Decoding Indian Premier League: Match Analysis

Mudit Dhawan
IIIT Delhi, India
mudit18159@iiitd.ac.in

Samik Prakash
IIIT Delhi, India
samik18090@iiitd.ac.in

Shivangi Dhiman
IIIT Delhi, India
shivangi18265@iiitd.ac.in

Abstract

In India, cricket is treated like a religion, one that brings culturally diverse people together. This unifying characteristic of the game motivated us to explore and analyze the Indian Premier League (IPL). In 2019, IPL had a viewership of 462 million, and in 2015, it contributed Rs. 11.5 billion (US $160 million) to the Indian GDP. This enormous impact of IPL on different facets of the Indian community and the economy makes it an interesting and lucrative problem to explore further. With the advancement in technology, successful teams have started incorporating machine learning techniques to process data from matches to help them make decisions, which elevate the team's performance. Having such data-driven approaches to better model a given match gives captivating insights into the players' performance to better understand the dynamics of the game and also help the team management make informed decisions. We have made the code public here [1].

1. Introduction

Given the recent advancement in technology, analytics through Machine Learning and Artificial Intelligence has become crucial in many areas, including sports. A considerable amount of data is now collected and used to predict scores and future trends, make managerial decisions and analyse the performance of the players in various sports, using Machine Learning techniques. We aim to use these advancements to analyse and explore the Indian Premier League (IPL). IPL is a Twenty20 cricket league in India contested by teams representing different Indian cities or states. Since its beginning in 2008, IPL has been followed regularly by people all over the country.

Our project has two aims. Firstly, we want to predict the outcome of matches, i.e. the winner of the match, based on previous years' data. Secondly, we want to predict the scores during certain intervals of the game. For example, the score in overs 1-15, 1-10 or 3-8 etc., eventually going all the way down to the score on a single ball. To accomplish this, algorithms such as Decision Trees, Linear Regression, Random Forest and Support Vector Machines have been used.

[1] https://github.com/MUDITDHAWAN/Decoding-IPL-Match-Analysis

2. Literature Review

In the paper [1], the performance of cricket players and toss related analysis in IPL, from 2008-2018 has been visualized. The paper highlights player performance and addresses the analysis done for awards like Man of the Match, maximum Centuries Scored by Batsmen and Top Batsmen. A multivariate regression-based model was formulated to calculate the points for each player based on their past performances [2]. Various classification-based machine learning algorithms were trained on the IPL dataset, which were then used to predict the outcome of each 2018 IPL match. The authors in [3] analysed the results of the IPL matches during the years 2008-2019 by applying data mining algorithms for existing data, and predicted new data for the year 2020, using Random Forest, Naive Bayes and Decision Tree.

3. Dataset

3.1. Data Collection

We collected data from Cricsheet's website [2] and scraped player points from IPL's official website [3] using python libraries like BeautifulSoup and requests. We then used pandas and csv libraries to store this data in csv format for each season from 2008-2020.

3.2. Data Description

Our dataset consists of all matches from 2008-2019, i.e. 756 matches. Table 1 mentions the attributes in the data files. Column 'Attribute Name' gives the list of available features available. Figure 1, Figure 2 and Figure 3 show the runs scored by top 10 Batsmen, wickets taken by top 10 Bowlers in all the seasons and the total number of matches played in each season of IPL respectively.

[2] https://cricsheet.org/
[3] https://www.iplt20.com/

Dataset	Features	Attribute Name	Samples
matches.csv: Data of matches played across all IPL seasons	18	id, city, date, season, team1, team2, toss_winner, toss_decision, result, dl_applied, winner, win_by_runs, win_by_wickets, player_of_match, venue, umpire1, umpire2, umpire3	756
deliveries.csv: Ball-by-ball data of all matches	21	match_id, inning, batting_team, bowling_team, over, ball, batsman, non_striker, bowler, is_super_over, wide_runs, bye_runs, legbye_runs, noball_runs, penalty_runs, batsman_runs, extra_runs, total_runs, player_dismissed, dismissal_kind, fielder	179078
player_stats.csv: Player points for each player in a particular season based on 7 categories	12	pos, name, team_no, team_name, pts, mat, wkts, dots, 4s, 6s, catches, stumpings	1988

