IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 11, November 2026

DOI: 10.17148/IJIREEICE.2026.141107

IPL MATCH WIN PREDICTION SYSTEM

Sai Sunder G[1], Parvesh S[2], Saran T[3], Sharan R T[4], Neelam Sanjeev Kumar[5]

Student, Department of CSE (Emerging Technologies), SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[1,2,3,4]

Assistant Professor, Department of CSE (Emerging Technologies), SRM Institute of Science and Technology, Vadapalani Campus, Chennai, India[5]

Abstract: This paper presents a machine learning-based IPL Match Win Prediction System that estimates the probability of victory for each team after every ball of a Twenty20 match. Historical ball-by-ball data from the 2008–2024 IPL seasons is converted into labelled training examples using cricket-specific features: the current run rate (CRR), required run rate (RRR), wickets in hand, innings phase, venue and toss details. Logistic Regression, Random Forest and XGBoost were compared using a chronological train–test split that prevents data leakage from future seasons. XGBoost achieved the best performance with 84% accuracy and an AUC-ROC of 0.90 on the held-out season, outperforming Random Forest (81%) and Logistic Regression (76%). Probability calibration keeps the displayed win percentages realistic in close finishes. The model is served through a FastAPI endpoint, logged to a PostgreSQL database with an automatic SQLite fallback, and presented in an interactive Streamlit dashboard with feature-importance explanations.

Keywords: IPL, win probability, cricket analytics, XGBoost, machine learning, probability calibration, FastAPI, Streamlit.

I. INTRODUCTION

Cricket is among the most widely followed sports in the world, and the Indian Premier League (IPL) is its most commercially successful Twenty20 (T20) competition. During a match, the question that occupies fans, commentators and analysts is simple: who is going to win? The answer is not fixed at the toss; it changes ball by ball as wickets fall, boundaries are struck and the required scoring rate shifts. Traditional analysis relies on static pre-match statistics such as head-to-head records and average ground scores. These are useful for setting expectations but cannot react to events on the field, where a single over of twenty-five runs can overturn a prediction.

Machine learning is well suited to this problem. Ball-by-ball data is available for every season since 2008, and each delivery can be turned into a labelled example whose label is the eventual match result. In this paper we propose a **ball-by-ball win-probability model** that merges pre-match context with the live match state, evaluates three classifiers fairly, calibrates the output probabilities, and deploys the result as a complete, usable system consisting of a prediction API, a database and a dashboard. The principal contributions are: (i) cricket-specific feature engineering shared between training and serving; (ii) a leakage-free chronological evaluation protocol; (iii) explicit probability calibration; and (iv) an open, end-to-end implementation built with freely available tools.

II. RELATED WORK

Duckworth and Lewis [1] formalised overs remaining and wickets in hand as the two resources that determine scoring potential, although their method targets interrupted matches rather than live win probability. Bailey and Clarke [2] showed that combining pre-match information with the evolving match situation improves in-play outcome prediction in One Day Internationals. Simulation approaches by Swartz et al. [3] and Davis et al. [4] estimate win probabilities through Monte Carlo ball-by-ball models, but require detailed player outcome models and are computationally heavier. On the machine learning side, Logistic Regression [5] offers fast, direct probabilities but assumes linear effects; Random Forest [6] captures non-linearity yet tends to compress probabilities; and gradient-boosted trees such as XGBoost [7] perform strongly on tabular data with built-in regularisation. Platt scaling [8] and the study of Niculescu-Mizil and Caruana [9] highlight why calibration is needed when probabilities, not just labels, are shown to users.

Four gaps motivate this work: the separation between pre-match and in-play analysis; random train–test splits that let deliveries from the same match leak between sets and inflate accuracy; reporting accuracy while ignoring probability quality; and the absence of open, end-to-end, reproducible systems, since many published models stop at an offline notebook and many commercial engines are closed.

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 11, November 2026

DOI: 10.17148/IJIREEICE.2026.141107

III. DATASET DESCRIPTION AND PREPROCESSING

The data is publicly available IPL ball-by-ball data (Cricsheet [12]) covering 2008–2024 and more than one thousand matches across all franchises. A ball-by-ball table (one row per delivery: match, innings, over, ball, teams, runs, extras, wicket) is joined on the match identifier to a match-level table (season, venue, toss winner, toss decision, winner), which supplies the training label. Preprocessing comprised: (1) removal of invalid records and of matches with no result or non-standard outcomes; (2) standardising franchise and venue names that changed over time; (3) merging the two tables; (4) computing the chase target as first-innings total plus one; (5) counting legal deliveries only, since wides and no-balls add runs but not balls; (6) encoding categorical fields (one-hot for the linear baseline, category codes or one-hot for tree models); and (7) labelling each delivery of the chase as 1 if the batting team went on to win, otherwise 0.

