Owen Waller

Owen.waller@hotmail.co.uk

Cricket score predICtor

Hawk-Eye Innovations Graduate Scheme Coding Task

**Stage 1: Data Collection and Preparation**

* 1. **Data selection**

Data was provided from cricsheet.org in the form of JSON files representing the events of each game. In order to ensure consistency, data from all male T20 matches was collected, as the consistent format would allow for better predictions. This included the following leagues:

* IPL
* NTB
* BPL
* PSL
* ILT
* CTC
* SSM
* CPL
* LPL
* BBL
* International T20s

This gave me a file with just over 7000 T20 games’ data.

* 1. **Data extraction**

Each JSON contained the events of the match, as well as metadata such as the date, teams, venue and players. In order to analyse and perform modelling on the data, it was important to get it in a more standardised format. The “Match\_Data\_Extractor.py” file was used to get the data from each match in the following format, with one row for each team in the match:

|  |  |
| --- | --- |
| **Field** | **Comments** |
| Match ID | Extracted from file name |
| Match Date | Date |
| Team |  |
| Opponent |  |
| Venue |  |
| Batting Order | First or second |
| Total Runs |  |
| Total Overs | Games often end before 20 over |
| Total Wickets |  |
| Run Rate | Runs per over |
| Boundary % | Number of hits which are 4s or 6s |
| Dot Ball% | Number of hits which result in no runs |
| Extras | Runs scored by methods other than batting |
| Bowlers Economy Rate | Number of runs conceded per over bowled |

* 1. **Data Cleansing**

The dataset was cleansed by removing any games with null fields (a low number due to the fairly standard data extracted from the files). Furthermore, any teams with name changes throughout the time period covered by the dataset were adjusted to ensure uniform name throughout.

Finally, any rows where a team didn’t reach 20 overs were removed, to allow for higher accuracy from the model.

* 1. **Feature Selection**

In order for a model to predict well, it must use relevant features in determining a team’s score. The following features were selected:

|  |  |
| --- | --- |
| **Feature** | **Explanation** |
| Average Score | Both lifetime and in last 5 games |
| Average Run Rate | Both lifetime and in last 5 games |
| Average wickets | Both lifetime and in last 5 games |
| Average Boundary % | Both lifetime and in last 5 games |
| Average Extras | Both lifetime and in last 5 games |
| Average Dot Ball % | Both lifetime and in last 5 games |
| Opponent Average Economy | The average economy rate of the opposing team |
| Average Score vs Opponent | Measures the team’s performance against this particular team |
| Average Run Rate vs Opponent | Measures the team’s performance against this particular team |
| Average Wickets vs Opponent | Measures the team’s performance against this particular team |
| Elo Rating | Measures a team’s general level – calculated after each match using the following formula:  New Elo = Current Elo + K(Actual Score - E)  Where E is an estimation of the score calculated by using the current elo ratings. |
| Opponent Elo Rating | The elo rating of the opponent before the game. |

This gives the model 16 predictors to use in its predictions.

* 1. **Data Preparation**

Now that the features have been selected and the dataset generated (using Get Features.py), the dataset must be prepared to be inputted into any prediction model. First of all, any records with null data must be removed – this is likely to be team’s first matches vs any opponent, as the averages are not yet calculated. This leaves just over 5000 records in the dataset.

The data must also be normalised to prevent certain features having a disproportionate impact on the model’s predictions, and 0.1-0.9 normalisation was used in this case.

**Stage 2: Model Implementations**

**2.1 Linear Regression**

First of all, I experimented with a linear regression. Although this model is unlikely to be accurate for such complex data and a high number of features, it can reveal which features are most impactful.

The model performed poorly as expected, with an r^2 of only 0.2 – extremely inaccurate.

**2.2 Random Forest**

The random forest model often captures relationships between the features well in sports prediction, and as such is a very good choice for the predictive model required here. Random forest works by using ensemble learning between decision trees to reach a single output.

Despite its suitability to the task, the random forest model still struggled in traditional performance metrics, with a high RMSE of 21.7 and an R^2 of only 0.43.

However, in terms of predicting the winner of the match, the model performed very well, correctly guessing **81%** of the time (2040/2524 matches). This suggests that although the model struggles to capture the randomness of cricket scores themselves, it captures team quality very well and as such offers utility in this way. In comparison, the linear regression model could only predict the winner 51% of the time – essentially equivalent to a coin toss.

**2.3 XGBoost**

The XGBoost model works similar to random forest, using the gradient boosted trees algorithm, however trains the model sequentially rather than concurrently. It resulted in similar performance, with a low R^2 score but correctly predicting the winning team **79% of** the time.