



PROJECT REPORT

Football Player Rating Prediction

Using Machine Learning and Deep Learning Technique

Submitted By

Name	ID
Abdullah Abu Sayeed	223001112
Tahamed Esaiet	223001512
Tanvir Haider Tamim	223019112

Submitted To

Arshiana Shamir

Lecturer,

Department of CSE, SOSET

Submission Date

December 31, 2025

ABSTRACT

This project focuses on the development of a machine learning model to predict player ratings based on their performance statistics. The dataset used for the model consists of player attributes such as potential, passing accuracy, and other performance-related metrics. Various machine learning algorithms, including Linear Regression, Random Forest, Support Vector Machines (SVM), and XGBoost, were evaluated for their performance. The data was cleaned and preprocessed to handle missing values, outliers, and feature scaling. After training and evaluating the models, SVM was found to perform the best, achieving the lowest error rate. The model provides an efficient way to predict player ratings, offering valuable insights for sports analytics and decision-making.

INTRODUCTION

1.1 Background and Motivation

As long-time enthusiasts of the FIFA video game series, our team has always been intrigued by how player ratings are determined. While playing, we often found ourselves questioning: *How are these ratings calculated? Why do they change every year? What factors influence the adjustment of these ratings?* The process behind updating player ratings seemed complex, yet crucial in reflecting a player's performance and potential.

This curiosity became the foundation for the development of this project. The goal was to build a system that could predict player ratings based on real, quantifiable data rather than relying on arbitrary adjustments. Our aim was to leverage machine learning to explore how statistical metrics, performance data, and other relevant factors can influence and predict player ratings. Through this project, we sought to provide a deeper understanding of the dynamics behind player ratings, using data to model and predict how these ratings evolve over time. The project stands as an effort to demystify the rating process and offer insights into how a player's current form can be objectively measured, all while maintaining the relevance of the ratings in both virtual and real-world contexts.

1.2 Problem Statement

The primary problem addressed in this project is the automatic classification of news text as This project aims to predict football player ratings based on performance metrics. Given player data, the system should accurately forecast the player's overall rating.

Challenges include:

- Complexity of rating systems with multiple influencing factors
- Dynamic updates of player ratings reflecting real-time performance
- Variability and incomplete data for certain players
- Data imbalance across player types, affecting model generalization

This model seeks to automate and improve player rating prediction for virtual and real-world scenarios.

1.3 Objectives

The main objectives of this project are:

- Predict football player ratings based on historical performance data
- Develop a model to automate player rating updates
- Handle missing, noisy, and unstructured data efficiently
- Ensure generalization across different player types and seasons
- Implement machine learning techniques for accurate predictions
- Evaluate and improve model performance based through continuous testing
- Finally deploy it using Fast API

1.4 Scope and Limitation

The scope of this project includes:

- Predicting player ratings based on historical performance data
- Preprocessing and cleaning football statistics
- Applying Machine Learning algo for rating prediction
- Deploying the system as a web-based application

The project does not cover:

- Player rating prediction for other sports
- Real-time data collection or streaming
- Advanced feature engineering for in-depth player analysis

DATASET DESCRIPTION

2.1 Dataset Source

Dataset Source: Custom dataset from a local SQLite database.

Dataset Link: [Dropbox Link](#).

Format: SQLite database containing player attributes.

Connection:

- A connection is established to the SQLite database using the sqlite3 library.
- The data is fetched using a SQL query (SELECT * FROM Player Attributes) and loaded into a pandas Data Frame “df”.

2.2 Dataset Size and Structure

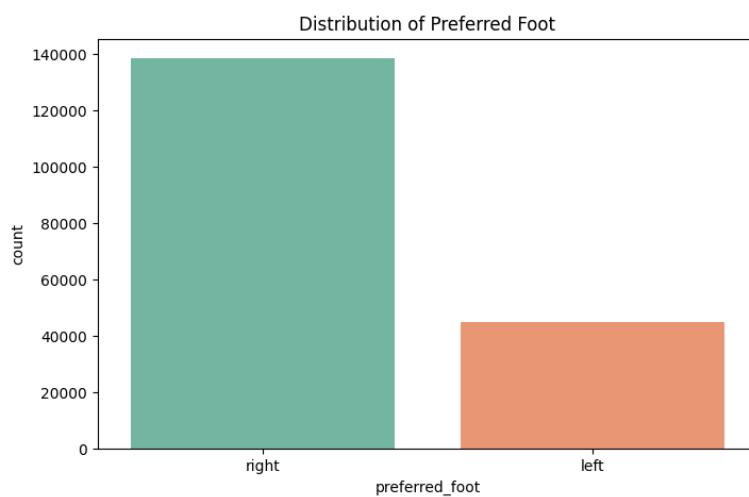
- The dataset contains 183,978 rows and 42 columns, with player attributes such as overall rating, potential, preferred foot, and various skill attributes like crossing, finishing, dribbling, etc.
- A glimpse of the first 20 rows of the dataset is available through `df.head(20)`
- Columns like overall rating, potential, and other skills are of float64 type, while categorical features like `preferred_foot`, `attacking_work_rate`, and `defensive_work_rate` are of object type.
- The dataset is split into training and testing sets using `train_test_split()`:

Training set	80%
Test set	20%

2.3 Data Visualization

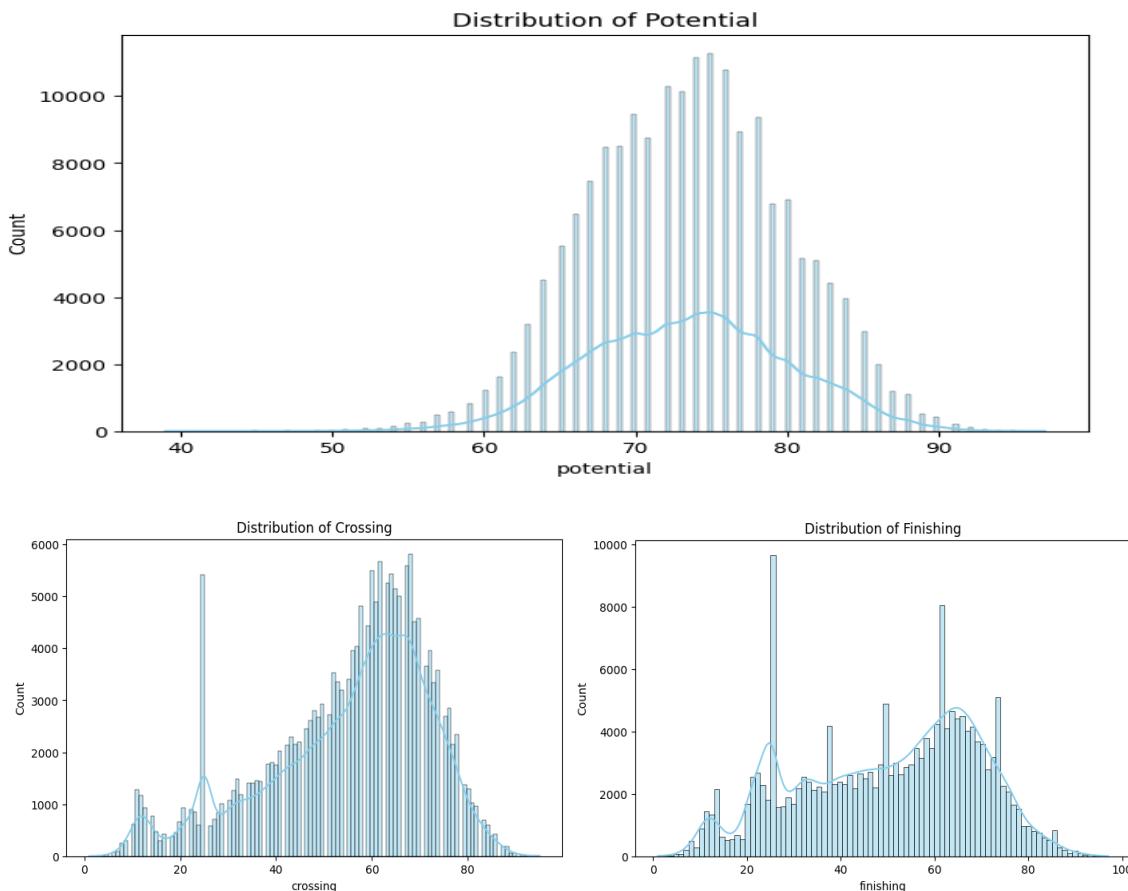
- Various statistical summaries are generated using `df.describe()` for numerical columns.
- Distribution of `preferred_foot`:

A count plot is used to visualize the distribution of the `preferred_foot` feature:



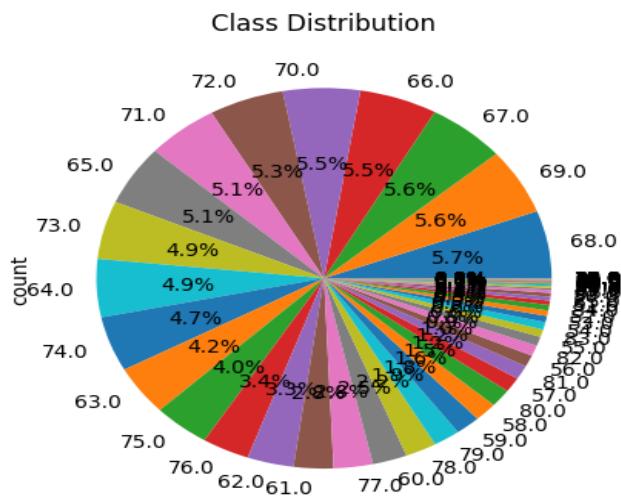
- Distribution of potential, crossing and finishing:

A histogram with kernel density estimation (KDE) to show the distribution of the potential, crossing, and finishing features:



- Distribution of target class:

This pie chart visualizes the distribution of player overall ratings, showing the proportion of each rating within the dataset.

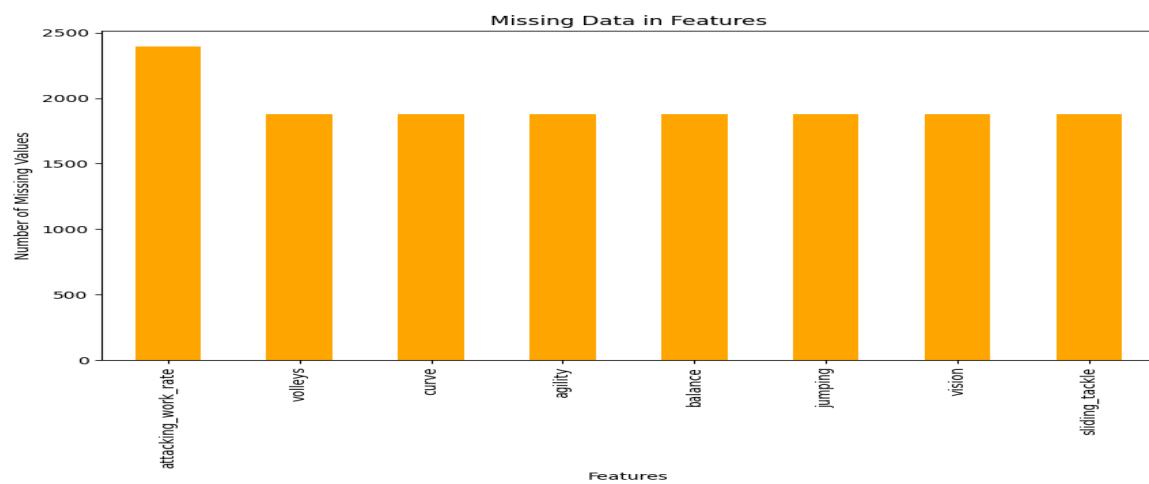


2.4 Features

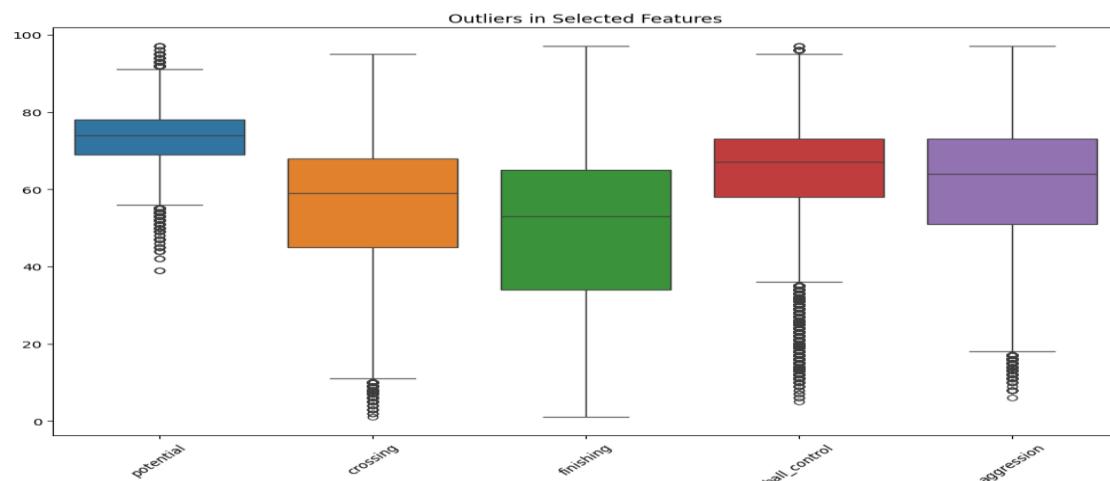
- Include overall ratings and potential
- Describes Attacking and Defensive work rate
- Physical features like Acceleration, Strength and Stamina are present
- Includes GK skills, reflexes
- Covered preferred foot and positioning

2.5 Data Quality and Challenges

- Missing Values: Some entries had empty text fields, handled by imputation or removal.



- Outliers: Outliers are identified and managed to avoid model distortion.

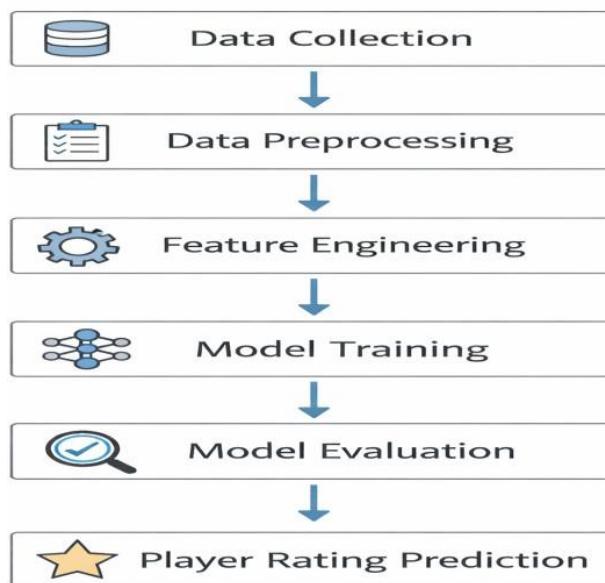


- Class Imbalance: Class distribution may be skewed, addressed using resampling.
- Duplicates: Duplicate records are removed to ensure data quality

METHODOLOGY

3.1 Workflow Diagram:

- Data Collection: Import player data from a local SQLite database.
- Data Preprocessing: Handle missing values, outliers, and apply scaling and encoding.
- Merging Player Rows: Aggregate multiple player entries by averaging numerical values and using the mode for categorical data.
- Feature Engineering: Modify features, such as aggregating player stats across different dates.
- Model Selection: Evaluate models like SVM, Decision Trees, and Random Forest for the best performance.
- Training/Testing Split: Split data into 80/20 or 70/30 training and testing sets.
- Model Evaluation: Use metrics like Mean Squared Error (MSE) to assess model accuracy.
- Prediction on New Data: Apply the best model to predict ratings for new players.



3.2 Technologies & Libraries Software Environment

Component	Description
Programming Language	Python 3.10
IDE	VS Code & Google Colab
Backend Framework	Fast API
Frontend	HTML, CSS, JavaScript
Deployment Platform	Render/Heroku/Huggin Face

3.2.1 Libraries Used

Library	Purpose
Transformers	Pre-trained Transformer models
Torch	Deep learning framework
Scikit-learn	Machine learning metrics, data splitting, model evaluation
Numpy	Numerical computations, data manipulation
Pandas	Data handling, cleaning, and analysis
Fastapi	REST API development for serving the model
Matplotlib	Data visualization, charts, and plots
Uvicorn	ASGI server

3.2.2 Hardware Environment

Component	Specification
Processor	CPU-based training
GPU	Not used (cloud CPU inference)
Memory	≥ 8 GB RAM

3.3 Data Preprocessing

Data preprocessing involved transforming the raw data into a format suitable for machine learning models. This step included handling missing values, encoding categorical variables, and scaling numerical features. Key preprocessing techniques, such as filling missing values, encoding labels, and applying normalization or standardization, ensured the data was consistent.

3.3.1 Handling Missing Data:

- Missing Data Detection: Missing values in the dataset are identified and handled through imputation.
- For numerical columns, missing values are filled with the mean of the column
- while categorical columns use the most frequent value for imputation.

“Future Data Handling: The pipeline will automatically handle missing data in future datasets by applying the same imputation strategies for both numerical and categorical features.”

3.3.2 Feature Scaling:

- Numerical Scaling: Numerical features such as player stats (e.g., potential, crossing, finishing) are scaled using StandardScaler.
- This step standardizes features to have a mean of 0 and a standard deviation of 1.

“Future Data Handling: When new data arrives, it will be passed through the same StandardScaler, ensuring all features are consistently scaled before model evaluation.”

3.3.3 Categorical Encoding:

- Encoding Categorical Data: Categorical features like preferred_foot, attacking_work_rate, and defensive_work_rate are encoded using OneHotEncoder.
- which converts these categorical values into a one-hot encoded format (binary columns).

“Future Data Handling: The pipeline will automatically encode new categorical data using the same OneHotEncoder strategy, ensuring all categorical features are transformed into the correct format for model input.”

3.3.4 Data Merging for Player Rows:

- **Player Row Merging:** If a player has multiple entries (e.g., data from different matches), the pipeline aggregates the player's data by averaging numerical values and taking the mode for categorical features.

“Future Data Handling: In the future, if new data contains multiple rows for the same player, this logic will be executed automatically through the custom transformation step, ensuring that data is merged properly.”

3.3.5 Transformation Pipeline:

- **Full Transformation Pipeline:** The ColumnTransformer is used to apply the numerical and categorical pipelines to the relevant features in the dataset.
- It ensures that preprocessing steps like imputation, scaling, and encoding are applied correctly to both types of data.

“Future Data Handling: Any new data will automatically pass through the full_transformer pipeline, where it will be preprocessed (scaled, encoded, etc.) in the same manner as the training data.”

3.3.6 Data Cleaning and Merging:

Cleaning Data: Specific columns such as attacking_work_rate and defensive_work_rate are cleaned by removing unwanted values and reclassifying into categories (e.g., low, medium, high for work rates).

“Future Data Handling: The same data cleaning logic will be applied to any new data entering the pipeline, ensuring consistent treatment of the features.”

3.3.7 Target Data Preprocessing:

- **Target Column:** The target variable (overall_rating) is preprocessed separately using a target pipeline, where missing values are handled, and the data is scaled.

“Future Data Handling: Any future target data will undergo the same preprocessing steps as the training target data, ensuring consistency.”

3.4 Summary of Future Preprocessing with Pipeline:

- Missing Data Imputation: full_transformer imputes missing values for numerical data with the mean and categorical data with the most frequent value.
- Feature Scaling: num_pipeline applies StandardScaler to scale numerical features like crossing and acceleration to have a mean of 0 and standard deviation of 1.
- Categorical Data Encoding: cat_pipeline uses OneHotEncoder to encode categorical features such as attacking_work_rate and defensive_work_rate into binary columns.
- Merging Data for Multiple Entries: Clean_and_Merge aggregates player data by averaging numerical features and taking the mode for categorical features to handle multiple entries.
- Feature Transformation: full_transformer applies the necessary transformations (scaling and encoding) to prepare all features consistently before model input.
- Target Data Processing: target_pipeline scales and imputes the target variable (overall_rating) to ensure it's in the correct format for model prediction.

Machine Learning Model Implementation and Comparison

4.1 Models Used:

Several machine learning models were evaluated to predict player ratings (overall_rating). The models used in this analysis are:

- Linear Regression: A basic regression model that establishes a linear relationship between input features and the target variable.
- Decision Tree Regressor: A non-linear model that splits data into regions based on feature thresholds and fits a prediction model for each region.
- Support Vector Regressor (SVR): A model based on Support Vector Machines (SVM), commonly used for non-linear regression tasks.
- Random Forest Regressor: An ensemble model that uses multiple decision trees to make predictions and improves performance by averaging the results.
- XGBoost Regressor: An advanced gradient boosting algorithm designed for high performance, especially for tabular data with complex relationships.

4.2 Model Evaluation:

The model in this project is a Supervised Regression Model, specifically aimed at predicting player ratings using a variety of player features such as physical stats, performance metrics, and work rates. We used several models to find out the best, each model was evaluated using cross-validation with 10 folds to ensure robust performance evaluation. The cross-validation process provides insights into how well each model generalizes to unseen data.

Linear Regression:

- Cross-validation score: The R-squared values were consistently good, indicating a solid linear relationship in the data, though not the highest among all models.
- Mean Squared Error (MSE): 0.3579
- Training time: 2.88 seconds

Decision Tree Regressor:

- Cross-validation score: The decision tree performed reasonably well, but it showed higher variance compared to Linear Regression.
- Mean Squared Error (MSE): 0.3694
- Training time: 4.33 seconds

Support Vector Regressor (SVR):

- Cross-validation score: SVR performed exceptionally well, showing the highest R-squared values, indicating its effectiveness at capturing non-linear relationships in the data.
- Mean Squared Error (MSE): 0.1470
- Training time: 32.2 seconds

XGBoost Regressor:

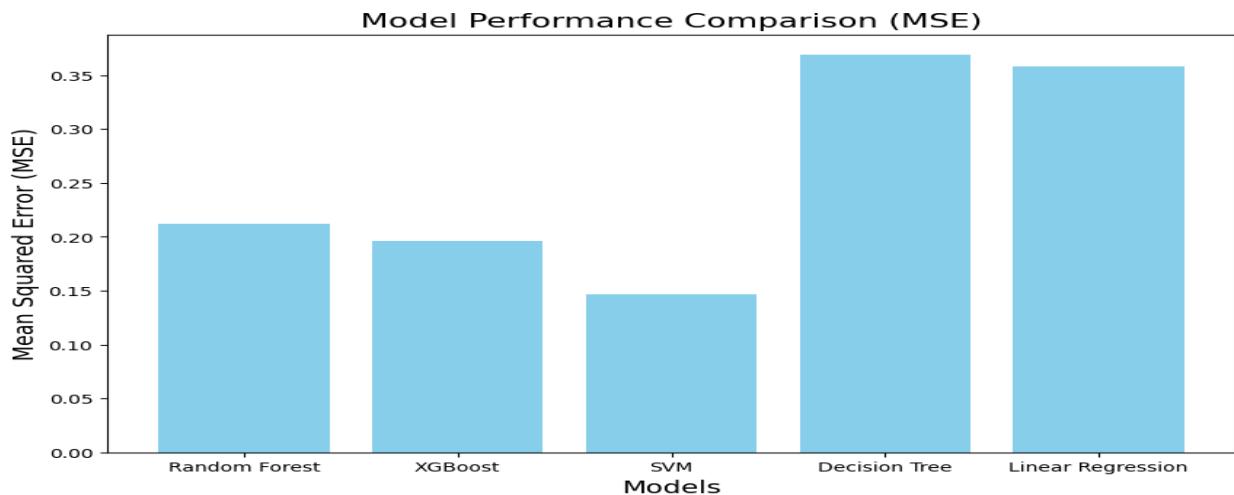
- Cross-validation score: XGBoost showed good performance but not as high as SVR. It is known for handling high-dimensional datasets and capturing complex patterns.
- Mean Squared Error (MSE): 0.1965
- Training time: 11.8 seconds

