Probabilistic Model Checking in Sports Analytics

Bharadwaj Mudumba and Mahima Shah

[1]Computer Science, Penn State Harrisburg, Penn State University, Street, City, 100190, State, Country.

Abstract

The cricket score prediction models proposed in the recent past focused on few aspects either with the past statistics or with the live feed of the match. In the cricket score prediction using random forest model, we broadened the horizon by predicting the score of a match by covering the maximum aspects without over fitting the model. This project aims to formally verify the score prediction system using model checking. As the score is probabilistic in nature we used probabilistic model checker and used UPPAAL to verify the system.

1 Introduction

Cricket is the second most popular sport played majorly in subcontinent, England, Australia, South Africa, and West Indies. It is played in three formats viz, Test, ODI and T20. IPL is a franchise-based T20 tournament based in India in which 8 teams compete with each other in the round-robin stage followed by playoffs. In the recent times, researchers created models to predict the cricket scores, out of which a popular method Duckworth-Lewis is formulated and adopted by ICC in the 1990's which is used till date. However, it is often criticized for the shorter formats like T20. Few other models have been developed in recent years, which either predicts the score based on the recent form and the past encounters between the teams, which shouldn't be the ideal model as a team can win irrespective of previous records, Few models requires the game to be in progress and uses the current game scores and statistics to predict the score without the knowledge of the playing teams. We decided to

create a model to address the above limitations by considering both current match statistics and the team data.

Our objective is to prepare a model that accurately predicts IPL match scores using machine learning algorithm. Further, verify the model using probabilistic model checker. The model checker will verify the completeness of the system. It will also verify the safety and liveness properties of the system.

In contrast, to support vector machines (SVMs) and neural networks (NNs), random forests (RFs) are capable of producing interpretable models due to the formal semantics associated with there underlying tree structures. Moreover, when analysing structured data, RFs often outperform other approaches. Given the above observations, we considered that a decision tree based machine learning approach fits our needs. The formal semantics of decision trees offers an ideal support for the application of formal methods. [1]

2 Literature Review

There are several models proposed that uses various deep learning and machine learning strategies for the prediction of the cricket score. The paper "Cricket score prediction by Machine learning" predicted the scores of the team from current situation and win percentage of both the teams for the ODI format that is a 50 over match. But it also mentions about the difficulty in predicting the scores for middle overs.[2]

The paper "Cricket Analysis and prediction of Projected score and winner using Machine learning" states the prediction of the score and winner using Linear regression. But this model considers the data from past five overs for the prediction of the score. Where as our model considers the past statistics as well as the live feed from the current match.[3]

The paper "Predicting outcomes of cricket matches using classification learners" shows application of KNN, Naive Bayes and Random Forest classifiers for reaching the goal of project. Our project along with machine learning implements the formal method using probability model checker to get more accurate results.[4]

The paper "Combining Deep Learning and Probabilistic Model Checking in Sports Analytics" shows how the Formal methods can be integrated with deep learning techniques in order to get accurate results for the predictions.[5]

The paper "Formal Methods in Requirement Phase of SDLC" assist in how and where to apply Formal methods.[6]

3 Methodology

3.1 System to be verified - IPL Score Prediction

To verify the application , it is important to clearly define the requirements of the application. As part of this we have framed a set of functional requirements of the application.

1. Overs should not be more than 20.
2. Balls in the over cannot be more that 6.
3. The number of wickets should not be more than 11.
4. Batting team and the bowling team cannot be the same.
5. The random forest algorithm must be passed with the playing team, opposition, current over, runs scored in last 5 overs, wickets lost in last 5, current score.
6. Current score should not be less than the runs scored in the last 5 overs.
7. Runs scored in last 5 overs should always be less than or equal to the current score.
8. No output should be generated until all the inputs are provided.