After preprocessing, each row represents one delivery of a second innings. Because every match yields many rows with the same label and the chasing side wins close to half of all IPL matches, the label is approximately balanced and accuracy is a meaningful metric. Any statistic learned from data, such as a venue encoding, is computed from training seasons only, and no feature may depend on the final score or on events after the current ball.

IV. FEATURE ENGINEERING

The match state is summarised by a small set of quantities capturing the pressure on the batting side. CRR measures scoring speed so far, RRR the pace now required, wickets in hand the remaining batting depth, and the innings phase (powerplay: overs 1–6; middle: 7–15; death: 16–20) the stage of play. Venue, toss winner/decision, and the batting and bowling teams are added as static context.

runs_left = target - current_score balls_left = 120 - legal_balls_bowled wickets_in_hand = 10 - wickets_lost
CRR = current_score * 6 / legal_balls_bowled RRR = runs_left * 6 / balls_left
All definitions live in one reusable function used both offline for training and online by the API, which removes training–serving skew.

V. SYSTEM ARCHITECTURE

The system is a layered pipeline (Fig. 1). The data and preprocessing layers clean and merge the records; the feature layer computes CRR, RRR, wickets in hand, phase, venue and toss encodings; the modelling layer holds the XGBoost classifier with Logistic Regression and Random Forest baselines; the service layer is a FastAPI application (served by Uvicorn) exposing */predict*, */predictions*, */matches* and */health* endpoints, with Pydantic request validation; and each prediction is logged through SQLAlchemy to PostgreSQL, falling back automatically to SQLite when PostgreSQL is unavailable. A Streamlit dashboard consumes the API and shows both teams' win probabilities, CRR and RRR, the innings phase, a ball-by-ball probability curve and the feature importance. Because the model sits behind an API, the dashboard is only one of several possible clients.

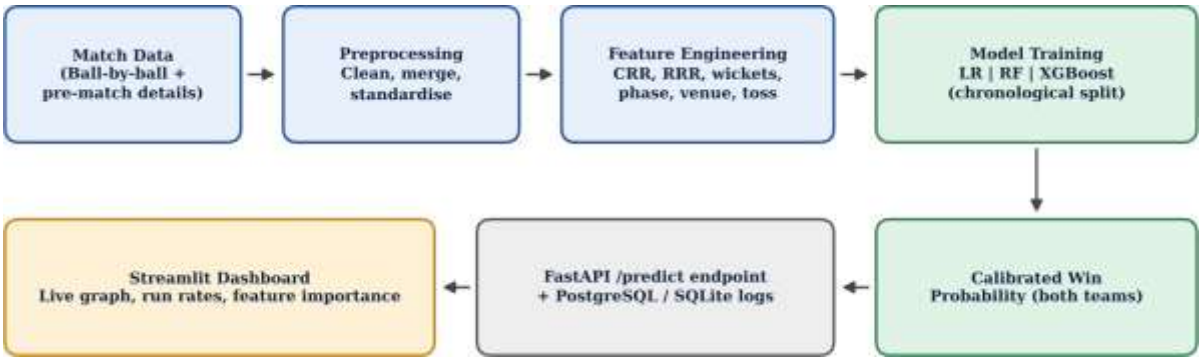


Fig. 1. System Architecture of the IPL Match Win Prediction System

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 11, November 2026

DOI: 10.17148/IJIREEICE.2026.141107

VI. METHODOLOGY

A. Models. Three classifiers were chosen to measure the benefit of model complexity against a simple baseline. Logistic Regression models the log-odds of a win as a linear combination of features with a small L2 penalty. Random Forest averages many trees grown on bootstrap samples with random feature subsets. XGBoost, the primary model, is a regularised gradient-boosted tree ensemble in which the win probability for feature vector x is

$$p(x) = 1 / (1 + \exp(-F(x))), \quad F(x) = \sum_{k} f_k(x)$$

where each f_k is a regression tree fitted to the gradient of the logistic loss, with a penalty on the number of leaves and leaf weights. Number of trees, maximum depth, learning rate and row/column subsampling were tuned on a validation subset with early stopping; all models used identical features and the same split.

B. Training and validation. The data was split chronologically: earlier seasons form the training set and the most recent season the test set, which was used only once, at the end. A validation subset carved out of the training seasons (keeping each match whole, since splitting a match across folds would leak information) was used for tuning and for fitting the calibration. Accuracy (predicted winner = team with probability above 0.5), AUC-ROC and log-loss were reported.

