IPL

MATCH WINNER PREDICTION-

MACHINE LEARNING

INTRODUCTION:

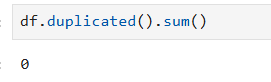
The Indian Premier League (IPL) is one of the most popular and competitive cricket tournaments in the world. This project focuses on leveraging a dataset containing detailed information about IPL matches to predict the winner of a cricket match based on various factors such as the teams playing, the toss winner, toss decision, and other match-related elements.

The primary objective of this project is to develop a predictive model that can accurately forecast the winner of an IPL match. By using machine learning algorithms like Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Trees, Random Forest, and XGBoost, the project aims to identify the best-performing model for this task. Additionally, hyperparameter tuning will be applied to further improve the selected model's performance.

Throughout the project, several steps will be followed, including data loading, cleaning, and preprocessing, as well as conducting exploratory data analysis (EDA) to uncover patterns within the dataset. The final model will be evaluated using performance metrics such as accuracy, precision, recall, and F1-score, and the best model will be saved for future use. Ultimately, the goal is to create a robust, efficient machine learning model capable of predicting IPL match outcomes with high accuracy, providing valuable insights for fans, analysts, and teams alike.

DATA CLEANING:

* Dataset containing 756 rows and 18 columns
* No duplicate values in the given data



* Checking for null values in the dataset, the umpire3 column contains 84% null values, city contains 0.93%, and winner and player\_of\_match both contain 0.53% null values. The umpire1 and umpire2 columns contain 0.23% null values. Since umpire3 contains the most null values, the entire column is dropped. The missing values in the city, player\_of\_match, umpire1, and umpire2 columns are filled with the value ‘Unknown’, and rows with null values in the Winner column are dropped.
* The result column contains two values: 'normal' and 'tie'. One-hot encoding is used to map 'normal' to 1 and 'tie' to 0, and the result column is then dropped as it is no longer needed.
* In the season column, the values are in object format, so the year is extracted and stored in a new year column. The season column is then dropped.
* Replacing ‘Delhi Capitals’ with ‘Delhi Daredevils’ and ‘Deccan Chargers’ with ‘Sunrisers Hyderabad’ because those team names were updated
* Finally, irrelevant columns like id, venue, and umpire2, which contain a high number of unique values, are also dropped.
* The toss\_decision column contains only two values: 'field' and 'bat'. Binary encoding is applied, mapping 'field' to 0 and 'bat' to 1.
* We have to convert object datatype into numerical because predictions are works well with numerical datatype. Hence with the help of label encoding technique, the fields such as 'city', 'team1', 'team2', 'toss\_winner', 'toss\_decision', 'result', 'winner', player\_of\_mactch, 'umpire1', 'SeasonYear'.
* The outliers are detected using ‘win\_by\_run’ and those rows were removed because there is a huge difference of bound values from minimum and maximum.

EXPLORATORY DATA ANALYSIS:

1. Number of Each IPL matches won by each team (Bar Graph):

This plot helps to easily compare the number of wins across different teams in the IPL. It provides a quick visual summary of which teams have been the most successful in terms of match victories. The horizontal bar format makes it easy to read and compare the values, especially if there are many teams involved. The number of colors corresponds to the number of unique teams in the winner column. From this visualization, Mumbai Indians have the highest number of wins, followed by Kolkata Knight Riders and Chennai Super Kings.

1. Decision taken by the toss winning team (Bar Graph):

This chart represents the number of times a team has chosen to field or bat after winning the toss. The insights gathered here is most of the teams prefer fielding over batting after winning the toss.

1. Proportion of matches in each season (Pie Chart):

This chart shows the proportion of IPL matches played in each season. The seasons are from the year between 2008 and 2019. The proportion is fairly distributed, with some years having slightly more matches. In 2012 and 2013, most of the teams played.

1. Distribution of wickets taken (Histogram):

This histogram represents the distribution of the number of wickets taken per match. Most matches have a lower number of wickets taken, with a few matches having higher wickets.

1. Number of IPL matches per season

This chart visualizes that the number of each season from 2008 to 2019. The number of matches increased over the years peaking around 2012 and 2013. A decline is observed in later years, stabilizing around 60 matches per season.

1. Top 10 players (Line Chart):

This graph shows the top 10 IPL players with the highest number of “Player of the Match” awards. The insight gathered here is Virat Kohli leads the chart, followed by Suresh Raina and Gautam Gambhir. The number of awards decreases gradually from left to right. Players like AB de Villiers, Chris Gayle and Rohit Sharma are also among the top performers.

1. Correlation (Heatmap):

This heatmap displays the correlation between different numerical different numerical variables in the IPL dataset. Strong negative correlation (-0.67) between win\_by\_runs and win\_by\_wickets indicating that teams winning by large number of runs are less likely to win by wickets. And dl\_applied(Duckworth-Lewis method applied) has minimal correlation with other variables. Season year does not strongly influence match-winning margins.

1. Winner by Toss Decision (Stacked bar chart):

This chart shows the winner rate based on the toss decision(bat or field) for different IPL teams. Most teams tend to win more often when choosing to field first compared to batting. Teams like Mumbai Indians and Kolkata Knight Riders have a higher win percentage option to field. Some teams show a balanced winning rate regardless of the toss decision.

DATA PREPROCESSING:

#### 1. **Converting Categorical Data to Numerical Using Label Encoding:**

We need to convert variables with text data into numerical values. This is crucial because regression models work effectively with numerical data, and many machine learning algorithms cannot handle categorical data directly. Label encoding is a method that assigns a unique integer to each category in a variable. The LabelEncoder class from sklearn.preprocessing is used to convert categorical data into numerical data by assigning a unique integer to each category.

2. **Binary Mapping for Toss Decision:**

The toss\_decision column has categorical values "bat" or "field", which need to be converted into binary values. This is useful because regression models work more efficiently with numerical inputs. This method applies the binary\_map function to each value in the toss\_decision column. As a result, "bat" becomes 1 and "field" becomes 0.

3. **Defining Features (x) and Target (y) Variables:**

The feature set (x) and the target variable (y) for training a machine learning model where the features are the independent variables used to predict the target, which is the dependent variable. The feature set (x) consists of the columns used as input variables for the model. These columns are related to the teams, toss results, match outcomes, etc. The target variable (y) is the winner column, which represents the team that won the match. This step prepares the data by separating the input features (x) and the target variable (y) that the model will learn to predict.

#### 4. **Exploring Correlations Between Features and Target Variable:**

Correlation analysis allows us to identify which features are strongly related to the target, which is crucial for feature selection and model improvement. The result is a correlation matrix that shows the strength and direction of the relationships between features. Positive correlations suggest that as the feature value increases, the likelihood of the target being a certain value also increases. Negative correlations indicate the opposite. Identifying features with strong correlations with the target helps improve model performance.

5. Standard Scaling of Features:

Feature scaling ensures that all features contribute equally to the model, preventing any feature with a larger range from dominating the model's behavior. Standardization (also known as Z-score normalization) is one of the most common methods for scaling numerical features. The StandardScaler from scikit-learn is used to standardize features by removing the mean and scaling them to unit variance. After the training set has been standardized, it is essential to **transform the test data using the same scaling parameters** that were learned from the training data.

MODEL TRAINING:

* LOGISTIC REGRESSION:

The Logistic Regression model is underperforming, with an accuracy of just 23.24%, indicating it correctly classifies only a small portion of the test data. The precision and recall scores, both at 21% and 23.24%, respectively, suggest that the model struggles to identify positive instances and frequently misclassifies negative instances as positive. The F1 score of 19.96% further highlights the imbalance between precision and recall, showing that the model’s overall performance is well below the acceptable threshold. The confusion matrix reveals significant misclassifications across multiple classes. Hyperparameter tuning using GridSearchCV and RandomizedSearchCV was performed with K-Fold cross-validation, yielding average ROC AUC scores of 0.7595 and 0.7505, respectively. GridSearchCV slightly outperformed RandomizedSearchCV, but both methods showed some improvement over the baseline. **GridSearchCV** performed better than **RandomizedSearchCV**, suggesting that an **exhaustive search** over hyperparameters yields better results in this case.

* KNN MODEL:

The K-Nearest Neighbors (KNN) model, with hyperparameters tuned using GridSearchCV, achieved an accuracy of 44.37%, showing moderate performance. The best hyperparameters found were a **Manhattan distance metric, 10 neighbors**,and **distance-based weights,** which slightly improved the model's ability to classify the data. The precision score of 45.02% indicates that when the model predicts a positive outcome, it is correct about 45% of the time, while the recall score of 44.37% suggests the model captures only about 44% of the true positive cases. The F1 score of 43.56% further reflects the model’s struggle to balance precision and recall effectively. The confusion matrix reveals several misclassifications across multiple classes, particularly where the model has difficulty distinguishing between certain categories.