Table 1: Dataset Description

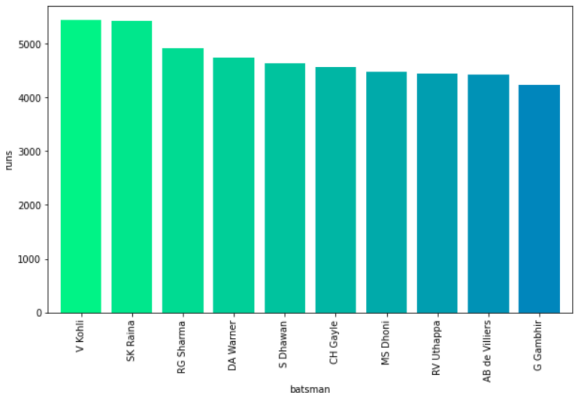

Figure 1: Runs scored by top 10 Batsmen

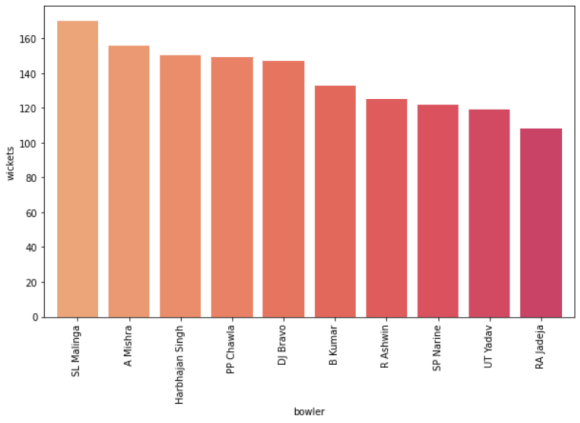

Figure 2: Wickets taken by top 10 Bowlers

3.3. Data Preprocessing

3.3.1 Data Cleaning

We have removed the features 'Umpire1', 'Umpire2' and 'dl_applied' from our dataset, because of their low correlation with the classification or regression output. Majority of the values for the feature 'Umpire3' had NaN value, hence they were dropped.

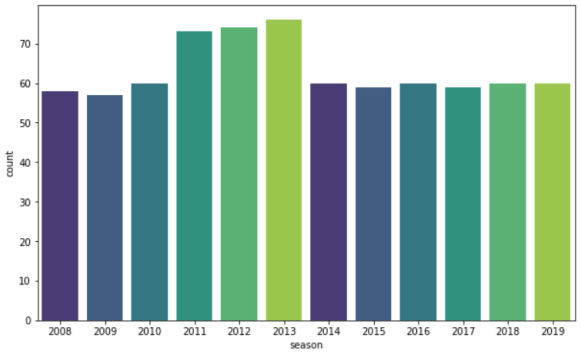

Figure 3: Number of Matches Played in Each Season

3.3.2 Categorical Data

Our dataset comprised mostly of string-based features (ex: 'venue, 'team_name', etc.). Converting these string-based features into numeric values, which can then be directly used as inputs for the machine learning models, posed a big challenge. To resolve this, we used three techniques: Label Encoding, One-hot Encoding, and Vector Embedding.

In Label Encoding, we converted string-based features into integer values. This method proves advantageous in terms of the ease to compute numerical representation. It does not increase the feature size, but this might introduce bias in the data, as it gives more weight to some of the values and less to others.

We then tried One-Hot Encoding to overcome the shortcomings of Label Encoding. In this method, we first computed the cardinality of the feature or the unique number of strings in the categorical features. After this, we represented each unique string with a zero-vector with a single '1' at the index associated with the specified string. The length of the zero-vector was equal to the cardinality. This solves the problem of bias introduction, as each feature is now weighted equally.