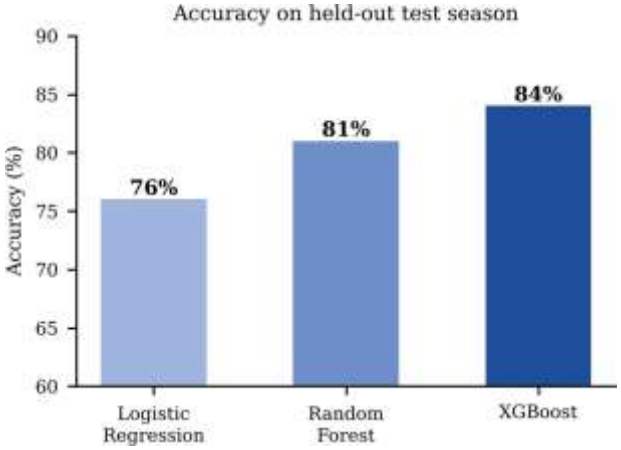
C. Calibration. Boosted trees can output probabilities more extreme than observed frequencies. A monotonic mapping from raw scores to observed win frequencies is fitted on the validation set so that, for example, situations reported at 70% are won about 70% of the time. This is most important in close finishes, where overconfidence is most misleading.

VII. RESULTS AND ANALYSIS

All results are computed on the held-out most recent season. Table I and Fig. 2 show that XGBoost achieved 84% accuracy, an AUC-ROC of 0.90 and the lowest log-loss (0.38), ahead of Random Forest (81%) and Logistic Regression (76%). The five-point gain from the linear baseline to Random Forest shows that non-linear modelling matters, and the further three points from boosting indicate that sequential error correction and regularisation help on the hard, close situations in which the required rate and the wickets in hand pull in opposite directions. An AUC of 0.90 means that a randomly chosen situation from a match the batting side won receives a higher win probability than one from a match it lost about 90% of the time.

TABLE I. MODEL COMPARISON ON THE HELD-OUT TEST SEASON

Model	Accuracy	AUC-ROC	Log-Loss
Logistic Regression	76.0%	0.79	0.52
Random Forest	81.0%	0.86	0.44
XGBoost (selected)	**84.0%**	**0.90**	**0.38**



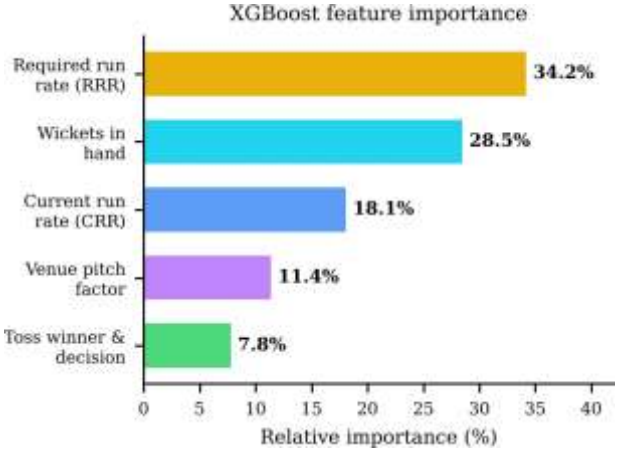


Fig. 2. Model accuracy (left) and Fig. 3. XGBoost feature importance (right)

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 11, November 2026

DOI: 10.17148/IJIREEICE.2026.141107

Feature importance. The required run rate (34.2%) and wickets in hand (28.5%) together account for 62.7% of the model's decision weight, followed by CRR (18.1%), venue pitch factor (11.4%) and toss winner and decision (7.8%). The static features (venue and toss) total only 19.2%; they matter mainly in the early overs, before dynamic information accumulates. This agrees with the Duckworth–Lewis intuition that overs and wickets are the key resources of an innings and supports the central hypothesis that live state is more informative than pre-match statistics.

Example output. Table II shows a dashboard scenario: Mumbai Indians chasing Chennai Super Kings at Wankhede Stadium need 53 runs from 34 balls with 7 wickets in hand (CRR 9.21, RRR 9.35, middle overs). The model assigns 62.2% to Mumbai Indians and 37.8% to Chennai Super Kings. This is plausible: the rates are nearly level and seven wickets provide batting resources, yet 53 off 34 remains demanding, so about 60% rather than 90% reflects the benefit of calibration. With only three wickets in hand at the same rate, the probability would be expected to fall considerably.

