**PROJECT DOCUMENTATION**

**MACHINE LEARNING**

|  |  |
| --- | --- |
| **NAME** | **DIVYA PRASATH R S** |
| **COURSE** | **DA/DS** |
| **BATCH** | **MAY 2025** |

**REINFORCEMENT PROJECT**

**1.Problem Statement :**

The aim of this project is to predict the winner of an IPL match based on various features such as the teams playing, toss winner, toss decision, and other match-related factors using different machine learning models like Logistic Regression, SVM, KNN, Decision Trees, Random Forest, and XGBoost. The best model must be selected after evaluating the performance and hyperparameters must be tuned for further improvements.

**2. Data Understanding**

**Dataset Description**

The dataset used in this project is the **IPL Matches Dataset**, which contains information about Indian Premier League (IPL) cricket matches. Each row represents a single match, and the columns capture details about the match, such as the season, venue, participating teams, toss decision, match outcome, and performance highlights.

The dataset consists of **756 rows** (matches) and **18 columns** (features).

**Column Description**

1. **id** – Unique match identifier.
2. **Season** – The IPL season in which the match was played.
3. **city** – City where the match took place.
4. **date** – Date of the match.
5. **team1** – First competing team.
6. **team2** – Second competing team.
7. **toss\_winner** – Team that won the toss.
8. **toss\_decision** – Toss decision (bat/field).
9. **result** – Match result type (normal, tie, no result).
10. **dl\_applied** – Whether the Duckworth-Lewis method was applied (1 = Yes, 0 = No).
11. **winner** – The team that won the match.
12. **win\_by\_runs** – Margin of victory if the winning team defended a total (in runs).
13. **win\_by\_wickets** – Margin of victory if the winning team chased (in wickets).
14. **player\_of\_match** – Player adjudged as best performer of the match.
15. **venue** – Stadium where the match was held.
16. **umpire1** – On-field umpire 1.
17. **umpire2** – On-field umpire 2.
18. **umpire3** – Third umpire (only available for some matches).

**3. Data Cleaning**

**Steps Taken for Cleaning the Data**

1. **Handling Missing Values**
   * Columns like **city** and **umpire3** contained missing values.
   * Missing **city** values were filled using the venue information.
   * The column **umpire3** was dropped as it had many missing values and was not relevant for prediction.
   * Verified that other key columns (such as winner, team1, team2, toss\_winner) did not have missing values.
2. **Removing Irrelevant Columns**
   * Columns such as **id** and **date** were dropped because they are identifiers and do not contribute to prediction.
   * Columns like **umpire1** and **umpire2** were removed as they don’t influence match outcomes.
3. **Encoding Categorical Features**
   * Features like **team1**, **team2**, **winner**, **toss\_winner**, **venue**, and **city** are categorical.
   * These were converted into numerical format using **One-Hot Encoding** to make them suitable for machine learning models.
4. **Standardizing Numerical Features**
   * Columns such as **win\_by\_runs** and **win\_by\_wickets** were retained as numeric values.
   * Feature scaling (StandardScaler) was applied to ensure fair treatment of numerical values across models like Logistic Regression, KNN, and SVM.
5. **Checking for Duplicates**
   * Verified that no duplicate match records existed in the dataset.
6. **Balancing Target Classes**
   * The **winner** column (target variable) was checked for imbalance across teams.
   * Stratified train-test split was applied to preserve class distribution.

**4.Exploratory Data Analysis (EDA)**

**Key Visualizations**

1. **Match Distribution by Season**
   * **Chart**: Bar chart showing the number of matches played each season.
   * **Insight**: Helps identify growth of IPL across seasons.
2. **Top Winning Teams**
   * **Chart**: Horizontal bar chart of total matches won by each team.
   * **Insight**: Highlights consistently successful franchises (e.g., MI, CSK).
3. **Toss Decision Analysis**
   * **Chart**: Pie chart showing the proportion of “Bat” vs “Field” after winning the toss.
   * **Insight**: Captures captain preferences and strategies across seasons.
4. **Toss Winner vs Match Winner**
   * **Chart**: Bar chart comparing how often toss-winning teams also won the match.
   * **Insight**: Shows whether toss has a significant impact on match outcome.
5. **Win Margin Distribution**
   * **Chart**:
     + Histogram for win\_by\_runs.
     + Histogram for win\_by\_wickets.
   * **Insight**: Reveals common winning margins (close matches vs big wins).
6. **Player of the Match Awards**
   * **Chart**: Bar chart of top players with the most “Player of the Match” awards.
   * **Insight**: Identifies consistent match-winners in IPL history.
7. **Venues with Most Matches**
   * **Chart**: Bar chart of matches played per venue.
   * **Insight**: Highlights the most frequently used stadiums.
8. **Season-wise Winning Trends**
   * **Chart**: Line chart of win counts for top 4 teams across seasons.
   * **Insight**: Shows dominance patterns (e.g., MI in 2010s, CSK consistency).

**Insights Drawn**

* The number of matches **increased steadily** over the first few seasons and stabilized after IPL expanded.
* Certain teams like **Mumbai Indians (MI) and Chennai Super Kings (CSK)** dominate the win count, showcasing consistent performance.
* **Fielding first (chasing)** is the more popular toss decision, especially in later seasons, aligning with the “chasing advantage” in T20 cricket.
* The toss does **not guarantee victory**, but winning the toss increases the probability of winning slightly.
* Most matches are decided by **small margins** (few runs or wickets), showing how competitive IPL games are.
* A few star players (e.g., Virat Kohli, AB de Villiers, MS Dhoni, Rohit Sharma) consistently appear in “Player of the Match” charts.
* Stadiums like **Wankhede (Mumbai)**, **Eden Gardens (Kolkata)**, and **M. Chinnaswamy (Bangalore)** host the maximum number of games.

**5.Models Trained**

We trained and tested the following models:

* Logistic Regression
* Support Vector Machine (SVM)
* K-Nearest Neighbours (KNN)
* Decision Tree Classifier
* Random Forest Classifier
* AdaBoost Classifier
* Gradient Boosting Classifier
* XGBoost Classifier

**Logistic Regression :**

* **Logistic Regression** is a **supervised machine learning algorithm** used for **classification problems**.
* It predicts the probability of an outcome that can only have **two possible values** (binary classification), though it can be extended to **multi-class classification**.
* Unlike **Linear Regression**, which predicts continuous values, Logistic Regression predicts **discrete categories** (e.g., Win/Loss, Yes/No, Spam/Not Spam).

**Support Vector Machine (SVM)**

* **Support Vector Machine (SVM)** is a **supervised machine learning algorithm** used for both **classification** and **regression** tasks (mostly classification).
* It works by finding the **best decision boundary (hyperplane)** that separates data points of different classes with the **maximum margin**.

**K-Nearest Neighbours (KNN)**

* **K-Nearest Neighbours (KNN)** is a **supervised machine learning algorithm** used for **classification** and **regression**.
* It is a **non-parametric, instance-based learning method** – meaning it makes predictions based on the training data itself (lazy learner) instead of learning a fixed model.

**Decision Tree Classifier**

* A **Decision Tree Classifier** is a **supervised machine learning algorithm** used for both **classification** and **regression tasks**.
* It works by **splitting the dataset into smaller subsets** based on feature values, forming a **tree-like structure of decisions**.
* Each **internal node** represents a feature (attribute), each **branch** represents a decision (rule), and each **leaf node** represents the outcome (class label).

**Random Forest Classifier**

* **Random Forest Classifier** is an **ensemble learning algorithm** that builds multiple **decision trees** and combines their outputs to improve accuracy and reduce overfitting.
* It belongs to the family of **bagging (Bootstrap Aggregating) methods**.

**AdaBoost Classifier**

* **AdaBoost (Adaptive Boosting)** is an **ensemble learning method** that combines multiple **weak learners** (usually shallow decision trees, also called *decision stumps*) into a single **strong classifier**.
* It works by giving **higher weights to misclassified samples** so that the next weak learner focuses more on the difficult cases.

**XGBoost Classifier**

* **XGBoost (Extreme Gradient Boosting)** is an advanced implementation of the Gradient Boosting algorithm, optimized for **speed and performance**.
* It uses **parallelized tree boosting**, **regularization**, and other optimization techniques to provide state-of-the-art accuracy for classification and regression tasks.

**6. Model Evaluation**

* Evaluation metrics used:
  + **Accuracy** → overall correctness
  + **Precision** → positive predictive value
  + **Recall** → sensitivity (true positive rate)
  + **F1-Score** → harmonic mean of precision & recall
  + **Confusion Matrix** → misclassification visualization

**Hyperparameter Tuning**

* Used **GridSearchCV** and **RandomizedSearchCV**.
* Example tuning:
  + Logistic Regression: C, solver
  + SVM: C, kernel, gamma
  + KNN: n\_neighbors, weights
  + Decision Tree: max\_depth, criterion
  + Random Forest: n\_estimators, max\_depth, min\_samples\_split
  + XGBoost: learning\_rate, max\_depth, n\_estimators

**7. Model Comparison**

**Best-Performing Model**

* The **XGBoost Classifier** achieved the **highest Accuracy (0.97) and F1 Score (0.96)** among all models.
* It balances both **bias and variance** effectively due to gradient boosting and ensemble learning.
* Other models like Gradient Boosting and Random Forest also performed well, but XGBoost slightly outperformed them.

**Justification:**

* Chosen as the final model because it consistently provides the best balance of Accuracy, Precision, Recall, and F1 Score.
* Saved the final tuned XGBoost model as **final\_model.pkl** for deployment.

8.MODEL COMPARISON

A screenshot of a graph

AI-generated content may be incorrect.

10.CONCLUSION

**1.Summary of the Process:**

1. The dataset was first **cleaned and processed**, including handling missing values, encoding categorical variables, and splitting into features (X) and target (y).

2. Multiple **machine learning models** were trained: Logistic Regression, KNN, SVM, Decision Tree, Random Forest, and XGBoost.

**3. Hyperparameter tuning** was performed to optimize model performance.

4. Models were evaluated using metrics such as **Accuracy, Precision, Recall, F1-Score**, and **Confusion Matrix**.

**2.Insights:**

1. Data preprocessing and feature transformation significantly impacted model accuracy.

2. Ensemble models like Random Forest and XGBoost often provided higher accuracy and robustness.

3. Simpler models (Logistic Regression, Decision Tree) offered better interpretability with slightly lower accuracy.

**3.Future Work / Improvements:**

1. Incorporate more features or external data to improve predictions.

2. Explore other advanced models or deep learning techniques.

3. Perform further tuning and feature engineering to enhance accuracy.