**The IPL Cricket Match Outcome Prediction Using Data Mining and Machine Learning**

**Ayush TRIPATHI1\*, Rashidul ISLAM 2, Vatsal KHANDOR3, Vijayabharathi MURUGAN4**

*1Department of computer Science, Raj Kumar Goel Institute of Technology, Ghaziabad*

*2 Heritage Institute of Technology, Kolkata*

*3 Dwarkadas J. Sanghvi College of Engineering Vile Parle, Mumbai*

*4Indian Institute of Technology, Madras*

**(\*Corresponding author’s e-mail: ayushtr0904@gmail.com)**

*Received: xxx, Revised: xxx, Accepted: xxx*

**Abstract**

The IPL (Indian Premier League) is one of the most viewed cricket matches in the world. With a perpetual increase in the popularity and advertising associated with it, forecasting the IPL matches' conclusion is becoming a need for the advertisers and the sponsors. This paper is centered on the implementation of machine learning to foretell the winner of an IPL match. The cricket in the T-20 format is highly unpredictable. Many features contribute to the result of a cricket match, and each attribute feature has a weighted impact on the outcome of a game.

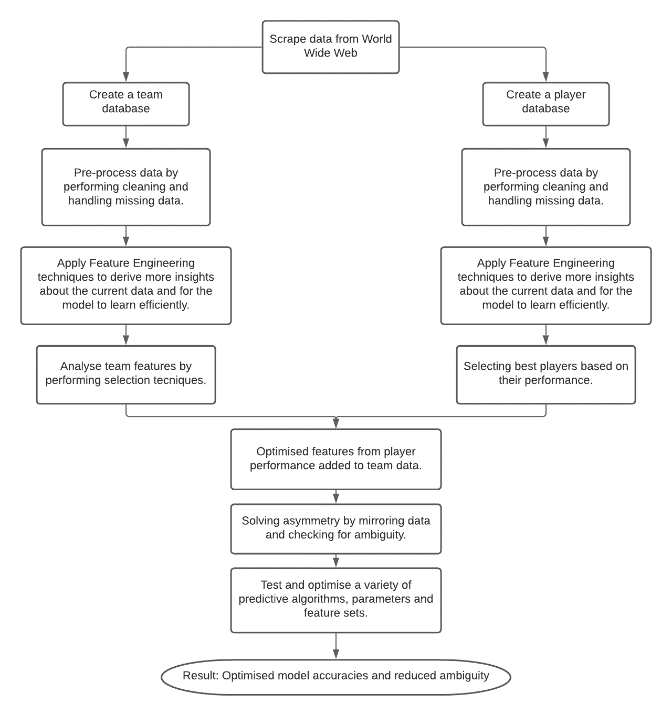
Similarly, a team's past performance adds to the possibility of winning a match against the opposing team. In this paper, through data mining, we define a meaningful dataset and further derived essential features using various methods like feature engineering and Analytic Hierarchy Process. Also, different machine learning algorithms were used to create a model for predicting the winner. The performance of the model developed was evaluated using various classification techniques. As per our investigation, the tree-based classifiers provided better results with our derived model.

**Keywords:** The Indian Premier League, Data Mining, Machine Learning, Analytic Hierarchy Process

**Introduction**

The IPL (Indian Premier League) is a 20-20 cricket league in India where eight teams (representing eight cities in India) play against each other. The IPL is India's biggest cricket festival - the most celebrated and viewed events, where the action is just not limited to the cricket field. Apart from national and global broadcasts, the matches are transmitted to regional channels in eight different languages. The jingles, promotional events, cheerleaders, advertising, fan clubs, interactions, betting, are all celebrated along with the players and the matches.

The brand value of the IPL was ₹475 billion (US$6.7 billion) in 2019 (as per Duff & Phelps). The entire revenue cycle of the IPL revolves around advertising. IPL also utilizes television timeouts, and there are other humongous opportunities associated with advertising. The IPL cricket league has proved to be a 'game-changer' for both Cricket and the entire Indian advertising industry. Hence, there are tremendous opportunities associated with predicting the outcome of an IPL match.

In this paper, we explored and develop machine learning models to predict the IPL matches. Figure 1 illustrates the entire process followed while conducting the research. During the research, we followed a multi-step approach to gather and pre-process the historical data. We then trained the models using multiple

Machine Leaning algorithms such as Logistic Regression, Naïve Bayes, Adaboost, Xgboost, and ExtraTreesClassifiers to develop a predictive model.

We discovered and dealt with an anomaly that we faced while switching the data between the tables during the process.

The highest accuracy was observed with Random Forest i.e. 60.043 with a standard deviation of 6.3%.

These derived models can be utilized to predict the results of the IPL matches. They can be further be used by the brands, sponsors, advertisers to keep up their marketing and advertising strategies.

*Figure 1: Process followed for the research*

**Related works**

Many researchers have contributed towards predicting the results of cricket matches. For instance, Rabindra Lamsal and Ayesha Choudhary [1] have proposed a Machine Learning model that predicts the outcome of an IPL match. The authors acquired the dataset available for all the 11 seasons in the archives at the official website of the IPL. They have also applied the concept of Multivariate Linear Regression to calculate the strengths of a team using the data from the Player Points section of the Indian Premier League website [2]. For prediction, the authors utilized various classifiers, namely Naive Bayes, Extreme Gradient Boosting, Support Vector Machine, Logistic Regression, Random Forests, and Multilayer perceptron.

The authors Madan Gopal Jhanwar and Vikram Pudi [3], in their research, adopted team composition method to predict the outcome of an ODI match. To calculate the player’s strength and aggregate to finalise the team strength, the authors have used the data career statistics of the players (both recent and overall performance). Other features like venue and toss were included. The model derived from their research gives the best result with KNN algorithm.