TABLE II. EXAMPLE DASHBOARD OUTPUT

Item	Value
Batting (chasing) / Bowling (defending)	Mumbai Indians / Chennai Super Kings
Stadium / Toss	Wankhede Stadium, Mumbai / Bowling side won the toss
Equation	Needs 53 runs off 34 balls with 7 wickets remaining
CRR / RRR / Phase	9.21 / 9.35 / Middle overs (7–15)
Win probability (MI / CSK)	**62.2% / 37.8%**

Functional testing. End-to-end tests (dashboard → API → database) checked that the probabilities behaved as expected on cricketing grounds in the scenarios of Table III, and that logs were written under both PostgreSQL and SQLite configurations.

TABLE III. INTEGRATION TEST SCENARIOS

Scenario	Expected behaviour
Start of a chase	Probability close to even, adjusted by venue and toss
Fast start in powerplay	Probability of the batting team rises
Middle-overs collapse	Probability of the batting team falls sharply
Tight finish	Probability moves strongly with each boundary or wicket
Almost certain win	Very high but not exactly one after calibration
Database unavailable	Falls back to SQLite and continues logging
Invalid input (overs > 20)	Request rejected with a clear validation error

VIII. ADVANTAGES AND LIMITATIONS

Advantages. The system reacts to every wicket and boundary, unlike static pre-match statistics; tree ensembles capture interactions such as a high required rate combined with few wickets in hand without manual feature crossing; scoring takes a fraction of a millisecond, so the delay in a live graph is dominated by the data feed and interface rather than the model; the dashboard's feature-importance panel explains *why* a probability changed; and the API-first design lets a broadcast graphic or mobile app reuse the same predictions.

Limitations. The 84% accuracy is averaged over all deliveries of the chase; late-chase situations are far easier than early ones, so early-over accuracy is lower. Deliveries within a match are correlated, so the effective number of independent test examples is closer to the number of matches, and results are from a single held-out season; they indicate performance rather than guarantee it. The model works at team level and ignores player form, batter–bowler matchups, weather, dew, injuries and team composition; it suits second-innings prediction because the target is unknown in the first innings; and it was tested on manually entered or replayed states rather than a commercial live feed. A fuller reliability analysis over many matches would give stronger evidence of calibration quality.

IX. CONCLUSION AND FUTURE WORK

This work built and evaluated a complete IPL Match Win Prediction System that merges pre-match context with the live match state to update win probabilities after every ball. Under a leakage-free chronological protocol, XGBoost reached

DOI: 10.17148/IJIREEICE.2026.141107

84% accuracy and an AUC-ROC of 0.90, eight points above the Logistic Regression baseline, with the required run rate and wickets in hand dominating the predictions. Calibration keeps probabilities realistic, and FastAPI, SQLAlchemy (PostgreSQL with SQLite fallback) and Streamlit show that an interpretable prediction system can be built from open tools on ordinary hardware. Future work includes player-level statistics and head-to-head records, a real-time scoring API, a separate first-innings score model, weather and dew features, sequence models such as LSTM networks, SHAP-based explanations, and automated monitoring with periodic retraining.

REFERENCES

[1]. F. C. Duckworth and A. J. Lewis, "A fair method for resetting the target in interrupted one-day cricket matches," *J. Oper. Res. Soc.*, vol.49, no. 3, pp. 220–227, 1998.

[2]. M. Bailey and S. R. Clarke, "Predicting the match outcome in one day international cricket matches, while the game is in progress," *J. Sports Sci. Med.*, vol. 5, no. 4, pp. 480–487, 2006.

[3]. T. B. Swartz, P. S. Gill, and S. Muthukumarana, "Modelling and simulation for one-day cricket," *Can. J. Stat.*, vol. 37, no. 2, pp. 143–160, 2009.

[4]. J. Davis, H. Perera, and T. B. Swartz, "A simulator for Twenty20 cricket," *Aust. N. Z. J. Stat.*, vol. 57, no. 1, pp. 55–71, 2015. [5]. D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*, 3rd ed. Hoboken, NJ, USA: Wiley, 2013. [6]. L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[7]. T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016, pp. 785–794.

[8]. J. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *Advances in Large Margin Classifiers*, MIT Press, 1999, pp. 61–74.

[9]. A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proc. 22nd Int. Conf. Machine Learning (ICML)*, 2005, pp. 625–632.

[10]. F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[11]. T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. New York, NY, USA: Springer,2009.

[12]. Cricsheet, "Ball-by-ball data for cricket matches," cricsheet.org (public ball-by-ball cricket data source).

[13] N. S. Kumar, S. V. S, A. M. Enoch, K. T and W. C. R, "Hybrid Machine Learning Framework for Secure Information Verification in Social Media," *2025 IEEE 17th International Conference on Computational Intelligence and Communication Networks (CICN)*, Goa, India, 2025, pp. 907-913, doi: 10.1109/CICN67655.2025.11368151.