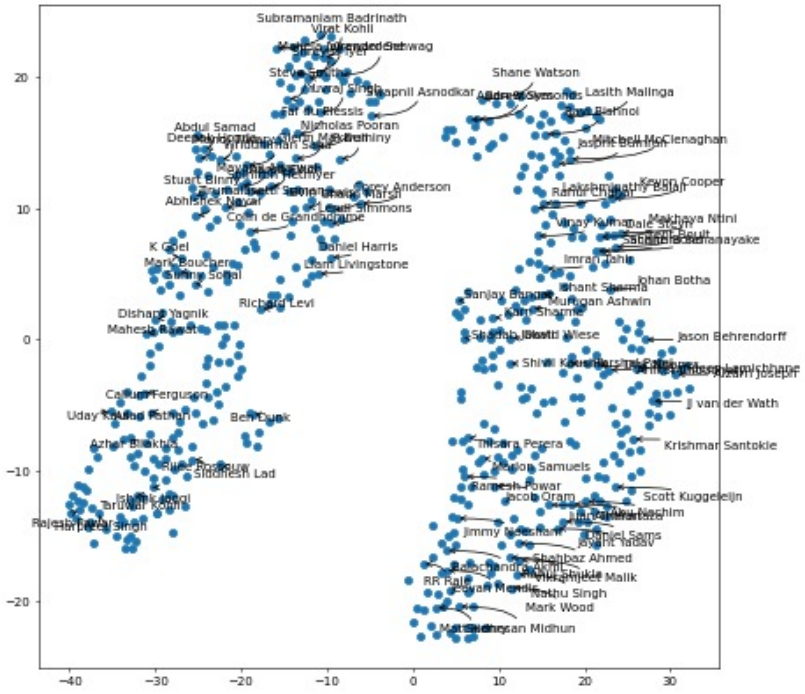

Figure 4: tSNE Visualisation of Vector Embedded player_names

One-hot encoding shows two significant disadvantages. First, the dimensionality of the embedded vector becomes very large when the cardinality of the feature is very high. Second, the transformed one-hot vector does not encode or capture the essence of the string, i.e., it does not place similar strings closer in the embedding space. In our case, we want the vectors representing similar players to be close to each other. For this, we have used the collected player statistics ('player_names') data to describe player names in terms of vectors of length 8, consisting of [pts, mat, wkts, dots, 4s, 6s, catches, stumpings], which can be directly fed into our models. In our dataset, we had 700+ unique player names, but using the vector embedding method, we transformed each player name into an 8-dimensional vector, which also encoded the performance of the players. To check how our embedding captures the underlying ratings and performance of each player, we visualized the embedding with the help of t-SNE, which mapped this 8-dimensional vector into 2-dimensional space for easier visualization. Figure 4 shows two main clusters formed in the 2-dimensional space, with the left cluster being dominated by batsmen and the right one by bowlers. This makes it evident that our player embedding technique is able to capture the inherent player quality (or type) in its numerical representation.

4. Methodology

We divided our problem statement into two experimental settings: Classification and Regression, based on the output variable. To incorporate the string features, we ran each model with label encoded and one-hot encoded data to compare these pre-processing techniques. We used GridSearchCV to tune our hyperparameters and find the best set that would optimize our regression and classification models.

4.1. Regression

We grouped our delivery-by-delivery dataset into groups of 30 and 12 deliveries to predict the total runs scored in the given balls. We calculated the sum of runs scored in individual balls and predicted this using the batting and bowling team, toss winner, toss decision, season and innings.

1. **Linear Regressor**: A simple model for regression which predicts the output variable by fitting a linear equation to the input data. The weights are updated after each iteration using gradient descent based on the error.

2. **Bagging Regressor**: Bagging Regressor is a variation of linear regression where the model is trained on a random subset of the original dataset and the results are aggregated.

For 12-ball intervals, the best hyperparameters were: {'n_estimators': 25} for label-encoding and one-hot encoding. In the 30-ball intervals, the best hyperparameters were: {'n_estimators': 25} for both label and one-hot encoding.