The IPL Score prediction is based on Random Forest Regressor. Random Forest is a supervised learning algorithm that uses ensemble learning which in turn uses large number of decision trees to train the data. Ensemble learning is a method that uses multiple decision trees to make more accurate predictions than a single model. Random forest overcomes the problem of data overfitting and high variance faced while predicting using decision trees. Decision trees are highly sensitive to training datasets. Considering these factors we have used random forest regressor as our training model. Figure 1 shows the user interface of the IPL score prediction system.

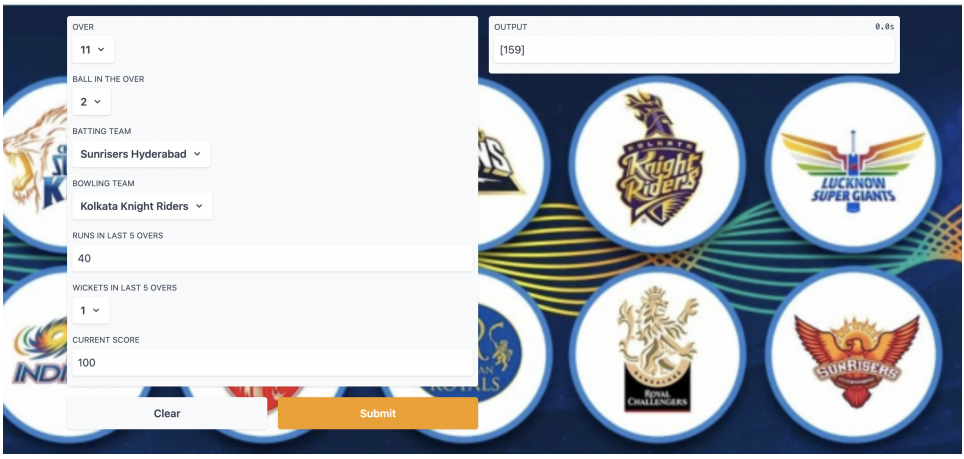

Fig. 1: User Interface of the System

3.2 Formal Verification - Model Checking

Model checking is a method to verify the completeness of the system. Using model checking, one can explore all possible future states from the current state the application can be. As shown in figure2, the model checker takes the model

and the properties of the system as an input. The output of the model checker is true if the property is satisfied otherwise, it generates a counter example which enables us to redesign the model making adjustments by tracing the counter examples.

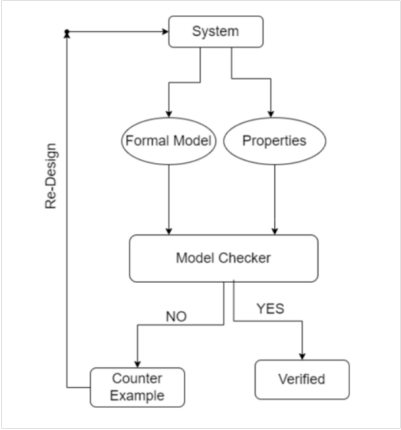

Fig. 2: Model Checking Structure

Model checker, also known as Property Checker, automatically checks whether the system model meets the desired specification properties. The specification implies to the property of liveliness and the safety property of the system. It refers to the algorithm for checking the state space of the transition system to determine the correctness of the system.

To verify a system using such a verification technique, the system needs to be described using a modelling language, the properties are formulated with the help of specification language and an algorithm is required to verify the system.

The system to be verified is represented in state transition diagram or graph and the properties are represented using propositional temporal logic. And then an efficient algorithm is used to determine whether the state transition graph satisfies the properties that are described in temporal logic formulas.

