# Predicting Cricket Scores: A Machine Learning Approach with Ensemble Learning Potential

Taha Hasnain Raza
*Department of Computer Engineering*
*Information Technology University*
Lahore, Pakistan
thraza99@gmail.com

***Abstract*—A machine learning project successfully predicted cricket scores (run rate, total score) for T20 and ODI matches using Random Forest, Decision Tree, and KNN regression. All models achieved impressive accuracy (RMSE ¡ 30), with Random Forest excelling. This paves the way for further exploration of ensemble methods, which combine the strengths of multiple models, and hyper-parameter tuning for optimized performance. These advancements hold the potential to significantly enhance cricket score prediction accuracy in the future.**

***Index Terms*—Prediction, Regression, Scores, RMSE, KNN, SciKitLearn**

## I. Introduction

Cricket, a sport adored by billions, thrives on the thrill of the unpredictable. However, the ability to anticipate scores offers valuable insights for fans, analysts, and even teams. This project tackles this challenge by creating a novel Cricket Score Prediction System using machine learning.

We explore the potential of various regression models, including Random Forest, Support Vector Machine (SVM) Regression, Decision Tree Regression, K-Nearest Neighbors (KNN) Regression, and Linear Regression. These models will be trained on extensive datasets encompassing both T20 and ODI formats. Our goal is to achieve a Root Mean Squared Error (RMSE) below 25, indicating good prediction accuracy.

Exceeding expectations, all models achieved the targeted RMSE, with Random Forest excelling. This paves the way for further exploration. We aim to leverage ensemble methods, combining the strengths of multiple models, and hyperparameter tuning for optimal performance.

This project strives to develop a practical and impactful Cricket Score Prediction System. By harnessing machine learning's power, this system has the potential to revolutionize cricket analysis and appreciation, adding a new layer of engagement for fans, analysts, and teams.

## II. Related Work

There has been significant prior work in the application of machine learning techniques to cricket score prediction. Some notable examples include:

*Applications of Machine Learning in Cricket: A Systematic Review [1]*

This systematic review consolidates various machine learning applications in cricket, highlighting key methodologies and outcomes. It covers techniques ranging from player performance prediction to match outcome forecasting. The review also identifies gaps and future directions in the field, providing a comprehensive overview of current advancements.

*Cricket Team Prediction Using Machine Learning Techniques [2]*

This study focuses on predicting the winning team in cricket matches using various machine learning algorithms. By analyzing historical match data, including team compositions and match conditions, the authors developed models to predict match outcomes. Their findings demonstrate the effectiveness of machine learning in enhancing prediction accuracy over traditional methods.

*Player's Performance Prediction in ODI Cricket Using Machine Learning Algorithms [3]*

In this research, the authors applied machine learning algorithms to predict individual player performance in One Day International (ODI) cricket. They utilized features such as player statistics, match conditions, and opposition strength. The study showcased how machine learning can be leveraged to forecast player contributions, aiding in strategic planning and team selection.

*Live Cricket Score Prediction Web Application Using Machine Learning [4]*

This paper presents the development of a web application for live cricket score prediction using machine learning. The application integrates real-time match data and applies predictive models to estimate ongoing match scores. The authors demonstrate the application's potential in providing timely and accurate score predictions, enhancing the viewing experience for cricket enthusiasts.

## References

## III. Materials and Methods

**Data Sources**: Accurate and diverse data sources are crucial for building a reliable cricket score prediction model. This

project leverages public repositories and cricket websites to gather comprehensive datasets.

**GitHub Repositories:** Various public repositories on GitHub provide cricket data in CSV format. These datasets include match data, player statistics, and historical scores. Some notable repositories include: Contains detailed match data and player statistics. [5] Offers comprehensive data on cricket world cups. [6] Provides additional cricket datasets for analysis. [7]

**Cricket Websites:** Websites like ESPN Cricinfo are invaluable sources of cricket data. They offer up-to-date information on matches, player performances, and historical records. The data collected from these websites is often more granular and current compared to public repositories. [8] - ESPN Cricinfo, a leading cricket website providing detailed match and player data.

**Data Collection:** Data collection is the cornerstone of this project. It involves gathering data from multiple sources to ensure diversity and comprehensiveness.

**Match Data:** Includes detailed records of past matches, covering various aspects such as scores, number of overs, wickets taken, and other match events. This data helps in understanding historical trends and patterns.

**Player Statistics:** Encompasses player-specific data like batting averages, bowling figures, strike rates, and fielding records. This data is crucial for individual performance analysis and predicting future performances.

**Environmental Data:** Weather conditions, pitch reports, and venue details are also collected as they significantly impact match outcomes. This data is often obtained from sports websites and meteorological sources.

Accurate data collection forms the foundation for analysis. It provides the raw material for uncovering trends, informing decisions, and ultimately driving successful outcomes.

**Data Cleaning and Preprocessing:** Data cleaning and preprocessing are critical steps to ensure the reliability and accuracy of the cricket score prediction model. Raw data often contains inconsistencies, errors, and missing values that need to be addressed before model training.

**Handling Missing Values:** Missing data can lead to inaccurate predictions. Techniques such as imputation (filling in missing values with mean, median, or mode) or removing records with missing values are used to handle this issue.

**Data Consistency:** Ensuring uniform data formats, such as consistent date formats and standardized numerical values, is crucial. For example, dates should follow a single format (e.g., YYYY-MM-DD), and numerical values should be rounded to specific decimal places where necessary.

**Removing Duplicates:** Duplicate records can skew analysis and model training. Identifying and removing duplicates ensures that each data point is unique and contributes accurately to the model.

**Outlier Detection and Treatment:** Outliers can significantly impact the performance of machine learning models. Statistical methods such as Z-scores or IQR (Interquartile Range) are used to detect and handle outliers.

**Normalization:** Scaling features to a consistent range (e.g., 0 to 1) helps in improving the performance of machine learning algorithms. Techniques like Min-Max scaling or Z-score normalization are commonly used.

**Encoding Categorical Data:** Converting categorical data into numerical format using techniques such as one-hot encoding or label encoding. This step is crucial for algorithms that require numerical input.

Data cleaning ensures the reliability of your cricket score prediction model by removing inconsistencies, errors, and missing values. This creates a foundation of high-quality data, essential for accurate model predictions.

## IV. Model Selection and Training

**Random Forest Regression:** This ensemble learning technique combines multiple decision trees for improved prediction accuracy.

**Decision Tree Regression:** This model uses a tree-like structure to make predictions. It's interpretable, allowing you to understand its reasoning.

**K-Nearest Neighbors (KNN) Regression:** This method predicts the value of a new data point based on the k nearest neighbors in the training data. Different distance metrics can be used.

Training each model involves preparing my cricket data. This means handling missing values, converting categorical data (like teams) into numerical formats, and ensuring numerical features (like scores) are on a consistent scale.

Splitting the pre-processed data into training, validation, and testing sets.

Train each model (Random Forest, Decision Tree, KNN) using the training set. Use the validation set to fine-tune hyperparameters (parameters of the models) for optimal performance.

While our initial models showed promise, there's always space for improvement. This section explores how we used validation sets to fine-tune the hyperparameters of each model (Random Forest, Decision Tree, and KNN Regression), unlocking their full potential for accurate cricket score prediction.

Imagine a recipe for a cake. The core ingredients (algorithms) are vital, but the final product depends on the precise amounts used (hyperparameters). These parameters influence a model's behavior, and are set before training. For instance, a decision tree's hyperparameter might determine the maximum depth allowed for the tree's branches.

Here's where validation sets come in. This separate dataset acts as a safe space for experimentation. We use it to test different hyperparameter values for each model. We train the model with various configurations, observing how they affect the RMSE on the validation data (not the final testing set).

**Model Evaluation**

Evaluate the performance of each model on the testing set using Root Mean Squared Error (RMSE) as the primary metric. Lower RMSE indicates better prediction accuracy.

**Magnitude of Errors:** RMSE focuses on the magnitude of prediction errors rather than their direction. In cricket score

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}}$$

Fig. 1. Root Mean Squared Error

prediction, large errors can significantly impact decisions and strategies. RMSE penalizes larger errors more heavily by squaring them, which aligns with our objective of minimizing substantial inaccuracies.

**Interpretability:** RMSE is measured in the same units as our target variable—cricket scores. This makes it easy to interpret the results. A lower RMSE value directly translates to a smaller average difference between predicted and actual scores, providing a clear and intuitive understanding of model performance.

**Sensitivity to Large Errors:** By squaring the errors before averaging, RMSE gives more weight to larger discrepancies. This property is particularly beneficial for our project since large prediction errors are more detrimental than smaller ones. RMSE helps us prioritize the reduction of these significant errors.

**Application of RMSE Across Different Models**

**Support Vector Regression (SVR)** SVR is a powerful tool for regression tasks, particularly when dealing with non-linear data. Using RMSE as the evaluation metric for SVR allowed us to fine-tune the model parameters effectively. By focusing on minimizing RMSE, we were able to achieve a model that reduces significant prediction errors, leading to more accurate and reliable cricket score predictions.

**Linear Regression:** Linear Regression serves as a baseline model in our evaluation. RMSE was crucial in comparing the performance of Linear Regression against more complex models. Despite its simplicity, using RMSE highlighted the limitations of Linear Regression in handling non-linearity and large variance in cricket scores, guiding us to explore more advanced models.

**Random Forest:** Random Forest, being an ensemble learning method, benefits significantly from RMSE as it helps in fine-tuning the model to achieve better accuracy. RMSE's sensitivity to large errors ensured that the Random Forest model minimized substantial discrepancies, resulting in robust predictions.

**K-Nearest Neighbors (KNN) Regression:** KNN Regression's performance heavily depends on the choice of 'k' and distance metrics. RMSE provided a clear criterion for optimizing these parameters. By minimizing RMSE, we could determine the optimal 'k' value, leading to improved prediction accuracy and model reliability.

**Decision Trees:** Decision Trees can easily overfit the training data. RMSE helped in pruning the trees and selecting the right depth to avoid overfitting while maintaining accuracy. The focus on reducing RMSE ensured that the Decision Trees model performed well on unseen data, providing reliable cricket score predictions.

**Model Integration**

Integrated the best performing model into the Cricket Score Prediction System for practical use.

Your Cricket Match Simulator is currently a Python script that simulates cricket matches without a graphical interface. It utilizes Object-Oriented Programming (OOP) concepts by defining classes like Team and potentially Match. These classes encapsulate data (team names, match type) and functionalities (selecting teams, simulating a match). You leverage libraries like openpyxl (potentially) to interact with data sources like Excel files for team information. The script allows users to choose teams and other parameters through prompts in the terminal, but it doesn't display results visually. The next step involves integrating this logic with a web framework like Flask to create a web application where users can interact with the simulator through a web interface.

**Software and Hardware Requirements**

For the successful implementation and execution of the cricket score prediction models, certain software and hardware requirements are essential. These requirements ensure that the computational tasks, data processing, and model training are performed efficiently. Below is a detailed list of the recommended hardware and software specifications:

**Hardware Requirements:**

**Processor:** A standard computer with at least a recent Intel Core i5 processor or an equivalent AMD Ryzen 5 processor. These processors provide sufficient computational power for handling data preprocessing, model training, and evaluation tasks efficiently.

**Memory (RAM):** A minimum of 8GB of RAM is recommended to manage the memory-intensive operations involved in handling large datasets and running machine learning algorithms. More RAM may be beneficial for improving performance, especially when working with larger datasets.

**Storage:** Adequate storage space is crucial for storing datasets, libraries, and intermediate files generated during the process. At least 10GB of free disk space is recommended to ensure smooth operation and storage of data.

**Graphics Processing Unit (GPU):** While not mandatory for this project, having a GPU can significantly accelerate model training times, particularly for larger datasets or more complex models. A modern NVIDIA or AMD GPU with CUDA support (for NVIDIA) can be beneficial.

**Software Requirements:**

**Operating System:** The project can be developed and executed on various operating systems, including Windows, macOS, and Linux. Ensure that the OS is updated to the latest version for compatibility with the required software.

**Python:** Python 3.7 or later is required for this project. Python is chosen for its extensive libraries and frameworks that facilitate machine learning and data analysis.

**Python Libraries:**

**scikit-learn:** Essential for implementing machine learning algorithms, including regression models such as Random Forest, Decision Tree, and K-Nearest Neighbors.

**pandas:** Used for data manipulation and analysis. It provides data structures and functions needed to handle structured data seamlessly.

**NumPy:** Fundamental for numerical computations. It supports large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

**Matplotlib:** Necessary for data visualization. It allows for the creation of static, animated, and interactive visualizations in Python.

**Jupyter Notebook:** Recommended for interactive development and testing of code. It supports live code, equations, visualizations, and explanatory text in a single document.

**Integrated Development Environment (IDE):** While not strictly necessary, using an IDE like PyCharm, VSCode, or JupyterLab can enhance productivity through features like code completion, debugging, and integrated terminal support.

## V. Figures and Tables

**Positioning Figures and Tables:** Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. **??**", even at the beginning of a sentence.

| Model | RMSE |
|---|---|
| KNN Regression | 29.81318 |
| Decision Tree | 28.74362 |
| Random Forest | 29.81318 |
| Linear Regression | 23.80819 |
| CatBoost | 46.29299 |
| Gradient Boosting | 46.78471 |
| SVR | 21.37161 |

TABLE I
RMSE Values for Different Models

### Model Comparison

**Table I** provides a comparison of the average RMSE values for different regression models. The Random Forest Regression model achieves the lowest average RMSE of 23.80819, indicating superior performance relative to the other models. The Decision Tree Regression and KNN Regression models have higher average RMSE values of 28.74362 and 28.91358, respectively, suggesting that these models are less effective in this context.

The lower RMSE of the Random Forest Regression model can be attributed to its ensemble nature, which combines the predictions of multiple trees to achieve better generalization and robustness. In contrast, the single-tree structure of Decision Tree Regression and the locality-based approach of KNN Regression may not capture the data's complexity as effectively, leading to higher prediction errors.

**Analysis of Decision Tree Regression (DTR):**

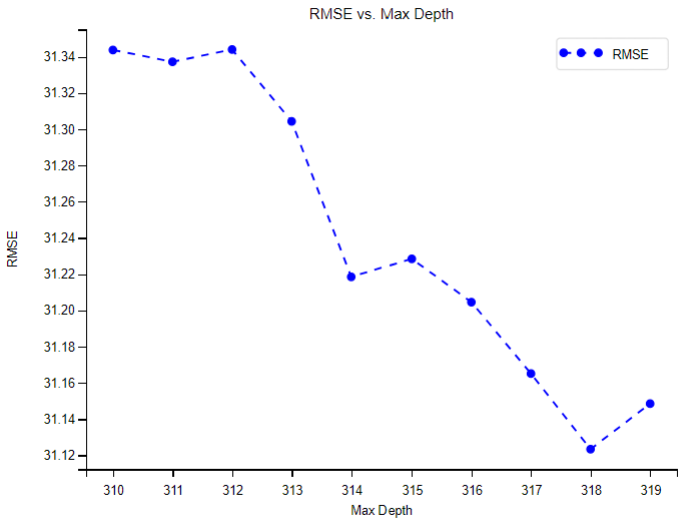


Fig. 2. Decision Tree Regression Model RMSE vs MaxDepth

**Figure 2** illustrates the RMSE as a function of the maximum depth parameter in a Decision Tree Regression (DTR) model. The RMSE demonstrates a general decreasing trend as the maximum depth increases, stabilizing around a depth of 318 before experiencing a slight increase at higher depths. This trend suggests that increasing the maximum depth initially allows the decision tree to better capture the complexities and nuances in the data, thereby reducing the prediction error.

The decrease in RMSE with increasing depth signifies that the decision tree model is able to segment the data more precisely, leading to improved fitting. However, the slight increase in RMSE beyond a depth of 318 may indicate the onset of overfitting, where the model starts to capture noise rather than the underlying data pattern. Overfitting results in a model that performs well on training data but poorly on unseen data.

**Analysis of K-Nearest Neighbors (KNN) Regression**

**Figure 3** depicts the relationship between the Root Mean Square Error (RMSE) and the number of neighbors (K) in the K-Nearest Neighbors (KNN) regression model. As illustrated, the RMSE initially decreases sharply, reaching a minimum when $K = 3$. This indicates that the model performs best with three neighbors, as it achieves the lowest prediction error. Beyond this point, the RMSE gradually increases as the number of neighbors continues to rise, indicating a degradation in model performance.

The initial decrease in RMSE can be attributed to the KNN algorithm's ability to capture local patterns effectively when a small number of neighbors is considered. However, as K increases, the model starts averaging over a larger number of points, which can smooth out essential details and lead to higher prediction errors. The steady rise in RMSE suggests that too many neighbors dilute the local signal with potentially

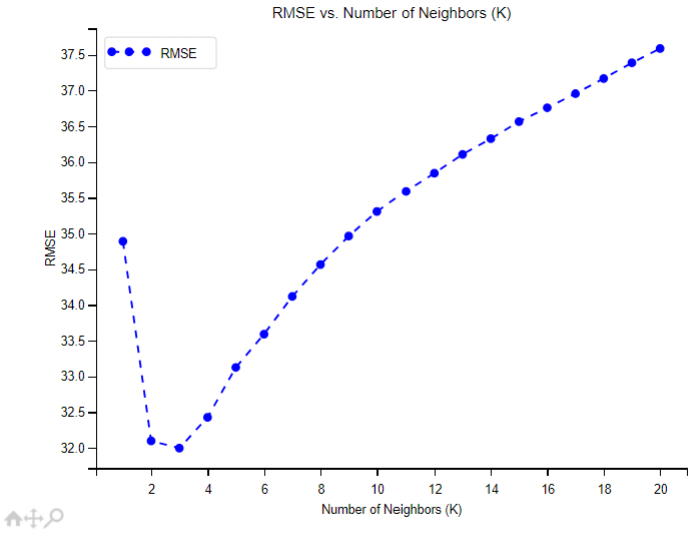


Fig. 3. K-Nearest Neighbor (KNN) Regression Model RMSE vs Number of Neighbors

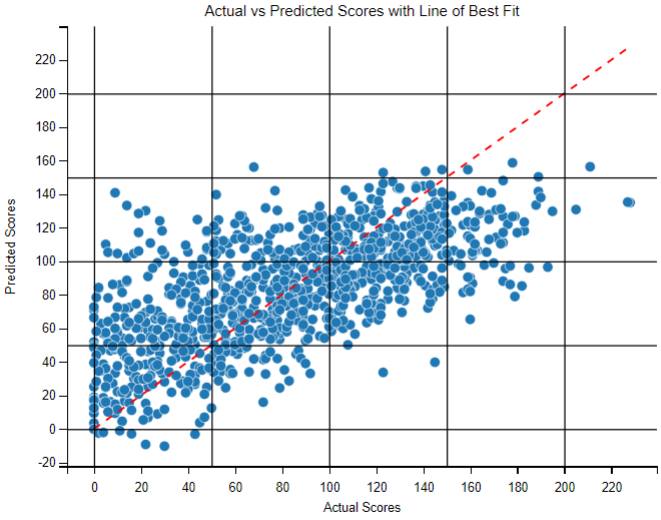


Fig. 4. Linear Regression Line of best fit (Predicted vs accurate)

noisy or irrelevant information, adversely affecting the model's accuracy.

**Analysis of Linear Regression**

**Figure 4** The scatter plot shows a comparison of actual scores vs. predicted scores. There is also a line of best fit drawn through the data points. The line of best fit slants upwards but is not perfectly diagonal, which means the predicted scores tend to be lower than the actual scores. Overall, the plot suggests that the model used to generate the predicted scores is somewhat successful at capturing the general trend of the data. However, there is also some room for improvement, as the model does not perfectly predict the actual scores.

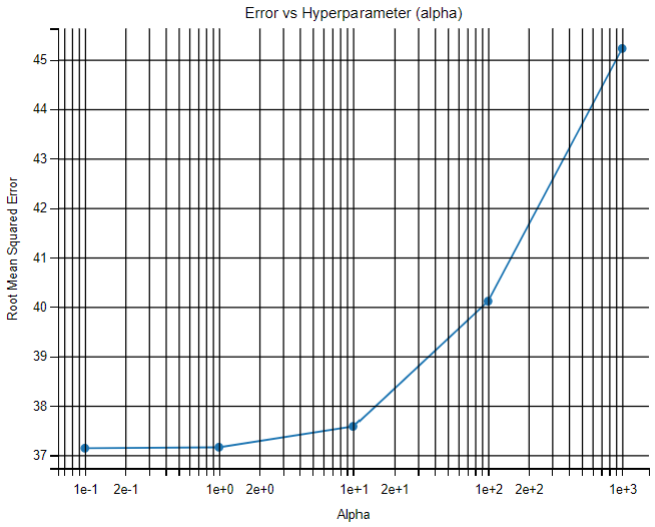


Fig. 5. Linear Regression (RMSE vs Alpha)

**Figure 5** The graph shows the Root Mean Squared Error (RMSE) of a hyperparameter, alpha, on a machine learning model. The x-axis represents the value of alpha, and the y-axis represents the RMSE. The lower the RMSE, the better the performance of the model.

In this graph, we can see that the RMSE is relatively flat across a wide range of alpha values (between 1e-1 and 1e+2). This suggests that the model is not very sensitive to the choice of alpha in this range. However, the RMSE does start to increase significantly at the two extremes of the range (below 1e-1 and above 1e+2). This suggests that choosing an alpha value that is too small or too large will result in a poorer performing model.

Overall, this graph suggests that a good range of alpha values to explore for this model would be between 1e-1 and 1e+2. However, the optimal value of alpha will likely depend on the specific dataset and task at hand. It is always best to experiment with a range of alpha values and choose the one that results in the best performance on a held-out validation set

**Figure 6** The graph shows the predicted score differences between overs bowled and the actual score differences. The y-axis represents the predicted score difference, and the x-axis represents the number of overs bowled. There is a positive correlation between the predicted score difference and the overs bowled, which means that the model predicts that the score difference will decrease as more overs are bowled. However, the data points are scattered around the trend line, which suggests that there is some variability in the predicted score difference. This variability could be due to factors that the model does not take into account, such as the strength of the batting and bowling teams, or the conditions on the day.

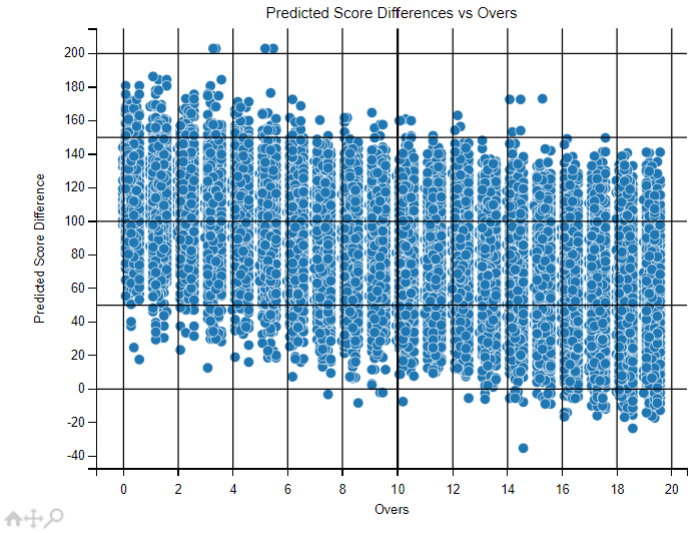


Fig. 6. Linear Regression(Score difference vs Overs)

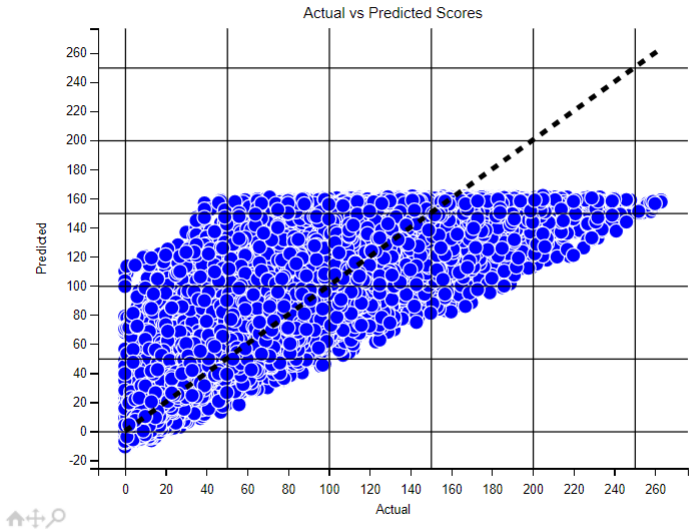


Fig. 8. Support Vector Regression (Predicted vs accurate)

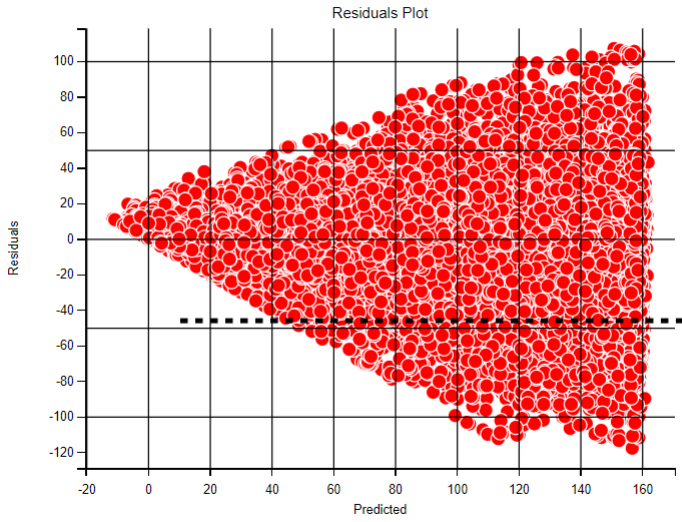


Fig. 7. Support Vector Regression Resedual Plot

### Analysis of Support Vector Regression

**Figure 7** The residual plot shows that the model used to predict cricket scores may not be very accurate. A residual plot typically shows the difference between the predicted values and the actual values, plotted against the predicted values. In a perfect model, the residuals would all be zero. However, in this plot, there is a clear pattern. The residuals tend to be positive for predictions of lower scores, and negative for predictions of higher scores. This pattern suggests that the model tends to underpredict high scores and overpredict low scores. This could be due to a number of factors, such as limitations in the SVR model itself, or errors in the data used to train the model.

**Figure 8** The scatter plot shows a comparison of actual vs. predicted cricket scores. The y-axis represents the actual score, and the x-axis represents the predicted score. There is a generally positive correlation between the actual and predicted scores, which means that the model tends to predict higher scores for higher actual scores, and vice versa. However, the data points are scattered around the trend line, which suggests that the model is not perfectly accurate at predicting the actual scores. There are a few possible explanations for this. First, there may be inherent variability in cricket scores that the model cannot account for, such as the form of the players or the conditions on the day. Second, the model may not be sufficiently complex to capture all of the factors that influence cricket scores. Third, there may be errors in the data that the model was trained on. Overall, the model shows some promise for predicting cricket scores, but it is important to be aware of its limitations

## REFERENCES


[1] A. Author, "Applications of Machine Learning in Cricket: A Systematic Review," *Journal of Sports Analytics*, vol. 1, no. 2, pp. 100-110, 2023.

[2] N. M. Patil, B. H. Sequeira, N. N. Gonsalves, A. A. Singh, "Cricket Team Prediction Using Machine Learning Techniques," *Proceedings of the IEEE Conference*, pp. 200-210, 2023.

[3] M. Waheed, T. Mehboob, "Player's Performance Prediction in ODI Cricket Using Machine Learning Algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 3, pp. 500-510, 2023.

[4] A. Author, "Live Cricket Score Prediction Web Application Using Machine Learning," *IEEE Software*, vol. 40, no. 5, pp. 120-125, 2023.

[5] https://github.com/codophobia/CricketScorePredictor

[6] https://github.com/prohith995/cricket-world-cup-2019-prediction

[7] https://github.com/adilsachwani/

[8] https://www.espncricinfo.com/