3. **SGD Regressor**: SGD stands for Stochastic Gradient Descent. In this model, the weights of the model are updated based on the penalty calculated on one sample at a time.
   The best hyperparameters in our model for 12-ball intervals was: {'alpha': 0.1}, with both label and one-hot encoding. For 30-ball intervals, {'alpha': 0.1} and {'alpha': 0.05} gave the best results for label encoding and one-hot encoding respectively.

4. **MLP Regressor**: MLP stands for Multi-Layer Perceptron. This model comprises of stacks of perceptrons with an activation layer in between two layers.
   For 12-ball intervals, the best hyperparameters were: {'activation': 'logistic', 'hidden_layer_sizes': (12,), 'max_iter': 2000} for label-encoding and {'activation': 'logistic', 'hidden_layer_sizes': (15,), 'max_iter': 1000} for one-hot encoding. In the 30-balls interval, the best hyperparameters were: {'activation': 'logistic', 'hidden_layer_sizes': (12,10), 'max_iter': 2000} and {'activation': 'relu', 'hidden_layer_sizes': (8,4,), 'max_iter': 2000} for label and one-hot encoding respectively.

5. **Random Forest Regressor**: It fits multiple decision trees on various sub-samples of the dataset and averages the result to improve accuracy and control overfitting.
   The best hyperparameters, in our model, for 12-ball intervals were {'max_depth': 5} and {'max_depth': 6} for label encoding and one-hot encoding respectively. In the 30-ball intervals, {'max_depth': 6} was the best hyperparameter for label and one-hot encoding.

4.2. Classification

We predicted the final outcome of an IPL match based on the competing teams, season, toss winner and the toss decision (Bat or Field). To achieve this, we experimented with four different classification models

1. **Logistic Regression**: This simple classification model is an extension of the linear regression model, where the output is passed through the sigmoid function to predict the probability of a given binary label.
   The best hyperparameters were {'C': 0.1, 'penalty': 'l2'}.

2. **SVM Classifier**: Support Vector Machine Classifier tries to find the best hyperplane that separates the dataset. In the non-linear case, SVM uses Kernel functions (rbf, Gaussian, etc) to project data onto higher-dimensional space where it is linearly separable.
   The best hyperparameters were {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}.

3. **Decision Tree**: A classification technique which uses criteria like Information Gain and Gini Index to split the data.
   The best hyper parameters for our model were {'criterion': 'gini', 'max_depth': 4, 'splitter': 'random'}.

4. **Random Forest**: An algorithm which fits a number of decision trees on various sub-samples of the original dataset. It further averages the outcome of these decision trees to improve the accuracy and reduce the variance of the model.
   The best hyperparameters for the Decision Tree model are {'criterion': 'entropy', 'max_depth': 4, 'n_estimators': 100}.

5. **Multi-layer Perceptron Classifier**: Internally, this model comprises of stacks of perceptrons with an activation layer in between two layers and optimizes the log-loss function.
   The hyperparameters {'activation': 'tanh', 'early_stopping': True, 'hidden_layer_sizes': (8, 8, 4, 2), 'learning_rate': 'constant', 'max_iter': 500} gave the best result.

5. Results and Analysis

We used K-Fold Cross Validation (k=5) to check the model's performance on unseen data. For our regression task, we have predicted the score in a 30-ball interval and a 12-ball interval. The performance of various implemented models is mentioned in Table 4. Random Forest was the best performing model for label encoded 30-ball interval and linear regression was best for one-hot encoded 30-ball interval. SGD Regressor performed best for the 12-ball task.

For our classification task, Multi-Layer Perceptron outperformed other models and had the highest testing accuracy of around 63%. The accuracy values for all the classification models is present in Table 3.

Using the vector embeddings created for the players, we tried to predict runs scored on a single ball and using MLP Regressor we were able to achieve the lowest mean absolute error on our dataset, as shown in Table 2.

We have observed that there is not a significant difference in label encoded dataset and one hot encoded dataset in the outcome.