3.2.1 Choosing the right Model Checking tool

There are various model checking tools available today and it is important for us to choose a model checking tool that fit in our requirements. To do this we have explored the following tools,

1. UPPAAL Model Checker
2. PAT - Process Analysis Tool. [7]
3. SPIN Model Checker [8]

4. PRISM Model Checker

Among these, we have chosen UPPAAL considering the factors
1. Probabilistic model checking
2. Better representation of Model
3. Complete IDE for model checking
4. Ease of use

Uppaal is an integrated tool environment for modeling, simulation and verification of real-time systems, developed jointly by Basic Research in Computer Science at Aalborg University in Denmark and the Department of Information Technology at Uppsala University in Sweden. It is appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure, communicating through channels or shared variables [WPD94, LPW97b]. [9].

3.3 Modeling

In UPPAAL, we create templates to model a system. UPPAAL requires the system to be represented in the form of a network of time automata, consisting of locations representing the state and edge connecting the locations representing the transitions. Better practise is, put each process in a template and channels provides the synchronization among processes.

We have represented IPL Score predictor application in 4 templates,

User Interface Template

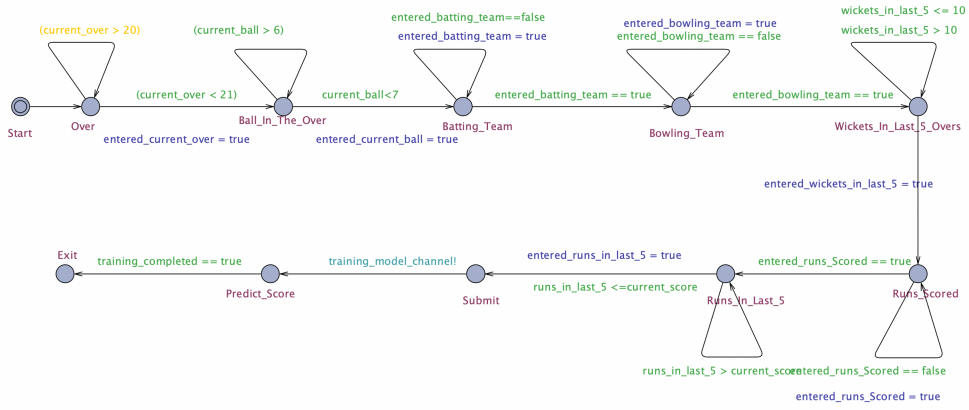

Fig. 3: User Interface Template in UPPAAL

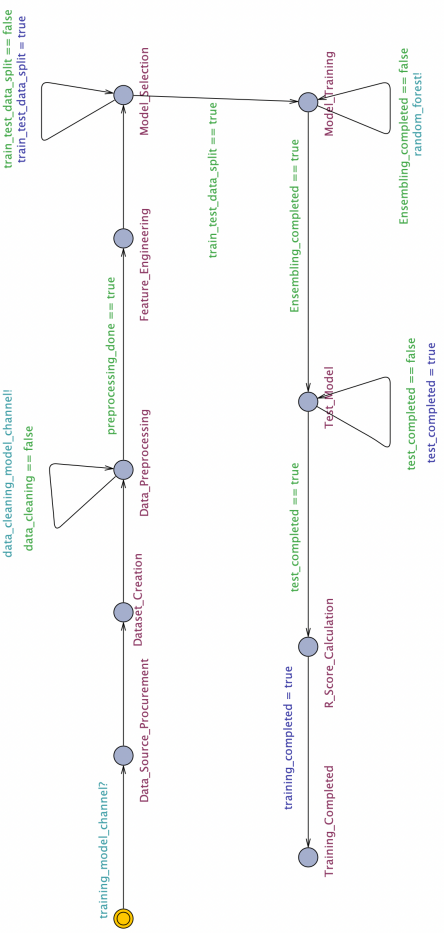

Fig. 4: Model Training Template in UPPAAL

System starts in the User Interface channel. It has the initial state and the final state. Once the system reaches "Submit" state, it synchronises with training channel. Once the training is completed the system returns to User interface channel and displays the predicted score and exits the system.

Training Template This template is reachable from User Interface channel. User interface channel has a submit location which is synchronised with the training model. Training model has incoming transition from training channel and has outgoing transitions to "Data cleaning" channel for data preprocessing, "Random forest" channel to perform "ensembling" and to "User Interface" channel once the training is completed.

Data cleaning Template This template is reachable from training model channel when it is in data preprocessing state and requires data cleaning. If data cleaning is not required training model bypasses this channel.In data

cleaning channel, it performs data preprocessing by removing null values, and dropping unwanted columns. And once the data is cleaned it is sent back to training channel.

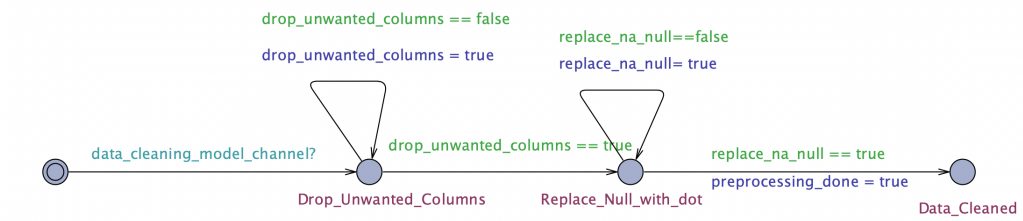

Fig. 5: Data Cleaning Template in UPPAAL

Random Forest Template This template is reachable from training channel when it is in training state. Once forest of trees are generated and ensembling is applied, the system is transitioned back to training channel.

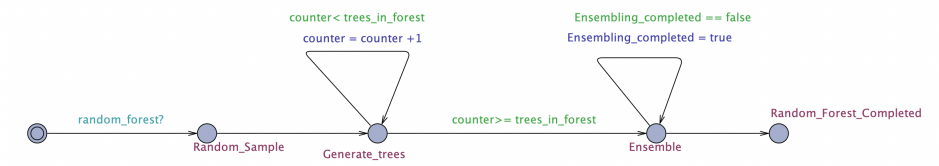

Fig. 6: Random Forest Template in UPPAAL

3.4 Simulator

The simulator is a validation tool that enables examination of the possible dynamic executions of a system during early design (or modeling) stages. The simulator enables to visualize executions through simulation trace and message sequence chart. Using simulator we can examine the behaviour of each states and transitions

3.5 Verifier

Verifier is a component to define the formal specifications. The specifications considered as properties are verified against the model and checks whether the property is satisfied or not.In UPPAAL, The properties are required to be defined in Computational Tree Logic.

3.5.1 Computational tree logic

The functional requirements must be converted into formal queries using computational tree logic and passed to test the model. If the property is satisfies

then the path to that state is well reachable. If not counter examples are generated. Computation tree logic is applied to formal verification techniques to verify the correctness and liveness properties of hardware and software systems.

3.5.2 Model checking properties

We aim to verify the completeness of the system by applying model checking. We have verified the Safety and Liveness properties using the model checker.

Safety Something bad should not happen. i.e, A state should not be reached if it is not intended to.

Liveness Something good should eventually happen. i.e, A state is intended to be reached, it should be reached eventually.

4 Requirement Specification

We have converted the functional requirements of our system into the following formal properties

S.No	Property Type	Property Description
1	Safety	A□$not((preprocessing\_done == false)\&(training.Feature\_Engineering == true)$
2	Safety	A□$not((training.train\_test\_data\_split == false)\&$ (training.Model_Training))
3	Safety	A□$not((ui.entered\_runs\_in\_last\_5 == false)\&(ui.Submit))$
4	Liveness	A<> ui.Submit
5	Liveness	A<> training.Model_Selection
6	Liveness	A<> training.R_Score_Calculation
7	Liveness	A<> ui.Predict_Score
8	Safety	A□$(ui.Submitimply(ui.entered\_runs\_in\_last\_5 == true))$
9	Safety	A□$((rf.counter > rf.trees\_in\_forest)\&(rf.random\_forest\_completed == false)$| (rf.random_forest_completed imply (rf.counter $\geq rf.trees\_in\_forest$))
10	Safety	A□$(cleaning.Data\_Cleanedimply(cleaning.drop\_unwanted\_columns\&$ cleaning.replace_na_null))
11	Liveness	A□$(training.Training\_Completedimply(Ensembling\_completed == true))$
12	Liveness	A□$(training.Feature\_Engineeringimply(preprocessing\_done == true))$
13	Liveness	A□$(training.Model\_Trainingimply(training.train\_test\_data\_split==true)$
14	Liveness	A□$(training.Test\_Modelimply(Ensembling\_completed == true))$
15	Liveness	A□$(rf.counter \leq rf.trees\_in\_forest)$

Table 1: Reachability Properties

5 Experiments and Test Results

.

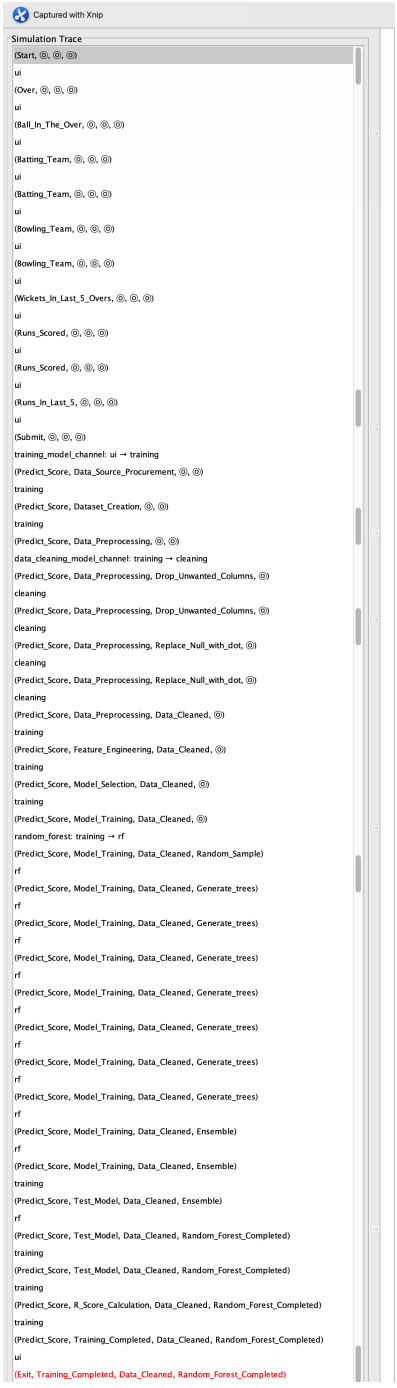

Fig. 7: Random Forest Template in UPPAAL

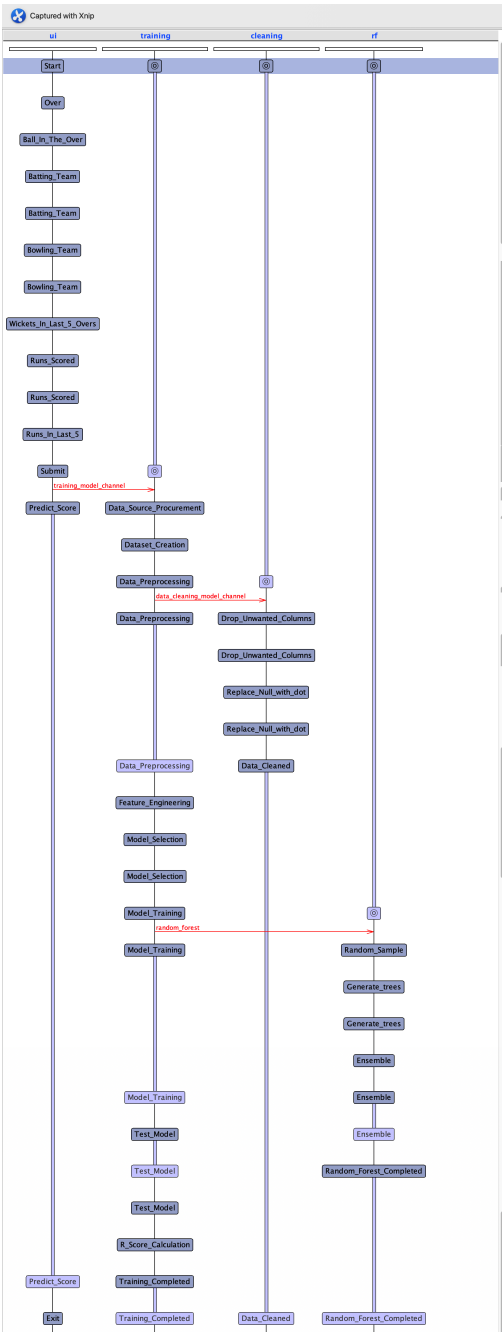

Fig. 8: Sequence Chart

We have verified the above specified properties and all of the properties are satisfied.

The diagnostic trace shows the trace generated during the execution, and the transitions are as expected

6 Discussion

The following are some of the things we learned as a result of this project:

a. Learned how to use and represent the system to be verified in UPPAAL.

b. How to formally specify the requirements of the system as properties to the model checker.

c. The representation of the properties using computational tree logic.

7 Conclusion and Future Enhancements

We have successfully implemented the model checking on our application and verified the properties of safety and liveness.

This model is open for modifications with the changes to the application. Such as it allows number of balls in an over to be more than six, which is a possibility in case of wide ball or no ball in the over.

Little modification in the overs state enables us to predict the score of ODIs and various other formats.

Score prediction is also utilized in a wide range of sports. As a result, a similar model for verification using formal methods can be constructed for any sport.

References

[1] Bride H., Dong J., Dong J.S., Hóu Z. (2018) Towards Dependable and Explainable Machine Learning Using Automated Reasoning. In: Sun J., Sun M. (eds) Formal Methods and Software Engineering. ICFEM 2018. Lecture Notes in Computer Science, vol 11232. Springer, Cham. https://doi.org/10.1007/978-3-030-02450-5_25

[2] @inproceedings Etal2021CricketSP, title=Cricket Score Prediction Using Machine Learning, author=Prof. R. R. Kamble, Et. al.,year=2021

[3] Apurva Lawate, Et. al.(2021) Cricket Analysis and Prediction of projected Score and Winner using Machine Learning. IJARCCE, Vol. 10, Issue 2

[4] Purkayastha, Vivek (2017) Predicting outcome of cricket matches using classification learners. Masters thesis, Dublin, National College of Ireland

[5] Jiang K. (2018) Combining Deep Learning and Probabilistic Model Checking in Sports Analytics. In: Sun J., Sun M. (eds) Formal Methods and Software Engineering. ICFEM 2018. Lecture Notes in Computer Science, vol 11232. Springer, Cham. https://doi.org/10.1007/978-3-030-02450-5_32

[6] @MISCPandey_formalmethods, author = S. K. Pandey and Mona Batra, title = Formal Methods in Requirements Phase of SDLC, year =

[7]PAT: Process Analysis Toolkit - An Enhanced Simulator, Model Checker and Refinement Checker for Concurrent and Real-time Systems. (n.d.). Pat.comp.nus.edu.sg. https://pat.comp.nus.edu.sg/

[8]Spin - Formal Verification. (n.d.). Spinroot.com. Retrieved May 1, 2022, from https://spinroot.com/spin/whatispin.html

[9]Features UPPAAL. (n.d.). Uppaal.org. Retrieved May 1, 2022, from https://uppaal.org/features/