**Materials and methods**

**1. Dataset**

* 1. **Dataset Gathering**

The historical dataset was obtained from Kaggle [4], ESPN Cricinfo [5], Wikipedia [6], and iplt20 [2]. The performance data of individual players was taken from the ESPN Cricinfo website [5], which was used to calculate the strength of each player and the team. The data was then scrapped using Python Library Beautiful Soup. The extracted data demonstrated 26 features -including batting and bowling performances of the players. The match results dataset was acquired from Kaggle [3]. This data set illustrated 18 features. The IPL winning point table yearly data was accumulated from the IPL website [2].

* 1. **Pre-processing of Data** 
     1. **Conversion of Data Format (Label Encoding)**

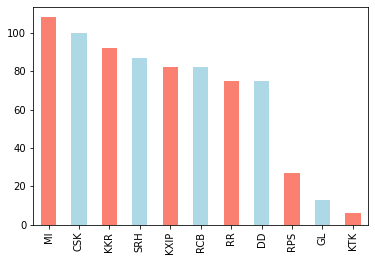
Most of the Machine Learning algorithms work better with numerical values than the string values. Hence, we converted the string values in the dataset to the numerical using the Label Encoding. The features that were converted are Team Names, Venue Names, Winning Team Name, Toss Winner Team Name.

* + 1. **Data Cleaning and dealing with null values**

Additionally, to produce accurate results, we then eliminated all the unnecessary features from the dataset. For example – Umpire Name, Stadium Name, Date, Dl applied, Player of the match. We also excluded the features that could result in data leakage. For example - Win by Runs and Win by Wickets. Further, we eradicated all the match rows without results (dismissed, drawn or null).

* 1. **Class Imbalance**

Class Imbalance is a problem in machine learning where the class distribution is highly imbalanced i.e. the number of instances in one class far exceeds the other.



*Figure 2: Historical data of the number of times each team has won matches in IPL*

During the process, we noticed that predicting the results with the team name is not feasible due to a massive Class Imbalance between the teams. For Example - MI won more than 100 matched whereas KTK only won less than 10. Refer Figure 1.

Hence, we decided to design our model based on features instead of the Team names to predict the winner - either Team 1 or Team 2 will be the winner.

Further, we also discovered that the number of times Team 1 is the winners are more than Team 2. To resolve this imbalance issue and balance the Team 1 winning and Team 2 winning in the label column, we interchanged a few Team 1 with Team 2.

* 1. **Assumptions**

We followed a few assumptions to make our model accurate and robust. For a few teams, the names were changed (due to legal actions or the team owner change), but as the players and team dynamics didn’t change, we kept them as the same team as before while conducting the research.

The name Delhi Capitals was changed to Delhi Daredevils, Deccan Chargers to Sunrisers Hyderabad, and Pune Warriors to Rising Pune Supergiant – but we assumed them as the same team as it was before the change in their names.

We have considered the data of only 11 players for a particular team based on the number of matches they played.

1. **Feature Engineering**
   1. **Base Features**

**2.1.1. Features from the processed data**

From the gathered and processed data, we extracted the following three meaningful features for predicting the winner.

|  |  |  |
| --- | --- | --- |
| ***City*** | ***Toss Winner*** | ***Toss Decision*** |

**2.1.2. Dream11 Strength Calculation**

In the first approach, we referred to the Dream11 points table (below) to derive our formula.

**Score Point Table:**

|  |  |  |
| --- | --- | --- |
| **Notation** | **Type of Points** | **Weight** |
| φMatches | Being a part of the starting XI | 4 |
| φRuns\_Scored | Every run scored | 1 |
| Φctchs | Catch taken | 8 |
| Φfifties | Total number of 50s | 8 |
| Φhundreds | Total number of 100s | 16 |
| Φnum\_4s | Total number of 4s scored | 1 |
| Φnum\_6s | Total number of 6s scored | 2 |
| φstmp | Stumping/ Run Out (direct) | 12 |
| Φr\_out | Run Out (Thrower/Catcher) | 8/4 |
| Φfduck | Dismissal for a Duck (only for batsmen, wicket-keepers and all-rounders) | -2 |
| Φbat\_innings | Number of times a player has batted in a match |  |
| Φbowl\_innings | Number of times a player has bowled in a match |  |
| Φwickets | Number of wickets taken by a bowler in the season | 25 |
| Φmaidens | Number of times a bowler has bowled an over without conceding any runs | 8 |
| Φ4\_wicket\_houl | Number of times a player has taken 4 wickets in a single match | 8 |
| Φ5\_wicket\_houl | Number of times a player has taken 5 wickets in a single match | 16 |
| Φbowl\_economy | Bowling economy of a player |  |
| Φbat\_strike\_rate | Batting Strike Rate of a player |  |
| Φmax\_matches | Maximum matches played by a team |  |

*Table 1: Score Point*

**Bonus Points:**

|  |  |
| --- | --- |
| Type of Points | **Weight** |
| Every boundary hit | 1 |
| Every six-hit | 2 |
| Half-Century (50 runs scored by a batsman in a single inning) | 8 |
| Century (100 runs scored by a batsman in a single inning) | 16 |
| Maiden Over | 8 |
| 4 wickets | 8 |
| 5 wickets | 16 |

*Table 2: Bonus points*

**Economy Rate:**

|  |  |
| --- | --- |
| Type of Points | **Weight** |
| Minimum overs bowled by a player to be applicable | 2 overs |
| Between 6 and 5 runs per over | 2 |
| Between 4.99 and 4 runs per over | 4 |
| Below 4 runs per over | 6 |
| Between 9 and 10 runs per over | -2 |
| Between 10.01 and 11 runs per over | -4 |
| Above 11 runs per over | -6 |