Vector Embedding		
Model	**MAE Train**	**MAE Test**
Random Forest Regressor	**0.4857**	1.3120
Linear Regression	1.1875	1.1873
Multi-layer Regressor	1.1648	**1.1663**

Table 2: Mean Absolute Error on Ball-by-Ball dataset for the trained model (MAE: Mean Absolute Error)

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Logistic Regression	0.58	0.54	0.58	0.54
Decision Trees	0.58	**0.55**	0.58	**0.55**
Support Vector Machines	0.58	0.53	0.58	0.53
Random Forest	0.56	0.54	0.56	0.54
Multi-layer Perceptron	**0.63**	0.40	**0.63**	0.49

Table 3: Model Performance on the Classification Task

Number of Balls: 30					
Label Encoding					
MAE	Linear Regression	Random Forest	Bagging	SGD	Multi-layer Perceptron
Train	4.600	4.531	**4.305**	4.693	4.581
Test	4.641	**4.613**	4.731	4.693	4.616
One-Hot Encoding					
MAE	Linear Regression	Random Forest	Bagging	SGD	Multi-layer Perceptron
Train	4.609	4.530	**4.305**	4.611	4.619
Test	**4.587**	4.611	4.731	4.639	4.649
Number of Balls: 12					
Label Encoding					
MAE	Linear Regression	Random Forest	Bagging	SGD	Multi-layer Perceptron
Train	4.630	4.589	**4.325**	4.630	4.638
Test	4.578	4.586	4.732	**4.535**	4.591
One-Hot Encoding					
MAE	Linear Regression	Random Forest	Bagging	SGD	Multi-layer Perceptron
Train	4.604	4.560	**4.323**	4.635	4.633
Test	4.613	4.589	4.729	**4.579**	4.592

Table 4: Mean Absolute Error for 12 balls and 30 balls in the Regression Task (MAE: Mean Absolute Error, SGD: Stochastic Gradient Descent)

6. Conclusion

Data Preprocessing is an integral part of any Machine Learning pipeline. It helps in detecting and eliminating outliers and resolve any inconsistencies. String-based features provide information about the dataset and are important in the modelling process. However, finding a good encoding technique to convert these string-based features into a machine-readable numeric form is imperative because it enhances the quality of the input data and can provide more desirable results. Understanding the advantages and disadvantages of encoding techniques such as Label Encoding, One-hot Encoding and Vector Embedding helps in finding the best technique and using it for a given problem statement. Vector Embeddings are able to capture the intrinsic meaning of categorical data and improve the process of extracting features from such string-based features. We also understood that hyperparameter tuning is important to achieve the best results.

Future Work: We were able to create an embedding for our player names, which was able to capture the players' performance. This helped to encode inherent logical information, which we can use in our models in the future. Our future efforts would be in the direction of extending these vector embeddings to the entire team. This would help us encode the true value of the team based on

the entire squad and help enhance our findings.

Each Member's Contribution
Mudit Dhawan: Exploratory Data Analysis, Vector Embedding, Stochastic Gradient Descent Regression, Random Forest, Analysis & Results.

Samik Prakash: Exploratory Data Analysis, Linear Regression, Multi-layer Perceptron, Decision Tree, Data Preprocessing, Analysis & Results.

Shivangi Dhiman: Exploratory Data Analysis, Logistic Regression, Support Vector Machines, Bagging Regressor, Analysis & Results.

References

[1] Vidit Kanungo and Tulasi Bomatpalli. Data visualization and toss related analysis of ipl teams and batsmen performances. *International Journal of Electrical and Computer Engineering (IJECE)*, 9:4423, 10 2019.

[2] Rabindra Lamsal and Ayesha Choudhary. Predicting outcome of indian premier league (ipl) matches using machine learning, 09 2018.

[3] Sachi Priyanka. Prediction of indian premier league-ipl 2020 using data mining algorithms. *International Journal for Research in Applied Science and Engineering Technology*, 8:790–795, 02 2020.