", "new_team"],␣
 ↪right_on=["date", "opponent"])
merged[(merged["prediction_x"] == 1) & (merged["prediction_y"] ==␣
 ↪0)]["actual_x"].value_counts()

#Created the way more accurate prediction than the previous one
```

[10]:
```
actual_x
1    21
0    13
Name: count, dtype: int64
```

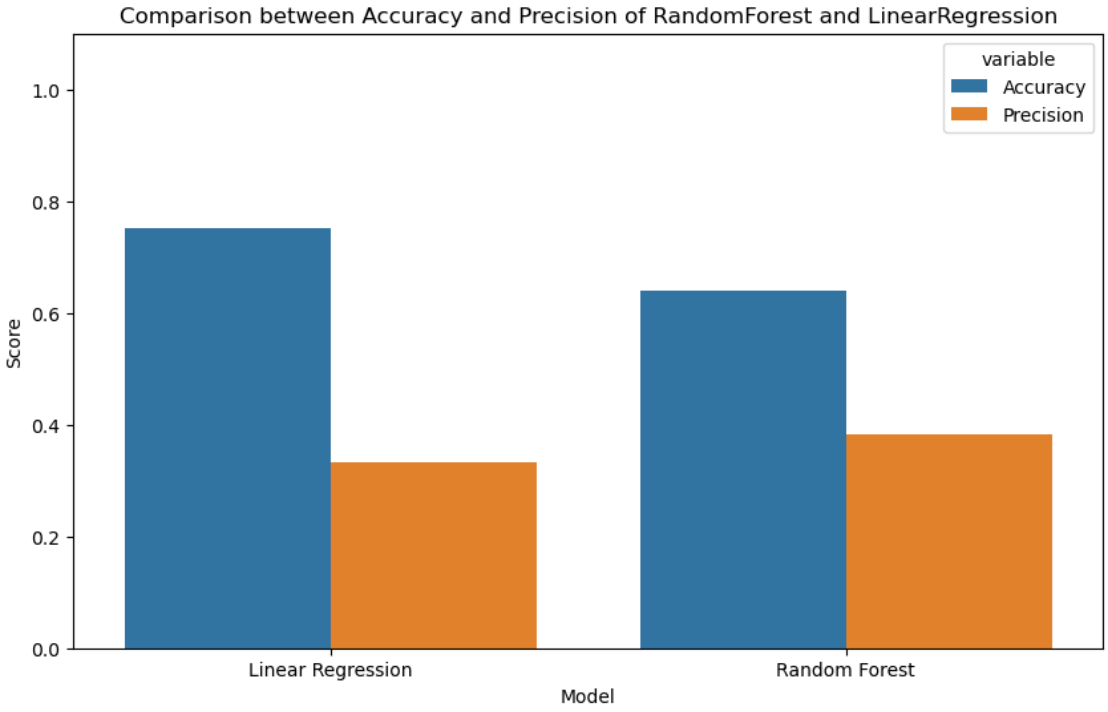
[11]:
```
from sklearn.metrics import mean_squared_error, accuracy_score, precision_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Split the data
train_val, test = train_test_split(matches, test_size=0.2, random_state=100,␣
 ↪shuffle=False)
train, val = train_test_split(train_val, test_size=0.25, random_state=100,␣
 ↪shuffle=False)

predictors = ["venue_code", "opp_code", "hour", "day_code"]

# Training Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(train[predictors], train["target"])

# Predictions for Linear Regression
y_pred_lin_reg_train = lin_reg.predict(train[predictors])
y_pred_lin_reg_test = lin_reg.predict(test[predictors])
y_pred_class_lin_reg_test = [1 if i > 0.5 else 0 for i in y_pred_lin_reg_test]
y_pred_lin_reg_val = lin_reg.predict(val[predictors])

# Metrics for Linear Regression
mse_lin_reg_train = mean_squared_error(train["target"], y_pred_lin_reg_train)
mse_lin_reg_test = mean_squared_error(test["target"], y_pred_lin_reg_test)
mse_lin_reg_val = mean_squared_error(val["target"], y_pred_lin_reg_val)
accuracy_lin_reg = accuracy_score(test["target"], y_pred_class_lin_reg_test)
precision_lin_reg = precision_score(test["target"], y_pred_class_lin_reg_test)

print("Linear Regression Results:")
print(f"\nTraining Set MSE: {mse_lin_reg_train:.2f}")
print("\nTest Set Results:")
print(f"Accuracy: {accuracy_lin_reg:.2f}")
print(f"Precision: {precision_lin_reg:.2f}")
print(f"MSE: {mse_lin_reg_test:.2f}")
```

```
print("\nValidation Set Results:")
print(f"MSE: {mse_lin_reg_val:.2f}")
```

```
Linear Regression Results:

Training Set MSE: 0.24

Test Set Results:
Accuracy: 0.75
Precision: 0.33
MSE: 0.21

Validation Set Results:
MSE: 0.25
```

[12]:
```
import matplotlib.pyplot as plt
import seaborn as sns



# For Linear Regression:
accuracy_lin_reg = accuracy_score(test["target"], y_pred_class_lin_reg_test) #␣
 ↪The accuracy obtained from your Linear Regression code
precision_lin_reg = precision_score(test["target"], y_pred_class_lin_reg_test)␣
 ↪# The precision obtained from your Linear Regression code

# For Random Forest:
accuracy_rf = val_acc_entropy # The accuracy obtained from your RandomForest␣
 ↪code
precision_rf =  precision_rf_entropy # The precision obtained from your␣
 ↪RandomForest code

# Preparing data for plotting
models = ['Linear Regression', 'Random Forest']
accuracy = [accuracy_lin_reg, accuracy_rf]
precision = [precision_lin_reg, precision_rf]

df = pd.DataFrame({
    'Model': models,
    'Accuracy': accuracy,
    'Precision': precision
})

# Melting DataFrame for easier plotting with seaborn
df_melted = pd.melt(df, id_vars=['Model'], value_vars=['Accuracy', 'Precision'])

# Plotting
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='value', hue='variable', data=df_melted)
plt.title('Comparison between Accuracy and Precision of RandomForest and␣
 ↪LinearRegression')
plt.ylabel('Score')
plt.ylim(0, 1.1)  # Assuming scores are between 0 and 1. Adjust if necessary.
plt.show()
```



[ ]: