Exploring Methods to Forecast T20 Cricket Game Scores

Anish Deshpande

2025-06-06

1 Abstract

Accurate real-time forecasting of final scores in T20 cricket can provide critical insights for team strategy, broadcasters, and bettors. In this project, we explore multiple modeling approaches to predict the final score of an inning given its partial progression. Using a curated dataset of over 22,000 T20 innings from Cricsheet, we construct a tensor representation of ball-by-ball cumulative runs and wickets. We evaluate the performance of several forecasting methods, including linear regression, ARIMA time series models, multi-dimensional robust synthetic control (mRSC), K-nearest neighbor (KNN) trajectory simulation, and multiple activation modeling via the `mactivate` package in R. We find that while linear regression offers strong early benchmarks, more sophisticated donor-based methods provide greater flexibility and robustness, especially post-intervention. Our results highlight the importance of incorporating historical game patterns and structural dynamics in modeling cricket scores over time. We find that the mRSC and the mactivation algorithms provide the most accurate and consistent results. We end by discussing future directions and use cases of these forecasting algorithms.

2 Introduction

T20 cricket, the shortest international format of the game, has seen an explosion in popularity over the last two decades, thanks to its fast pace, strategic complexity, and entertainment value. Unlike Test or One-Day matches, T20 games are limited to a single inning per team, with each inning restricted to 20 overs (i.e., 120 legal deliveries). The match begins with a coin toss, where the winning team elects to bat or bowl first. Team A bats first, aiming to score as many runs as possible before either losing all 10 wickets or completing 20 overs. Team B then attempts to chase this target, and the match concludes when the target is surpassed, the full quota of overs is bowled, or all 10 wickets fall.

What makes T20 cricket particularly dynamic is the high scoring rate and rapid momentum shifts. Because of the limited duration, batters tend to play aggressively, aiming for boundaries rather than survival. However, this risk-heavy approach is tempered by wicket loss: teams that lose early wickets often shift to a more conservative style of play to preserve their remaining batters. This interplay between aggression and caution, modulated by overs remaining, runs scored, and wickets lost, gives rise to rich scoring patterns that evolve ball by ball.

Accurately modeling and forecasting the evolution of an inning, especially after a partial trajectory is observed (e.g., after 8 overs), can be extremely valuable for decision-makers. Coaches may use such models to assess expected scores and adjust strategy; broadcasters and commentators can enhance viewer engagement; and sportsbooks or bettors can price game outcomes more accurately.

In this project, we aim to forecast the final score of an inning given partial information observed up to a specified intervention point. We explore various statistical and machine learning approaches that leverage both the inning's own trajectory and a large database of historical T20 innings to generate accurate and interpretable predictions.

3 Problem Statement

In this project, we aim to model and forecast the progression of a T20 cricket inning using partially observed data. Each inning is represented by a time series of cumulative metrics, specifically total runs scored and total wickets lost, recorded ball by ball. Collectively, our dataset spans N innings, each with up to $T = 120$ balls, and $K = 2$ key metrics. We organize this information into a 3-dimensional tensor $\mathcal{X} \in \mathbb{R}^{N \times T \times K}$, where $\mathcal{X}_{i,t,k}$ captures the cumulative value of metric k for inning i at ball t.

Let T_0 denote a designated intervention point within an inning. We define the trajectory from ball 0 to T_0 as the **pre-intervention** segment, and from $T_0 + 1$ to T as the **post-intervention** segment. Our objective is to accurately forecast the post-intervention trajectory of a given inning i^* — referred to as the **treatment unit** — using its own pre-intervention history along with the complete trajectories of the remaining $N - 1$ innings, which serve as **donor units**.

For example, consider a scenario where Team A is batting first and has reached a score of 60 runs for the loss of 2 wickets after 8 overs (i.e., 48 balls). Given this partial trajectory, we seek to predict the team's remaining progression over the next 72 balls. To do so, we compare its pre-intervention behavior with similar innings from historical data and synthesize plausible future outcomes using one of several modeling techniques.

This setup mirrors ideas from synthetic control and sequential forecasting frameworks, where the goal is to infer future or counterfactual behavior by borrowing strength from a large, structured archive of past instances.

4 Data Collection

The primary data for this project was sourced from cricsheet.org, an open-access platform that provides detailed ball-by-ball records of professional cricket matches in JSON format. We downloaded files for over 18,500 matches played over the past 20 years, including both international fixtures and major domestic T20 leagues such as the Indian Premier League (IPL), Big Bash League (BBL), and others.

Each JSON file contains rich structured information, including team rosters, innings summaries, and individual ball events (e.g., runs scored, wickets taken, extras, and player dismissals). Using a custom Python parser, we extracted and cleaned the relevant game-level and delivery-level features. This involved resolving player identifiers, inferring innings structure, handling missing or corrupt entries, and standardizing cumulative run and wicket tallies across deliveries.

To manage the data at scale, we stored it in a MySQL relational database hosted on AWS. Tables were designed to hold match metadata, player mappings, team configurations, and full ball-by-ball logs, enabling flexible querying for downstream analysis. We used `SQLAlchemy` and `pymysql` in Python for structured interaction with the database.

For the purposes of modeling, we focused specifically on T20-format matches and extracted all innings into a 3-dimensional tensor of shape $(22312, 120, 2)$, where:

- The first dimension corresponds to each individual inning.
- The second dimension spans the 120 legal deliveries (balls) of a full 20-over inning.
- The third dimension holds two cumulative metrics: total runs scored and total wickets lost.

This tensor was saved as a `.npz` file, a compressed NumPy archive format, to allow for efficient loading and access during model training and evaluation. This setup ensures high computational efficiency while retaining all the temporal resolution of the original ball-by-ball data.

The tensor can be somewhat visualized using this graphic:

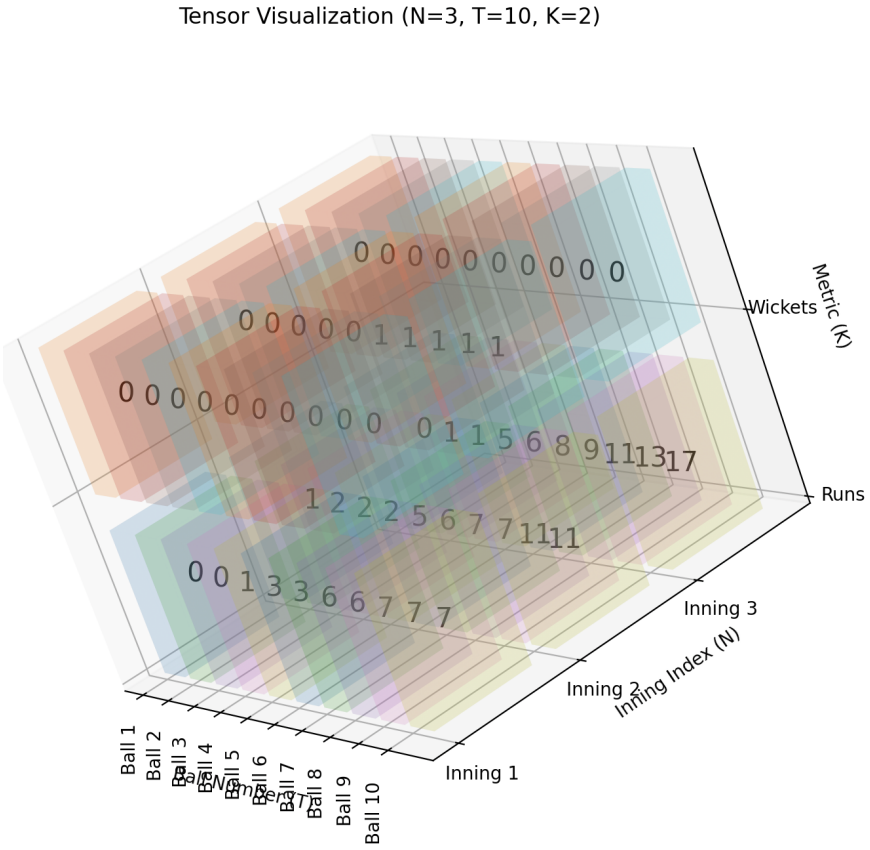

Figure 1: Visualization of Tensor.

The SQL relational database structure is as follows:

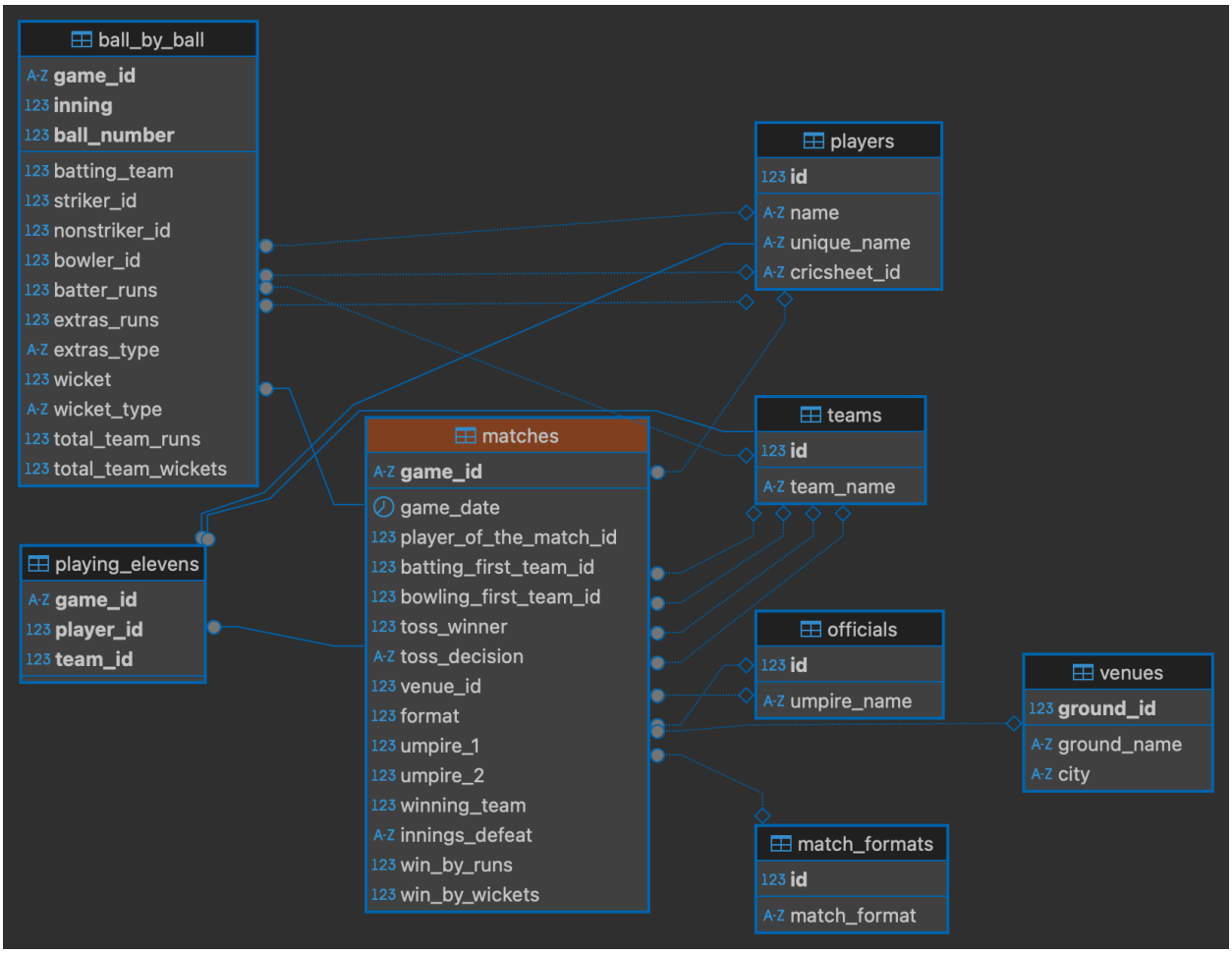

Figure 2: mySQL Database Structure.

Note that the data collected in the database includes all types of game formats (T20, ODI, Test, etc.), and attempts to retain as much information as possible from the raw JSON files. Even though the rest of our analysis focuses on solely the T20 games, we believe it will be useful to store as much information as possible, so that if we would like to do further analysis in the future or perhaps a comparative analysis between formats, we will have that data readily available.

5 Methods

This section explores the various forecasting methods used:

5.1 Linear Regression

We begin by implementing a linear regression model as a baseline to forecast the final score of an inning based on early-inning performance. Specifically, we model the cumulative runs at a future ball t_{target} using cumulative runs observed at a set of earlier balls $t_1, t_2, \ldots, t_p$, where each $t_i < t_{\text{target}}$.

Our dataset is organized as a 3-dimensional tensor $\mathcal{X} \in \mathbb{R}^{N \times T \times 2}$, where:

- N is the number of innings (in our case, 22,312),

- $T = 120$ is the number of deliveries in a full T20 inning,
- The third dimension represents the metrics of interest: total cumulative runs and total cumulative wickets at each ball.

For each inning i, we define the predictor vector as:

$$\mathbf{x}_i = \begin{bmatrix} \mathcal{X}_{i,t_1,0} \\ \mathcal{X}_{i,t_2,0} \\ \vdots \\ \mathcal{X}_{i,t_p,0} \end{bmatrix} \quad \text{and the target as} \quad y_i = \mathcal{X}_{i,t_{\text{target}},0}$$

We fit the linear model:

$$y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{i,j} + \epsilon_i$$

In our experiment, we chose predictors at ball numbers $t_1 = 24$, $t_2 = 36$, $t_3 = 48$, and $t_4 = 60$, corresponding to the cumulative runs at the end of overs 4, 6, 8, and 10. We evaluate the model's predictive performance at:

- $t_{\text{target}} = 72$: end of 12 overs,
- $t_{\text{target}} = 90$: end of 15 overs.

The resulting R^2 values, which represent the proportion of variance in the target explained by the predictors, serve as baseline metrics to compare against more complex models later in the paper.

We predict the 12th over score from the score at overs 4, 6, 8, and 10, and we get an R^2 **of 0.93**. Next, we predict the 15th over score using the same predictors, and we get an R^2 **of 0.80**. We see that forecasting small distances into the future using linear regression is quite accurate, however as we extend the forecast, the accuracy can quickly drop.

5.2 mRSC (Multi-dimensional Robust Synthetic Control)

Inspired by the method proposed by Vishal Misra et al., we explore a **robust synthetic control** framework for forecasting cumulative run trajectories in T20 cricket. The method builds on the idea of estimating **low-rank latent structure** across innings, by leveraging similarities in the **pre-intervention trajectory** to predict the **post-intervention trajectory**.

We fix an **intervention point** T_0, and reshape the data for a single metric k (say, runs) into a 2D matrix:

$$X_k \in \mathbb{R}^{N \times T} \quad \text{with each row } X_k^{(i)} \text{ representing an innings trajectory}$$

We define:

- $X_k^{(i)} = [X_{i,0,k}, X_{i,1,k}, \ldots, X_{i,T_0-1,k} \,|\, X_{i,T_0,k}, \ldots, X_{i,T-1,k}]$
- Let $X^{\text{pre}} \in \mathbb{R}^{N \times T_0}$, the matrix of pre-intervention trajectories,
- Let $X^{\text{post}} \in \mathbb{R}^{N \times (T-T_0)}$, the post-intervention matrix (with some missing entries).

Our goal is to **forecast the missing post-intervention trajectory for a treatment unit**, say row $i = 1$, using the remaining $N - 1$ donor units.

5.2.1 Low-Rank Matrix Completion with SVD

We concatenate the donors' trajectories (excluding the treatment unit) into a matrix $Z \in \mathbb{R}^{(N-1)\times T}$, where we assume the underlying matrix has a low-rank structure due to shared dynamics across innings.

We apply **Singular Value Decomposition (SVD)** to the pre-intervention part of this matrix:

$$Z^{\text{pre}} = U\Sigma V^\top$$

To remove noise and retain only the dominant structure, we threshold singular values using a hyperparameter λ, and construct a **rank-r** approximation:

$$Z^{\text{pre}} \approx U_r \Sigma_r V_r^\top$$

We then project the treatment unit's pre-intervention vector X_1^{pre} into this low-rank donor space and solve for weights $w \in \mathbb{R}^{N-1}$ such that:

$$X_1^{\text{pre}} \approx w^\top Z^{\text{pre}}$$

The same weights are then applied to donor **post-intervention** vectors to synthesize the **counterfactual trajectory**:

$$\widehat{X}_1^{\text{post}} = w^\top Z^{\text{post}}$$

5.2.2 Summary of Steps

1. **Construct** donor matrix $Z \in \mathbb{R}^{(N-1)\times T}$
2. **Split** into Z^{pre} and Z^{post} around the intervention time T_0
3. **Apply SVD** and truncate using singular values above threshold λ
4. **Project** treatment trajectory onto donor space to estimate weights
5. **Reconstruct** forecast via donor weights and Z^{post}

5.2.3 Notes on Robustness

To handle missing values (e.g., due to early innings termination), we apply **masking** during the decomposition and weight fitting steps, ensuring that only fully observed donor trajectories are included in the low-rank model.

This approach allows for a nonparametric, interpretable reconstruction of scoring dynamics, grounded in empirical similarities across games.

Here are some results we got when forecasting on a specific inning, using $T_0 = 60$ as the intervention point. The structure of this algorithm allows us to forecast not only runs, but also wickets as the game progresses past the intervention point. Notice how the future trajectory efficiently captures the flow of the game, as the curve becomes slightly steeper towards the end of the innings, suggesting high scoring rates in the death overs as typically seen in a T20 game especially when there are enough wickets in hand.

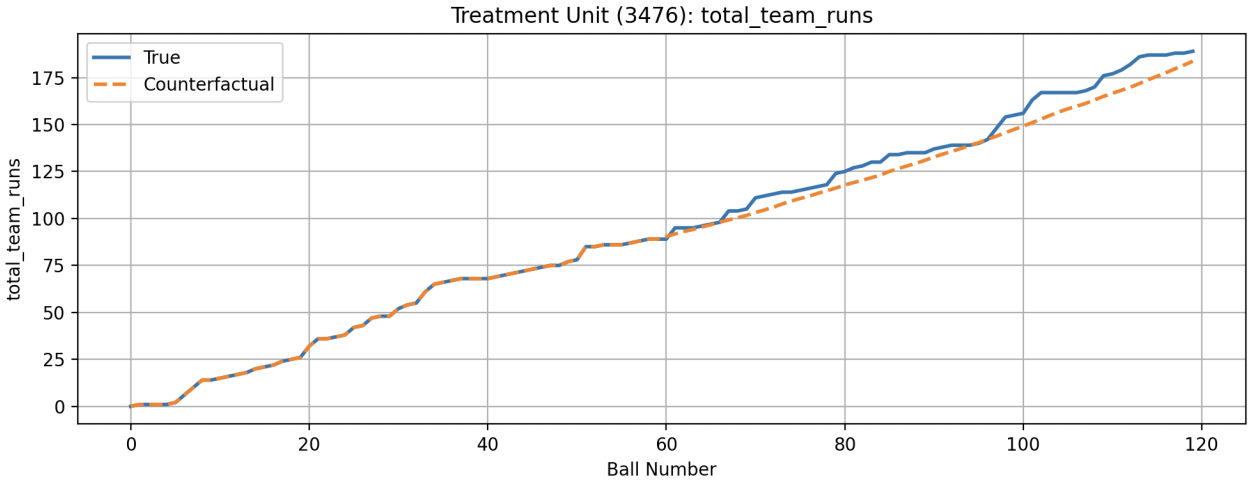

Figure 3: mRSC Runs Forecast on a specific inning.

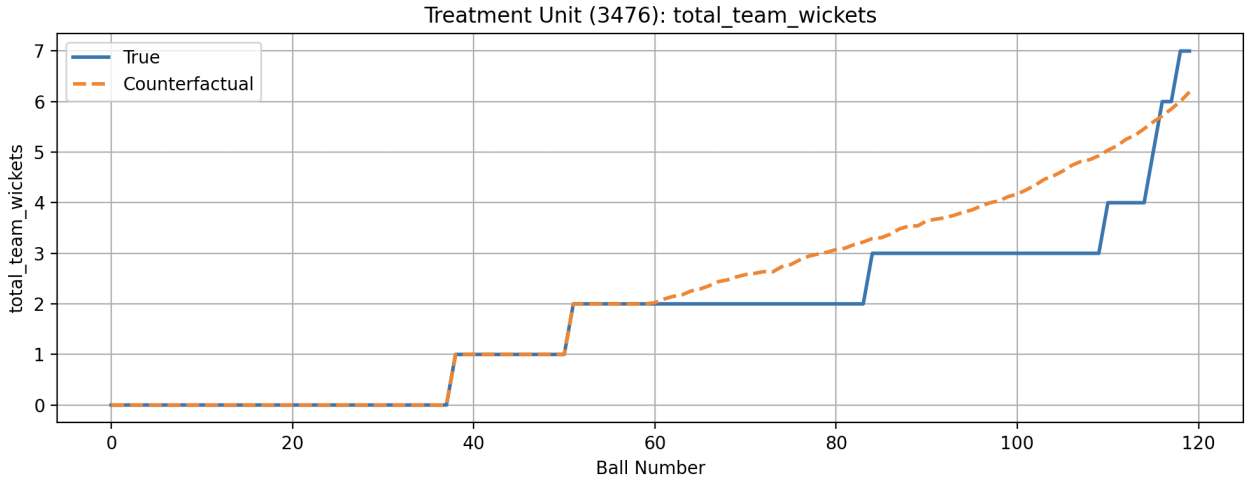

Figure 4: mRSC Wickets Forecast on a specific inning.

5.3 ARIMA

We fit ARIMA(2,1,2) models to individual cumulative run time series up to the intervention point and forecast forward. The model captures autocorrelation and local trends in the run sequence.

To capture short-term autocorrelations and temporal dynamics in scoring, we fit an Autoregressive Integrated Moving Average (ARIMA) model of order $(2, 1, 2)$ to the **cumulative run trajectory** of each inning up to a predefined intervention ball T_0. The fitted ARIMA model extracts time-series-specific coefficients that help characterize each innings' momentum and volatility.

5.3.1 ARIMA Model Formulation

Let y_t denote the cumulative runs at ball t. The differenced series $\Delta y_t = y_t - y_{t-1}$ (since $d = 1$) is modeled as:

$$\Delta y_t = \phi_1 \Delta y_{t-1} + \phi_2 \Delta y_{t-2} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \varepsilon_t$$

where:

- ϕ_1, ϕ_2 are AR coefficients,
- θ_1, θ_2 are MA coefficients,
- $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$ is white noise.

From each fitted ARIMA model, we extract:

- AR coefficients (ϕ_1, ϕ_2),
- MA coefficients (θ_1, θ_2),
- Residual mean and standard deviation,
- Estimated variance σ^2.

We augment these ARIMA-based features with additional statistics such as:

- Mean and standard deviation of pre-T_0 cumulative runs,
- Initial and final cumulative runs up to T_0,
- Number of wickets lost at T_0.

5.3.2 Forecasting Final Scores

Let $\mathbf{x}_i \in \mathbb{R}^d$ denote the extracted feature vector for the i-th inning. We define the **target** as the number of runs scored between balls $T_0 + 1$ and 120. We then train a Random Forest regression model $\hat{f}(\mathbf{x})$ using these global features across many innings:

$$\text{RemainingRuns}_i = \hat{f}(\mathbf{x}_i)$$

The predicted final score for inning i is then:

$$\widehat{\text{FinalScore}}_i = \mathcal{X}_{i,T_0,0} + \hat{f}(\mathbf{x}_i)$$

5.3.3 Performance

We trained the global model on 500 innings and used a 20% test split. The model achieved:

- **Mean Absolute Error (MAE)**: approximately 9–11 runs across held-out innings.
- This performance reflects a strong ability to summarize an innings' momentum using the ARIMA signature and extrapolate the final total.

We visualize actual vs. predicted cumulative run trajectories for sample innings:

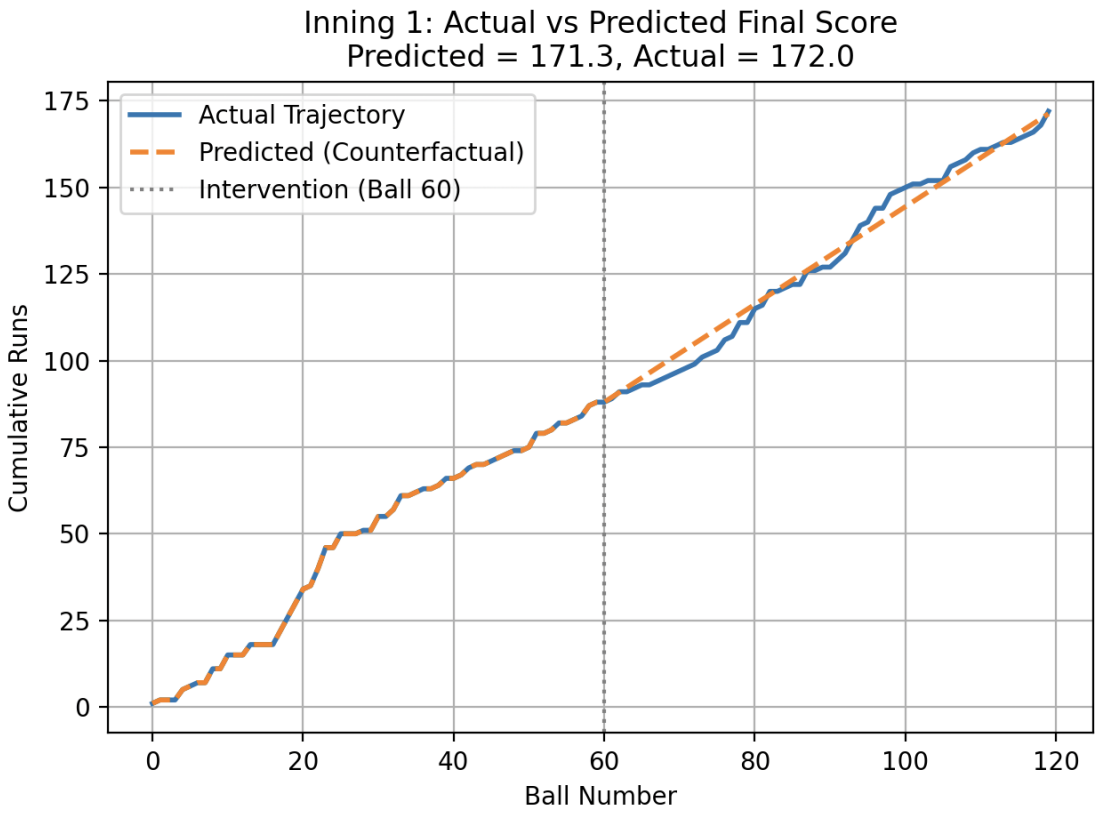

Figure 5: Predicted score for an inning using ARIMA.

We can also look at the RMSE distributions when fitting the ARIMA model to a large number of innings, again using $T_0 = 60$ as our intervention point. The graphic shows that while the majority of predictions have a low RMSE and are quite accurate, there are a few outliers which greatly skew the mean error, showing that there are rare cases in which the model performs extremely poorly.

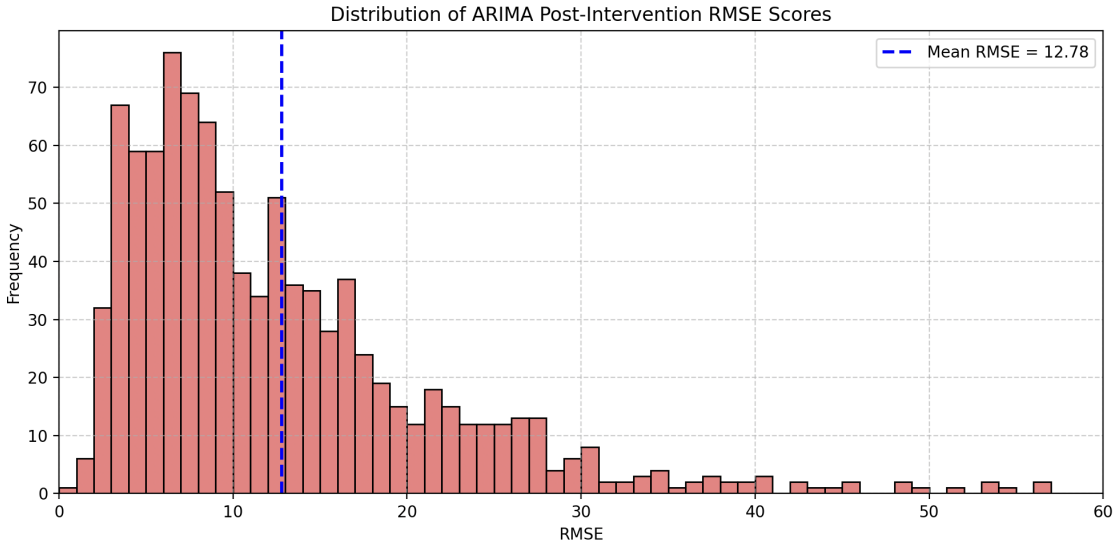

Figure 6: Distribution of RMSE for ARIMA model.

5.4 K-Nearest Neighbors Simulation

5.4.1 Brute Force Approach

For this simulation, we implemented a weighted distance metric prioritizing wickets ($\lambda = 20$) to match treatment innings with similar donor units. At each ball after T_0, we identify the top 50 nearest neighbors, sample their next-ball delta, and simulate forward until the 120th ball or the 10th wicket.

The distance function is:

$$D(i,j) = \sum_{t=1}^{T_0} (\mathcal{X}_{i,t,1} - \mathcal{X}_{j,t,1})^2 + \lambda (\mathcal{X}_{i,t,2} - \mathcal{X}_{j,t,2})^2$$

We then ran repeated random simulations for a given inning and intervention point, and plotted the results, shown below with $T_0 = 60$ and $K = 50$. One major issue that appears in this algorithm is the computational burden. The reason for this is because for each simulated ball, we evaluate the distance metric between the treatment unit and all other $N - 1$ donor units and then sort them. The simulations themselves give quite satisfactory results, however it was not possible to do a full analysis on its accuracy due to its slow runtime. A plot for the results can be seen below.

The choice to use $\lambda = 20$ is due to the fact that wickets are much rarer to see than runs. We can see run values of up to 200 in cricket games, but wickets can only go up to a value of 10.

Further analysis can be done to find a better distance metric, and one option is to use the dynamic relationship between overs remaining and wickets lost in the DLS method to come up with a variable λ value.

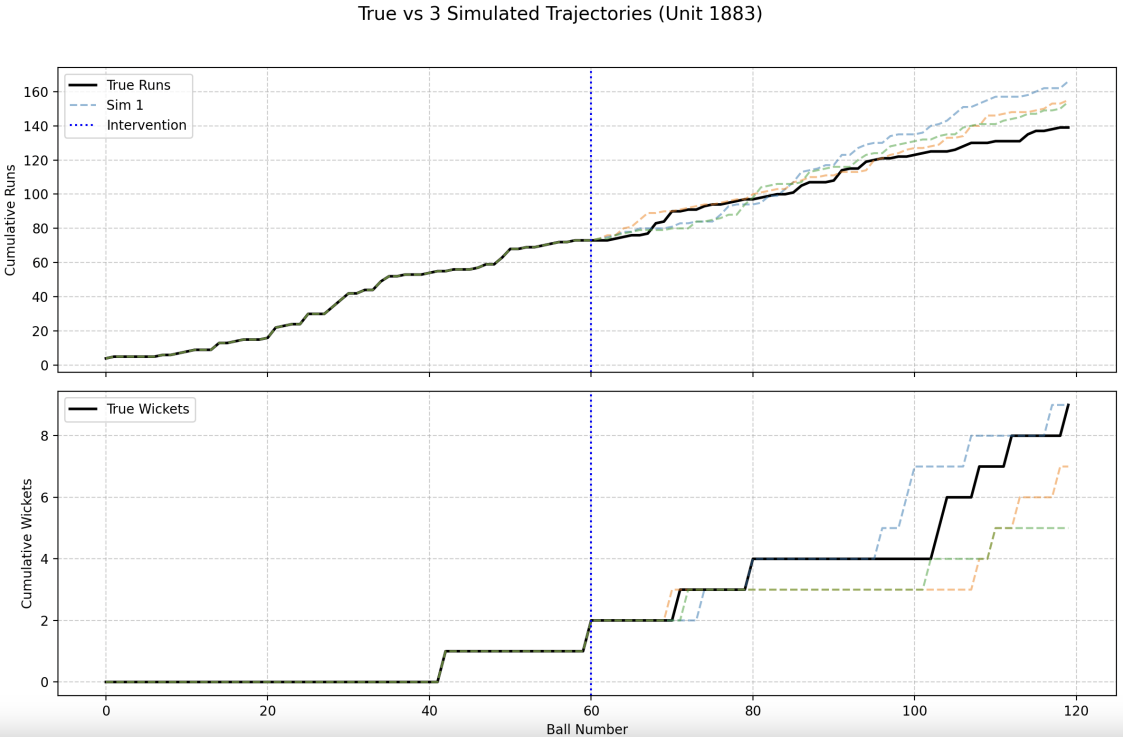

Figure 7: KNN simulations for an inning.

5.4.2 Hollow Distance Matrix Approach

To address the computational inefficiency of recalculating distances after each simulated ball, we implemented an optimized version of the KNN algorithm that precomputes and stores a **hollow self-distance matrix**. This matrix has dimensions $N \times N$, where N is the number of innings in the dataset. The (i, j)th entry stores the weighted Euclidean distance between the ith and jth innings up to the intervention point T_0, using the same metric $D(i, j)$:

Because we are comparing each inning with itself, the diagonal entries of this matrix are all zero, hence the term *hollow*. Once computed, this distance matrix allows us to quickly identify the top K nearest neighbors for any treatment inning without re-evaluating distances at every step. These neighbors are then locked in for the rest of the simulation, and at each subsequent ball after T_0, we randomly sample a next-ball delta (runs and wickets) from the corresponding position in one of the nearest neighbors.

This "precomputed neighbors" approach significantly reduces runtime and computational overhead, especially when simulating many innings or running repeated forecasts. While this comes at the cost of adaptability, as we no longer update the donor pool dynamically after each simulated ball, the tradeoff is reasonable when speed and scalability are a priority.

With this simplified approach, we have the ability to run many more simulations for a given treatment unit. Running more simulations and then averaging the end score after 20 overs for all simulations will give us a much better estimate of what the final score should be. One downside of this approach is that this self-distance matrix only applies for a given intervention ball $T_0 = 60$. If we want to forecast for $T_0 \neq 60$, we would have to create a new matrix for that new T_0.

The image below shows an example on a given inning, where we apply the precomputed distance matrix method to run 50 simulations of the counterfactual with $K = 50$. We record the final cumulative runs of each simulation and average them to get a final estimate of runs at the end of the innings and compare it with the true result. For this specific inning, we get a final prediction of 174.78 runs, compared to a true final score of 173, making the residual only 1.78 runs.

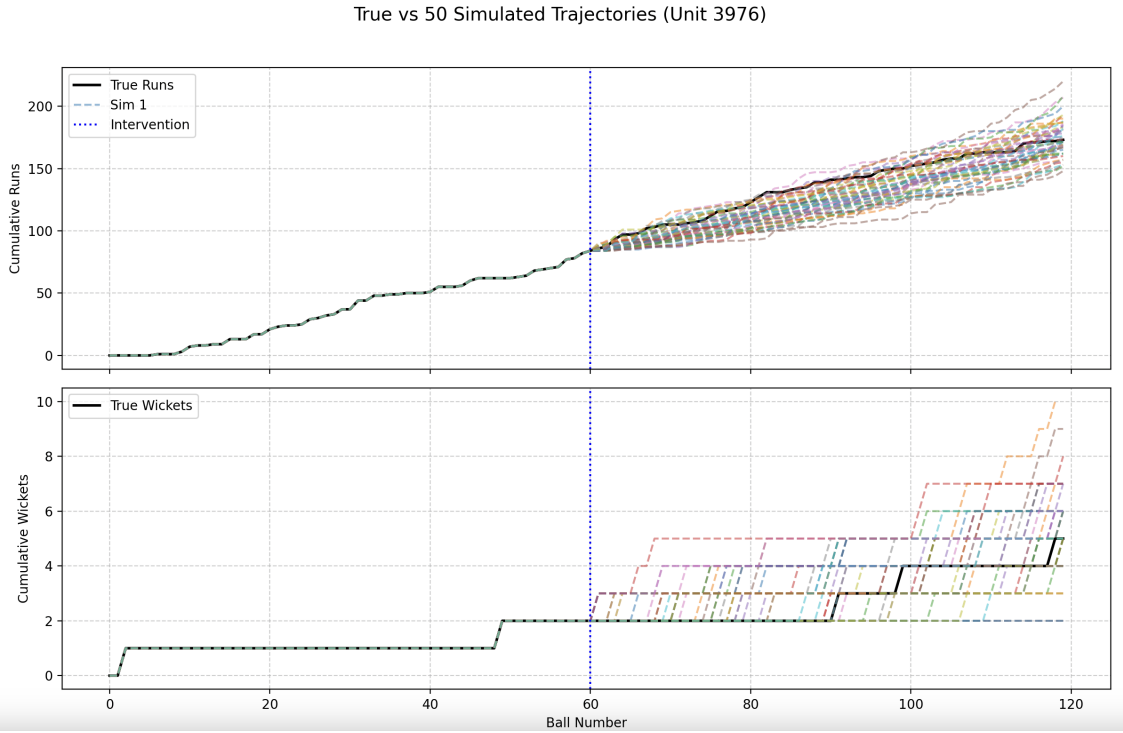

Figure 8: KNN simulation using distance matrix.

5.5 Mactivation (Multiple Activation)

Using Professor Zes's `mactivate` library in R, we predict final scores after 20 overs in an inning from an intervention point sometime earlier in the inning.

To complement our other forecasting methods, we implemented a multiple activation-based regression model using the `mactivate` R package developed by Professor Zes. This model is designed to capture complex relationships in the data through multiple activation layers. We predict final scores after 20 overs in an inning from an intervention point sometime earlier in the inning.

5.5.1 Data Preparation

We constructed a feature matrix using cumulative runs and wickets at several key points in the innings (e.g., 36, 48, 66, 78, and 90 balls), along with binary indicators such as whether more than two or four wickets remained at the 90-ball mark. The target variable was the final score at ball 120. All features were standardized using the mean and standard deviation of the training set.

We split the full dataset of over 22,000 T20 innings into training and test sets, using the final 4,000 most recently played innings as the held-out evaluation group. The standardized input matrix X and activation covariates U were used to fit the model, with the following structure:

- X: Main covariates (interactions, nonlinear terms)
- U: Subset of covariates used in activation layers

5.5.2 Model Specification and Training

We trained the hybrid activation model using the `f_fit_hybrid_01()` function, with key control parameters passed via `f_control_mactivate()`:

- `param_sensitivity = 10^{11}`: sensitivity rate
- `escape_rate = 1.0001`: Encourages movement across loss landscape
- `force_tries = 500-1000`: Enforces additional local exploration
- `m_tot = 14`: Number of activation features used
- `lambda`: Ridge regularization penalty

The model was trained for multiple configurations of input structure, tuning the predictor set and activation matrix. Optimization was done with high precision (`tol = 1e-24`). A downside for this algorithm was its slow runtime as it looked to optimize step by step in a high dimensional space. It took a couple of hours to run.

5.5.3 Performance Evaluation

For each model, we computed predictions for the test set across 14 activation gates $mtotal = 14$, and recorded the RMSE at each depth. The best configuration achieved a test-set R^2 of approximately 0.9232, outperforming models such as random forests.

This approach balances flexibility and interpretability, offering an appealing framework for future extensions where wickets and runs interact in complex ways. Unlike tree-based methods, the activation model provides smoother, more controlled generalization and clearer interpretability across activation depth.

6 Results

- Linear regression predicting 12th over score from overs 2, 4, 6, 8, 10: $R^2 = 0.93$
- Linear regression predicting 15th over score from same predictors: $R^2 = 0.80$
- mRSC produced reasonable counterfactuals in majority of innings, and was often able to capture the subtle changes in game flow and the dynamic relationship between runs and wickets. The RMSE across innings when using an intervention point at 10 overs and predicting up until over 20 was about 8.21 runs. This was better than the RMSE seen from the ARIMA model.

- ARIMA model performed well in smooth innings, but struggled with irregularity (high RMSE outliers). With a mean RMSE across innings of about 12.78 runs, the model can be trusted for most innings. However, the relative simplicity of the model when compared to mRSC results in the ARIMA model sometimes not being able to capture all information from the pre-intervention data.

- KNN model produced decently accurate results with natural variability, however the distance metric can be improved and computational efficiency can also be improved. Due to the slow running of this algorithm, it was not possible to compute an overall metric for its accuracy, though we did manage to make the algorithm more efficient through a hollow self-distance matrix.

- The mactivate model achieved an R^2 of 0.8754 when predicting the final score at 20 overs from an intervention point of 10 overs, and achieved an R^2 of 0.9232 when predicting over 20 score from over 15.

Overall, the mRSC and mactivate models show the highest promise, yielding very accurate results. The KNN algorithm using pre-computed matrices for intervention points is also a relatively accurate approach.

7 Conclusion

In this project, we explored a range of statistical and machine learning methods to forecast the final score of a T20 cricket innings using only partial game information available at a specified intervention point. Starting from simple linear regression models to more advanced approaches like ARIMA time series forecasting, robust synthetic control, and K-nearest-neighbor-based trajectory simulation, we evaluated the predictive performance and interpretability of each technique.

Our experiments show that even relatively simple models using early-over scores can yield surprisingly strong baseline predictions. However, the use of full trajectory-based donor matching and sequential simulation opens up far more nuanced and dynamic forecasting potential. Each method brings unique strengths: linear regression offers interpretability, ARIMA captures autocorrelation, and donor-based simulations adapt flexibly to the current game state.

Looking forward, there are several promising directions for extending this work. One avenue is to incorporate additional variables and features such as batter and bowler information, team compositions, venues, and recent team performances, as they can all be extracted from the underlying SQL database. These could significantly enhance predictive accuracy. Another direction is to integrate this forecasting engine into real-time tools for coaches, analysts, and broadcasters to aid in match strategy and commentary.

A future direction to focus in would be to compare the predictive accuracy of these models with that of sportsbooks. If we are able to find that a certain model is able to forecast more accurately than the sports book lines at a rate that can yield us long run profits, we could use this tool to earn money for ourselves, or we can also monetize this tool by providing it to other individuals for a subscription price.

8 References

[1] **Cricsheet**. "Open-source ball-by-ball cricket data." Available: https://cricsheet.org

[2] M. Amjad, V. Misra, D. Shah, and D. Shen, "mRSC: Multi-Dimensional Robust Synthetic Control," *arXiv preprint arXiv:1905.06400*, 2019. Available: https://arxiv.org/pdf/1905.06400

[3] D. Zes, *mactivate: A Package for Activation-Based Regression Models*, 2022. Available: https://cran.r-project.org/web/packages/mactivate/mactivate.pdf

[4] I. Bhattacharya, R. Ghosal, S. Ghosh, "A Statistical Exploration of Duckworth-Lewis Method Using Bayesian Inference", 2018. Available: https://arxiv.org/abs/1810.00908v1