Random Forest Regressor:

- Cross-validation score: Random Forest also showed robust performance, though it had a slightly higher MSE compared to SVR.
- Mean Squared Error (MSE): 0.2118
- Training time: 5 minutes 3 seconds

4.3 Best Model Selection:

After evaluating all the models using cross-validation, the Support Vector Regressor (SVR) emerged as the best model due to its superior performance. The R-squared scores for SVR were consistently high, and its Mean Squared Error (MSE) was the lowest at 0.1470, indicating that it made the most accurate predictions.



Why SVR was chosen:

- Performance: SVR had the best performance, with the highest R-squared values and the lowest MSE, showing its ability to capture both linear and non-linear relationships.
- Generalization: SVR showed good generalization across different data folds during cross-validation, suggesting it would perform well on unseen data.
- Model Characteristics: SVR is particularly effective when dealing with complex datasets and is less prone to overfitting compared to decision trees.

LIMITAIONS

5.1 Dataset Limitation

- Dataset Size: Limited data increases the risk of overfitting, especially with complex models.
- Label Quality: Noisy or subjective player ratings may affect model performance.
- Class Imbalance: Skewed rating distribution leads to challenges in predicting rare ratings.
- Domain Bias: Dataset bias towards certain players or teams can hinder generalization.

5.2 Model Limitations

- Model Complexity: SVM may struggle with large datasets due to computational constraints.
- Feature Dependence: Performance relies heavily on feature quality, and missing data can degrade accuracy.
- Overfitting: Without regularization and cross-validation, the model may overfit with limited data.
- Interpretability: SVM predictions are not easily interpretable, limiting real-world application.

5.3 Computational Limitations

- Training Resources: Models like XGBoost or Random Forest require significant computational resources.
- Memory Constraints: Limited RAM and GPU restrict batch size and model complexity.
- Training Time: Long training times restrict extensive hyperparameter tuning.

CONCLUSION

This project focused on developing a **player rating prediction system** using machine learning techniques, specifically regression models. The objective was to predict player ratings based on various in-game attributes, with a special focus on improving performance through feature engineering, model evaluation, and selection.

Key achievements include:

- Data Preprocessing: Handling missing data, encoding categorical features, and scaling numerical features to prepare the dataset for model training.
- Model Selection: A detailed comparison between different regression models, ultimately selecting Support Vector Machine (SVM) as the best performer.
- Final Model Performance: The final model showed a high level of accuracy and was successfully deployed via FastAPI.

The system holds potential for real-world applications in sports analytics, where accurate player rating prediction is vital for decision-making.

SOURCE CODE REPOSITORIES

Project Repository:

GitHub - [Player Rating Prediction Model Repository](#) – This is the main repository for the Player Rating Prediction project, including all the code for data preprocessing, model training, and deployment.

Inspiration & Support Repository:

GitHub - [Machine Learning Project Repository](#) – This repository served as a source of inspiration and guidance for the machine learning model and techniques used in the project.