*Table 3: Economy rate*

**Strike Rate (except Bowlers):**

|  |  |
| --- | --- |
| Type of Points | **Weight** |
| Minimum balls faced by a player to be applicable | 10 balls |
| Between 60 and 70 runs per 100 balls | -2 |
| Between 50 and 59.99 runs per 100 balls | -4 |
| Below 50 runs per 100 balls | -6 |

*Table 4: Strike Rate*

**Batting Score of a player:**

**Input:** Players p ∈{P(A,m)∪P(B,m)}, Career Statistics of player p: φ(p)

**Output:** Batsmen Score of all the players: φBatsman Score

1: for all players p ∈{P(A,m)∪P(B,m)} do

2: φ ← φ(p)

3: u ← (1\* ΦRuns\_Scored) +(1\*Φnum\_4s) + (2\*Φnum\_6s) + (8\*Φfifties) + (16\*Φhundreds) - (2\*Φfduck)

4: if Φbat\_strike\_rate < =50:

v ← -6

else if Φbat\_strike\_rate > 50 and Φbat\_strike\_rate < =60:

v ← -4

else if Φbat\_strike\_rate > 60 and Φbat\_strike\_rate < = 70:

v ← -2

endif

4: w ← v \* Φbat\_strike\_rate\* Φbat\_innings

5: y ← u + w

6: φBatsman Score ← y

7: end for

**Bowling Score of a player:**

**Input:** Players p ∈{P(A,m)∪P(B,m)}, Career Statistics of player p: φ(p)

**Output:** Bowling Score of all the players: φBowling\_Score

1: for all players p ∈{P(A,m)∪P(B,m)} do

2: φ ← φ(p)

3: u ← (25\*Φwickets) + ( 8\*Φctchs) + (12\*Φstmp) + (8\*Φ4\_wicket\_haul ) + (16\*Φ5\_wicket\_haul) + (8\*Φmaidens)

4: if Φbowl\_economy < = 6 and Φbowl\_economy > 5:

v ← 2

else if Φbowl\_economy > 4 and Φbowl\_economy < = 5:

v ← 4

else if Φbowl\_economy < = 4:

v ← 6

else if Φbowl\_economy > = 9 and Φbowl\_economy < 10:

v ← -2

else if Φbowl\_economy >= 10 and Φbowl\_economy < 11 :

v ← -4

else if Φbowl\_economy >= 11:

v ← -6

endif

4: w ← v \* Φbowl\_economy\* Φbowl\_innings

5: y ← u + w

6: φBowling\_Score ← y

7: end for

**Total Score of a player:**

**Input:** Players p ∈{P(A,m)∪P(B,m)}, φBowling\_Score, φBatsman\_Score

**Output:** Total Strength: φ Total \_Strength

1: for all players p ∈{P(A,m)∪P(B,m)}, φBowling\_Score, φBatting\_Score do

2: φ Total\_Strength ← (φBowling\_Score + φBatsman\_Score)/ φtot\_matches

3: endfor

**Team Strength**

**Input:** Top 11 Players p ∈{P(A,m)∪P(B,m)}

**Output:** Team Strength: φTeam \_Strength

1: for all players p ∈{P(A,m)∪P(B,m)}, φTotal\_Strength do

7: φTeam \_Strength ← φTotal\_Strength/Φmax\_matches

9: endfor

**Cumulative Team Strength**

For a particular year, the Team Strength represents the previous year performance, whereas the Cumulative Team Strength represents the mean of the strength of all the previous years. For example – for Mumbai Indians in 2016: the strength will be 2015 strength and cumulative will be the mean of 2008 -2015.

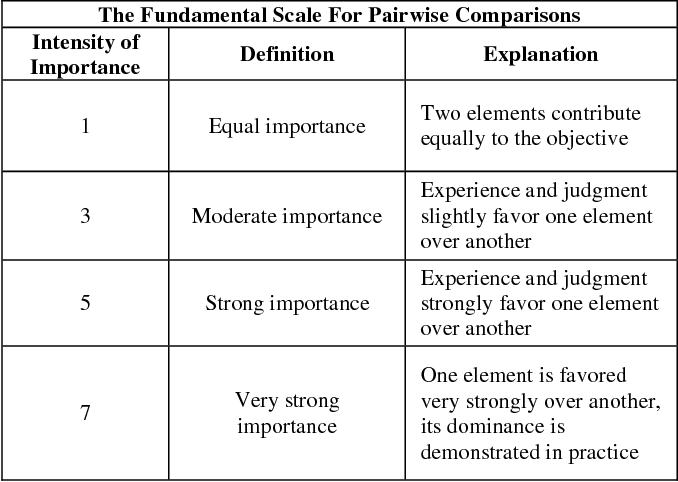
**From this section, we have collected eight significant features**:

|  |  |  |  |
| --- | --- | --- | --- |
| Team\_1\_BattingStrength | Team\_1\_BowlingStrength | Team\_1\_Strength | Team\_1\_CumulativeStrength |
| Team\_2\_BattingStrength | Team\_2\_BowlingStrength | Team\_2\_Strength | Team\_2\_CumulativeStrength |

**2.1.2. Analytic Hierarchy Process for Strength Calculation**

Different measures highlight different aspects of a player's ability; hence some features are essential compared to others. For example – The strike rate is an important feature for any game - especially T20, as the number of overs is less, and this feature adds to the team's ability to score maximum runs. Hence, we weighted the measures according to their relative importance over other measures (features). We have used the Analytic Hierarchy Process (AHP) to determine these weights for each player to calculate their bowling and batting measures. We calculated the weights for each team based on their past performance.

The Analytic Hierarchy Process is a method for decision-making in complex conditions in which many variables or criteria are considered in prioritizing and selecting options. AHP subdues complex decisions into a series of pairwise comparisons and captures both subjective and objective aspects of a decision. It generates a weight for each evaluation criterion. The higher the weight for a corresponding criterion, the more important is the corresponding criterion. Finally, the AHP combines the criteria weights and the options amounts, thus determining a global score for each option, and a consequent ranking. The global score for a given option is a weighted sum of the scores it obtained with respect to all the criteria.



*Figure 5: The fundamental scale for pairwise comparisons*

**Batting AHP**

**Priority Order**

From our knowledge of T20 cricket statistics and experience, we arrange the attributes in their decreasing order of importance as:

|  |
| --- |
| Batting Average > Innings > Strike Rate > 50’s > 100’s > 0’s |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Average** | **INN** | **SR** | **50's** | **100's** | **0's** |
| **Average** | 1 | 2 | 3 | 5 | 6 | 7 |
| **INN** | 0.5 | 1 | 2 | 4 | 5 | 6 |
| **SR** | 0.333333 | 0.5 | 1 | 3 | 4 | 5 |
| **50's** | 0.2 | 0.25 | 0.333333 | 1 | 2 | 3 |
| **100's** | 0.166667 | 0.2 | 0.25 | 0.5 | 1 | 2 |
| **0's** | 0.142857 | 0.166667 | 0.2 | 0.333333 | 0.5 | 1 |

*Table 5:*

Finally, we get following weights for the attributes: Average: 0. **388726074,** Innings: 0. **260099468,** Strike Rate: 0. **175428513,** Fifties: 0. **083438652,** Centuries: 0. **055018035,** Zeros: 0. **037289258**

|  |
| --- |
| *AHP bat = 0.388726074 \* Average + 0.260099468 \* Innings + 0.175428513 \* Strike Rate + 0.083438652 \* 50’s + 0.055018035 \* 100’s - 0.037289258 \* 0’s* |

**Bowling AHP**

**Priority Order**

From our knowledge of T20 cricket statistics and experience, we arrange the attributes in their decreasing order of importance as:

|  |
| --- |
| Overs > Economy > Wickets > Bowling Average > Bowling Strike Rate > 4W Haul |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Overs** | **Economy** | **Wickets** | **Bowling Avg** | **Bowling SR** | **4W Haul** |
| **Overs** | 1 | 2 | 4 | 6 | 6 | 7 |
| **Economy** | 0.5 | 1 | 4 | 5 | 5 | 6 |
| **Wickets** | 0.25 | 0.25 | 1 | 4 | 4 | 6 |
| **Bowling Avg** | 0.166666 | 0.2 | 0.25 | 1 | 1 | 5 |
| **Bowling SR** | 0.166666 | 0.2 | 0.25 | 1 | 1 | 4 |
| **4W Haul** | 0.142857 | 0.166666 | 0.166666 | 0.2 | 0.25 | 1 |

*Table 6:*

Finally, we get following weights for the attributes: **Overs:** 0.4174 **Innings:** 0.2634 **Strike Rate:** 0.1602 **Bowling Average:** 0.0975 **Four Wickets Haul:** 0.0615

|  |
| --- |
| *AHP Ball = 0.4174 \* Overs + 0.2634 \* Innings+0.1602 \* Strike Rate + 0.0975 \* Bowling Average + 0.0615 \* Four Wickets Haul* |

**We have formed six features from this section:**

|  |  |  |
| --- | --- | --- |
| *Team\_1\_AHP\_BAT* | *Team\_1\_AHP\_Ball* | *Team\_1\_AHP\_BAT+Ball* |
| *Team\_2\_AHP\_BAT* | *Team\_2\_AHP\_Ball* | *Team\_2\_AHP\_BAT+Ball* |

**2.1.3. Rank Calculation using AHP**

Using the AHP, we derived the coefficient for the win rate of each team against the other.

Assumption: We dropped KTK and GL Teams while calculating the weights because they never played against each other.

**Priority Order**

Next, we calculated the priority order through AHP by with the data of the matches played for win/loss for each team against each team.

For example, the Team CSK (Chennai Super Kings) & MI (Mumbai Indians) played 27 matches against each other, and according to the dataset, MI won 16, and CSK won the rest 11 games. In this instance, in the MI row, the input will be 16/11 = 1.454545, and in the CSK row, it will be reciprocal, which is 11/16 or 1/1.4545454 = 0.6875.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **CSK** | **DD** | **KKR** | **KXIP** | **MI** | **RCB** | **RPS** | **RR** | **SRH** |
| **CSK** | 1 | 2.5 | 1.857143 | 1.333333 | 0.6875 | 2.142857 | 2 | 2 | 2.142857 |
| **DD** | 0.4 | 1 | 0.769231 | 0.642857 | 1 | 0.571429 | 1.25 | 0.727273 | 1 |
| **KKR** | 0.538462 | 1.3 | 1 | 2.125 | 0.315789 | 1.4 | 3.5 | 1 | 1.888889 |
| **KXIP** | 0.75 | 1.555556 | 0.470588 | 1 | 0.846154 | 1 | 1 | 0.9 | 0.846154 |
| **MI** | 1.454545 | 1 | 3.166667 | 1.181818 | 1 | 1.777778 | 1.4 | 1 | 1.181818 |
| **RCB** | 0.466667 | 1.75 | 0.714286 | 1 | 0.5625 | 1 | 3.5 | 0.7 | 0.785714 |
| **RPS** | 0.5 | 0.8 | 0.285714 | 1 | 0.714286 | 0.285714 | 1 | 0.25 | 0.666667 |
| **RR** | 0.5 | 1.375 | 1 | 1.111111 | 1 | 1.428571 | 4 | 1 | 1.5 |
| **SRH** | 0.466667 | 1 | 0.529412 | 1.181818 | 0.846154 | 1.272727 | 1.5 | 0.666667 | 1 |

*Table 7:*

We then found the ranks of each team yearly based on win ratios among the team. The below are the ranks derived via AHP.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Team names* | *RPS* | *DD* | *SRH* | *GL* | *KTK* | *RCB* | *KXIP* | *RR* | *KKR* | *MI* | *CSK* |
| *Coefficients* | *0.604341* | *0.809025* | *0.904243* | *1* | *1* | *1* | *0.939665* | *1.27174* | *1.27679* | *1.518754* | *1.693047* |
| *Ranks* | *9* | *8* | *7* | *5* | *5* | *5* | *6* | *4* | *3* | *2* | *1* |

For KTK and GL, we took the mean value which is 1 as the coefficients. During the process, we determined the below two features:

**We have formed two features from this section:**

|  |  |
| --- | --- |
| *Team\_1\_Rank* | *Team\_2\_Rank* |

**2.1.4. Win Rate**

Win rate is an essential factor in determining the result of a game, especially a cricket match, in which the win rate almost determines the overall performance for a team.

If a team is continuously winning the matches against other teams, it is a sign that the team's form is good, and the probability of the team winning the subsequent matches is higher. On the other hand, a losing team reflects that it is not in good form and may even lose games further.

**Win rate calculation:**

In the next steps, we crawled and checked the entire IPL match list played every year by each team from 2008 to 2019. If the two teams were playing against each other for the first time, we reset the win rate to 0 for both the teams. Subsequently, as the team started playing matches against each other, we calculated the previously played matches and checked the winners for those occurrences. Then we defined a ratio for each team. This ratio is an important criterion in determining the result of the game.

The win rate ratio of a team reflects its past performance and can be analyzed to predict the future. The higher the ratio, the better are the chances for a team to win. For a match, we have taken the past win rate ratio of the team as below:

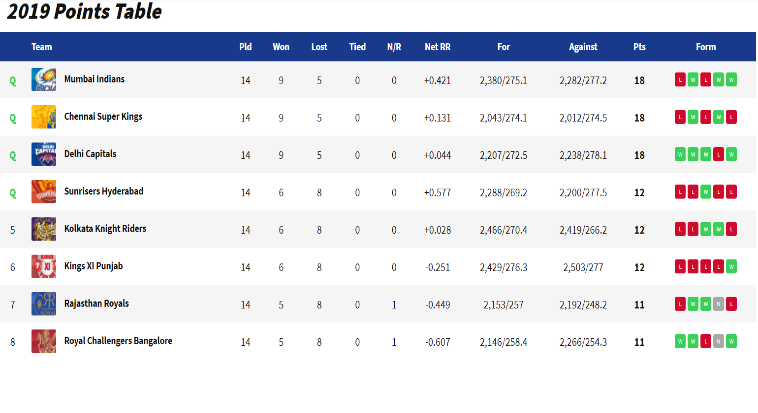
|  |
| --- |
| *Φwin\_rate(Match R) = Total Number of wins till match R-1/ Total Number of matches played till R-1* |

We derived the below two features:

**We have formed two features from this section:**

|  |  |
| --- | --- |
| *Team\_1\_Win\_Rate* | *Team\_2\_Win\_Rate* |

**2.1.5. Team Points**

 IPL is a league tournament based on a point system. Every year, two teams play against each other twice before they enter the semi-final match, if not eliminated. The point table comprises teams, match won/lost/tied, and net run rate. The ranking of the teams was done according to the points. We fed this past performance features of the teams to our model to predict the results.

*Figure 6: The IPL Match Teams Point table*

**We have formed two features from this section:**

|  |  |  |  |
| --- | --- | --- | --- |
| *Team\_1\_Point* | *Team\_2\_Point* | *Team\_1\_Cumulative Point* | *Team\_2\_Cumulative Point* |

For a particular year, the team point represents the previous year’s performance, whereas the cumulative team point represents the mean of the points of all the previous years.

**2.2. Intersection Features**

**Consistency** **[Multi collinearity is the natural question that comes here]**

The consistency of a team adds more weightage to its current performance than the overall performance. The current performance of a team has been given 80 percent weightage while the overall performance is given 20 percent weightage.

|  |
| --- |
| *Team 1 Consistency = (Team 1 Strength \* 0.8 + Team 1 Cumulative Team Strength \*0.2)/2*  *Team 2 Consistency = (Team 2 Strength \* 0.8 + Team 2 Cumulative Team Strength \*0.2)/2* |

**We have formed two features from this section:**

|  |  |
| --- | --- |
| *Team\_1\_Consistency* | *Team\_2\_Consistency* |

**Win Rate Strength** **[We should certainly consider renaming this Feature]**

The individual strength of a team represents how strong a team is by considering the stats. However, there are various other factors that impact the winning of a team - for example - playing sequence of a team, performance as a team, sentiments of the audience. We captured this information by multiplying the strength with the previous win rate of the team.

|  |
| --- |
| *Team 1 Win Strength =Team 1 Win Rate \* Team 1 Strength*  *Team 2 Win Strength =Team 2 Win Rate \* Team 2 Strength*  *Team 1 Win Cumulative Strength =Team 1 Win Rate \* Team 1 Cumulative Strength*  *Team 2 Win Cumulative Strength =Team 2 Win Rate \* Team 2 Cumulative Strength* |

|  |  |  |  |
| --- | --- | --- | --- |
| *Team\_1\_WinStrength* | *Team\_2\_WinStrength* | *Team\_1\_Win\_Cumulative Strength* | *Team\_2\_Win\_Cumulative Strength* |

**2.3. Transformation Features**

With all the base and Intersection features we have developed the transformed features. These features are created by subtracting two base features or intersection features from the same category. For example: Team1\_Team\_Strength is subtracted from the Team2\_Team\_Strength to create the new feature.

Since we created a lot of new features based on base and intersection features for our model, multicollinearity can occur. Multicollinearity occurs when multiple features in a model are highly linearly related, which means one variable can be predicted quite accurately using the other variable. The problem with multicollinearity is that it causes the model to overfit.

To deal with multicollinearity in our model we dropped all the base and intersection features which were used to create the new features.

**2.4. Solving the Asymmetry - Mirroring the data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Team1*** | ***Team 2*** | ***Team1\_Strength*** | ***Team2\_Strength*** | ***Winner*** | ***Team1*** |
| *CSK* | *MI* | *X* | *Y* | *1* | *CSK* |

When we consider the above row, it is apparent to human that while switching between TEAM1 & TEAM2 the results will be the same. However, to a machine-learning model, this is not apparent and hence we need to input this information.

We mirrored each row in the training set so that the model understands both the cases.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Team1*** | ***Team2*** | ***Team1\_Strength*** | ***Team2\_Strength*** | ***Winner*** |
| *CSK* | *MI* | *X* | *Y* | *1* |
| *MI* | *CSK* | *Y* | *X* | *0* |

Steps involved in creating the train and test sets are as below:

1. The original dataset is split using *sklearn train\_test\_split* into training and test sets
2. The training set is then mirrored as shown above and append to the original training set which result in the number of rows being doubled
3. The test set is also mirrored but we don't append them and create two test sets

**2.5. Model Ambiguity**

The asymmetric nature of the model in a few rows leads to ambiguity in the results. For instance:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Test Set*** | ***Team1*** | ***Team2*** | ***Winner*** | ***Prediction*** |
| *1* | *KKR* | *KXIP* | *1* | *1* |
| *2* | *KXIP* | *KKR* | *0* | *1* |

So, in the above case, we have different predictions for same case. We will call such an occurrence as Model Ambiguity. Note that this is not the same as incorrect prediction as the prediction will count correct in either test set 1 or test set 2. Therefore, we will evaluate the model using 5 parameters rather than just training and test accuracy:

* Training Accuracy: % of correct predictions in mirror and merged train setTest 1 Accuracy: % of correct prediction in original test set
* Test 2 Accuracy: % of correct prediction in mirrored test set
* Real Test Accuracy: % of correct prediction after discrediting the scores for ambiguous rows
* Ambiguity: % of rows in which ambiguity is observed

**Discussion and Results**

* 1. **Data Set Split:**

We have noticed that by changing the random state in our dataset the accuracy defers a lot. It is because the training and testing dataset is randomly split based on the state which we give. In order to prevent such a scenario and to make our model robust we have used *RepeatedStratifiedKFold*.

We have selected 10 folds and 2 iterations. Thus, to give a total of 20 folds. In our dataset there are around 760 rows. We know that in each season of an IPL there are around 70 matches. So in order to make our testing dataset equal to 70 rows, we have used 10 folds.

We used *RepeatedStratifiedKFold* over *StratifiedKFold* as our dataset is small, and *RepeatedStratifiedKFold* gives more fold with larger validation set. In a scenario where data is less *RepeatedStratifiedKFold* is preferred more over *StratifiedKFold*, as it gives more fold with larger validation set.

**Constant:** We have taken Random State = 827 throughout the project

**Cohen’s kappa:**

Cohen’s kappa tells how much better our classifier is performing over the performance of a classifier that simply guesses at random according to the frequency of each class.

*KP<0 : NO AGREEEMENT*

*0 <= KP <= 0. 20 : SLIGHT AGREEMENT*

*0.21 <= KP <= 0. 40 : FAIR AGREEMENT*

*0.41 <= KP <= 0.60 : MODERATE AGREEMENT*

*0.61 <= KP <= 0.80 : SUBSTANTIAL AGREEMENT*

*0.81 <= KP <= 1 : PERFECT AGREEMENT*

|  |  |
| --- | --- |
| **AUROC:**  Area Under the receiver operating characteristics (AUROC) tells us how much a model is capable of distinguishing between classes. The higher the area under the curve the better the model is. |  |
| **Model Implementation using Naïve Bayes:**  Table 8 shows the best result from Naïve Bayes. The Real test accuracy that we got is 60.035 % with a standard deviation of 6.2%. Cohen Kappa score 0.1891 shows the Slight Agreement.   |  |  |  |  | | --- | --- | --- | --- | | Ambiguity | Real Test Accuracy | Train Accuracy | Cohen Kappa score | | 3.008 ± 2.5 % | 58.233 ± 5.5 % | 60.757 ± 0.9 % | 0.1891 |   *Table 8: best result from Naïve Bayes*  Figure 7shows the ROC curve with the best result that we got. The Area under the Curve is 0.63; it means there is 63% chance that the model will be able to distinguish between Team1 winner and Team2 winner. Figure 8 shows the distribution of Real Test Accuracy for Naïve Bayes Classifier.   |  |  | | --- | --- | |  |  | | *Figure 7: ROC curve with the best result* | *Figure 8: Real test accuracy distribution for Naïve Bayes* |   Kurtosis of the Real Test Accuracies: -0.7953726614792531  Skewness of the Real Test Accuracies: -0.2357171690268254  **Model Implementation using Logistic Regression**  We tuned our model over 56 combinations. Table 9 shows the different hyper parameters that we have used for tuning the Logistic Regression model.   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  | | --- | --- | | penalty | l2 | | solver | lbfgs | | max\_iter | 100, 200, 300, 400, 600, 900, 1200 | | tol | 0.0001, 0.00001, 0.0005, 0.001 | | |  |  | | --- | --- | | penalty | l2 | | solver | lbfgs | | max\_iter | 1200 | | tol | 0.001 | | C | 5 | | Ambiguity | 10.38473 | | Real Test Accuracy | 54.30109 | | Train Accuracy | 62.15782 | | Cohen Kappa score | 0.0833 | | | *Table 9: Hyperparameters* | *Table 10: Best result with hyperparameters* | |  |

**ROC Curve**

Figure 9 shows the ROC curve with the best result that we got. Figure 10 shows the distribution of Real Test Accuracy for Logistic Regression.

|  |  |
| --- | --- |
|  |  |
| *Figure 9: ROC curve with the best result* | *Figure 10: Real test accuracy distribution for Logistic Regression* |

**Model Implementation using ADABOOST**

We have tuned our model over 56 combinations. Table 11 shows the different hyper parameters that we have used for tuning the ADABOOST model. Table 12 shows the best result with the corresponding hyper-parameters. The Real test accuracy that we got is 60.035 % with a standard deviation of 6.2%.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | learning\_rate | 0. 005 ,0.01 ,0.02, 0.05,0.15, 0.5, 0.1,1 | | n\_estimators | 10,20,50,100,200,800,1000 |   *Table 11: Hyperparameters* | |  |  | | --- | --- | | learning\_rate | 0.01 | | n\_estimators | 150 | | Ambiguity | 0.402 ± 0.9 % | | Real Test Accuracy | 60.035 ± 6.2 % | | Train Accuracy | 62.127 ± 0.9 % | | Cohen Kappa score | 0.194 | |
|  | *Table 12: Best result with hyperparameters* |

**ROC Curve**

Figure 11 shows the ROC curve with the best result that we got. Figure 12 shows the distribution of Real Test Accuracy for AdaBoostClassifier.

|  |  |
| --- | --- |
|  |  |
| *Figure 11: ROC curve with the best result* | *Figure 12: Real test accuracy distribution for AdaBoostClassifier* |

Kurtosis of the Real Test Accuracies: -0.6020731047837842

Skewness of the Real Test Accuracies: -0.4676762875419866

**Model Implementation using XGBOOST**

We have tuned our model over 3600 combinations. Table 4 shows the different hyper parameters that we have used for tuning the XGBOOST model. Table 5 shows the best result with the corresponding hyper-parameters. The Real test accuracy that we got is 60.035 % with a standard deviation of 6.2%.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | learning\_rate | 0.05, 0.10,0.15, 0.25 | | n\_estimators | 50,100,200,500,700,1000 | | max\_depth | 4,5,6,7,8 | | gamma | 0,0.1, 0.2,0.3 | | min\_child\_weight | 1,3 | | colsample\_bytree | 0.3, 0.4, 0.7 |   *Table 13: Hyperparameters* | |  |  | | --- | --- | | learning\_rate | 0.05 | | max\_depth | 4 | | min\_child\_weight | 1 | | gamma | 0 | | colsample\_bytree | 0.3 | | n\_estimators | 100 | | Ambiguity | 7.098 ± 2.9 % | | Real Test Accuracy | 55.42 ± 5.9 % | | Train Accuracy | 78.079 ± 0.9 % | | Cohen Kappa | 0.228 | |
|  | *Table 14: Best result with hyperparameters* |

**ROC Curve**

Figure 4 shows the ROC curve with the best result that we got. Figure 5 shows the distribution of Real Test Accuracy for XGBOOST.

|  |  |
| --- | --- |
|  |  |
| *Figure 13: ROC curve with the best result* | *Figure 14: Real test accuracy distribution for XGBOOST* |

Kurtosis of the Real Test Accuracies: -0.8632662762048833

Skewness of the Real Test Accuracies: 0.045619882877041766

**Model Implementation using ExtraTreesClassifiers**

We tuned our model over 320 combinations. Table 7 shows the different hyper parameters that we have used for tuning the ExtraTreesClassifiers model. Table 8 shows the best result with the corresponding hyper-parameters. The Real test accuracy that we got is 60.035 % with a standard deviation of 6.2%.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | n\_estimators | 100,200,600,900,1200,1500,1800,2100 | | max\_depth | 3,4,5,12,15 | | max\_features | Sqrt, log2 | | min\_sample\_leaf | 3,5,8,12 |   *Table 15: Hyperparameters* | |  |  | | --- | --- | | n\_estimators | 2100 | | max\_depth | 12 | | max\_features | log2 | | min\_sample\_leaf | 12 | | Ambiguity | 4.286 ± 2.0 % | | Real Test Accuracy | 59.506 ± 5.9 % | | Train Accuracy | 74.71 ± 0.5 % | | Cohen Kappa | 0.1864 | |
|  | *Table 16: Best result with hyperparameters* |

**ROC Curve**

Figure 15 shows the ROC curve with the best result that we got. Figure 16 shows the distribution of Real Test Accuracy for Extra Trees Classifier.

|  |  |
| --- | --- |
| *Figure 15: ROC curve with the best result* | *Figure 16: Real test accuracy distribution for Extra TreesClassifier* |

Kurtosis of the Real Test Accuracies: -0.21205306662258305

Skewness of the Real Test Accuracies: 0.5901504901262185

**Model Implementation using Random Forest Classifier**

We have tuned our model over 1200 combinations. Table 4 shows the different hyper parameters that we have used for tuning the Random Forest Classifier model. Table 5 shows the best result with the corresponding hyper-parameters. The Real test accuracy that we got is 60.035 % with a standard deviation of 6.2%.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | bootstrap | True, False | | max\_depth | 3,4,5,6,7 | | max\_features | 0.5, 'sqrt','log2' | | min\_samples\_leaf | 2, 4 | | min\_samples\_split | 2, 5 | | criterion | Gini, entropy | | n\_estimators | 800,1000,1200,1600,2000 |   *Table 17: Hyperparameters* | |  |  | | --- | --- | | max\_features | 0.5 | | bootstrap | True | | max\_depth | 3 | | min\_samples\_leaf | 4 | | min\_samples\_split | 2 | | colsample\_bytree | 0.3 | | n\_estimators | 1200 | | Ambiguity | 1.404 ± 1.4 % | | Real Test Accuracy | 60.043 ± 6.3 % | | Train Accuracy | 65.978 ± 0.7 % | | Cohen Kappa | 0.1785 | |
|  | *Table 18: Best result with hyperparameters* |

**ROC Curve**

Figure 17 shows the ROC curve with the best result that we got. Figure 18 shows the distribution of Real Test Accuracy for Random Forest Classifier.

|  |  |
| --- | --- |
|  |  |
| *Figure 17: ROC curve with the best result* | *Figure 18: Real test accuracy distribution for Random Forest Classifier* |

Kurtosis of the Real Test Accuracies: -0.8606335574086965

Skewness of the Real Test Accuracies: -0.24911462664478376

**Conclusion and Future Works**

The research's focus was to predict the IPL matches using machine learning utilizing the available historical data of IPL from season 2008-2019. In the process, various branches of Data Science were adopted to conduct the study, including data mining, visualization, preparation of database, feature engineering, applying the Analytic hierarchical process, creating prediction models, and training classification techniques.

The IPL dataset was gathered and preprocessed. The missing values were removed and variables were encoded into the numerical format to make the dataset uniform.

The essential features were then derived from data using the domain knowledge to extract raw data features via data mining techniques.

Several machine learning models were applied to the selected features to predict the IPL match results. The best results were concluded using the tree-based classifiers. We observed the highest accuracy of 60.043% with Random Forest with a standard deviation of 6.3%.

While calculating the player’s strength we have taken the series wise performance of the player rather than their match wise performance. To further develop this research, a more exhaustive approach can be taken by also adding the match wise data into the research. Also, the research cab be further enhanced by adding other factors like comparing players’ performances at a particular stadium.

**References**

1. <https://en.wikipedia.org/wiki/Indian_Premier_League>

2.<https://www.researchgate.net/publication/327904009_Predicting_Outcome_of_Indian_Premier_League_IPL_Matches_Using_Machine_Learning>  
3. <https://www.iplt20.com>  
4. <https://dtai.cs.kuleuven.be/events/MLSA16/slides/06_Madan_Gopal.pdf>  
5. <https://www.kaggle.com/manasgarg/ipl>  
6. <https://stats.espncricinfo.com/>  
7. <https://www.dream11.com/games/point-system>  
8. <https://en.wikipedia.org/wiki/Analytic_hierarchy_process>  
9. <https://en.wikipedia.org/wiki/Analytic_hierarchy_process_%E2%80%93_leader_example>  
10. <https://www.iplt20.com/stats/2019>  
11. <https://www.theanalysisfactor.com/linear-regression-analysis-3-common-causes-of-multicollinearity-and-what-do-to-about-them/>  
12. <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedStratifiedKFold.html>  
13. <https://www.statisticshowto.com/cohens-kappa-statistic/>  
14. <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>

**AHP References**

**M. G. Jhanwar and V. Pudi (2016),** "Predicting the Outcome of ODI Cricket Matches: A Team Composition Based Approach," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases.*

**Bandulasiri, A. (2008).** “Predicting the winner in one day international cricket” in *Journal of*

*Mathematical Sciences & Mathematics Education.*

**C. D. Prakash, C. Patvardhan and C. V. Lakshmi,** *"Data Analytics based Deep Mayo Predictor for IPL- 9," International Journal of Computer Applications, vol. 152, no. 6, pp. 6-10, October 2016.*

**AHP-Neural Network Based Player Price Estimation in IPL.** *“*International Journal of Hybrid Information Technology, Vol.7, No.3 (2014), pp.15-24*”*

**Names of 5-9 referees**

1. Name, Affiliation, Country, e-mail

2. Name, Affiliation, Country, e-mail

3. Name, Affiliation, Country, e-mail

4. Name, Affiliation, Country, e-mail

5. Name, Affiliation, Country, e-mail

6. Name, Affiliation, Country, e-mail

7. Name, Affiliation, Country, e-mail

8. Name, Affiliation, Country, e-mail

9. Name, Affiliation, Country, e-mail

**URL link:**

**Notes for Authors** >> http://wjst.wu.ac.th/index.php/wjst/about/submissions

Online Submissions >> http://wjst.wu.ac.th/index.php/wjst/about/submissions

Current Issue >> http://wjst.wu.ac.th/index.php/wjst/issue/current

**WJST Publication Ethics and Malpractice** >> http://wjst.wu.ac.th/index.php/wjst/about

**Journal Title Abbreviations** >> <http://library.caltech.edu/reference/abbreviations>