* SVM MODEL:

The Support Vector Machine (SVM) model, optimized with GridSearchCV, achieved promising results, with a **test set** accuracy of 64.79%. The best hyperparameters, identified through a 10-fold cross-validation process contributing to a cross-validation accuracy of 59%. The precision score of 65.84% indicates that when the model predicts a positive outcome, it is correct about 66% of the time. The recall score of 64.79% suggests that the model captures about 65% of the true positive cases, while the F1 score of 64.95% reflects a balanced performance between precision and recall. The confusion matrix shows the distribution of misclassifications across different classes, with the model performing relatively well in many categories but still struggling with a few others. This performance suggests that the model is performing decently but could benefit from further tuning, especially in terms of improving accuracy on the more challenging classes.

* DECISION TREE MODEL:

The Decision Tree Classifier, optimized with the best hyperparameters, achieved an impressive accuracy of 90.14%. The model’s precision, recall, and F1 score were all strong, with values of 91.68%, 90.14%, and 90.25% respectively, indicating that it is effective at both identifying true positives and minimizing false positives. The confusion matrix shows that the model performs exceptionally well in most classes, with a few misclassifications observed in certain categories. The classification report further highlights the model’s solid performance across different metrics, especially the high F1 scores in many of the classes. Overall, this model delivers high performance and demonstrates that it can handle most of the classification task effectively.

* RANDOM FOREST MODEL:

The **Random Forest Classifier**, optimized with hyperparameters achieved an **accuracy of 85.21%** on the test set. The model demonstrated solid precision, recall, and F1 scores, with values of **84.20%, 85.21%,** and **83.90%** respectively, indicating a well-balanced classification performance. The **OOB score** of **81.48%** further confirms the model's generalization ability. The **classification report** highlights the model’s strong performance across many classes, particularly in class 0, where it achieves perfect recall. Overall, while the Random Forest performs well across the majority of classes, improving its performance on the less-represented classes could help achieve even more balanced results.

* XGBOOST MODEL:

The **XGBoost Classifier** achieved a **test accuracy of 95.07%**, demonstrating outstanding performance on the classification task. The model’s precision, recall, and F1 score are also strong, with values of **93.25%, 95.07%,** and **94.08%**, respectively, indicating a balanced ability to identify positive instances while minimizing false positives. The **best estimator** is an XGBClassifier with optimized hyperparameters. The **best accuracy** from a 5-fold search with 100 parameter combinations reached **96.82%,** indicating that fine-tuning the model's hyperparameters further improved performance. The **confusion matrix** and **probabilities** reflect the model’s ability to confidently predict class labels, with a high degree of certainty, as seen in the values close to 1 for the predicted probabilities of correct classes. Overall, this model is highly effective and performs exceptionally well across various evaluation metrics, suggesting it is well-suited for the given classification problem.

MODEL COMPARISON:

Based on the comparison of different machine learning models, **XGBoost** stands out as the best model for predicting IPL match winners. It achieved the highest accuracy of **95.07%**, which increased to **96.82%** after hyperparameter tuning, outperforming all other models in terms of precision, recall, and F1-score. The model demonstrated exceptional predictive performance and consistency across various evaluation metrics, making it the most reliable choice for the task. While **Decision Tree** also showed strong performance with an accuracy of **90.14%**, XGBoost's superior handling of data and its improved performance through hyperparameter optimization made it the most effective model for this classification problem. **Random Forest** performed well with an accuracy of **85.21%**, but its results were still lower than XGBoost. Therefore, XGBoost is recommended as the optimal model for future IPL match predictions, providing a robust, efficient, and highly accurate solution.

FUTURE STEPS:

In the next steps of the analysis is to implement **K-Fold Cross-Validation** to evaluate how well the models generalize across different data splits. This will involve dividing the dataset into K subsets and training the model K times, using each subset as a validation set while training on the others. This method helps in assessing the model’s performance more reliably, ensuring that it doesn’t overfit to a particular data split. Additionally, perform a **deeper error analysis** by examining the types of errors each model makes, such as identifying misclassifications through confusion matrices and visualizing performance with metrics like ROC curves and precision-recall curves. This step will help to understand the model’s weaknesses and where it is failing, especially regarding class imbalances or overfitting. Finally, implement strategies to address any potential **class imbalance** in the dataset. Class imbalance can lead to biased models. Techniques to handle this include:

* SMOTE (Synthetic Minority Over-sampling Technique): Generates synthetic samples for the minority class.
* Class Weights: Adjusts the importance of the minority class during training, making it more impactful.
* Ensemble Methods: Combine multiple models to improve accuracy, especially on imbalanced datasets.
* Under-sampling Majority Class: Reduces the majority class’s size to balance the dataset.

CONCLUSION:

In conclusion, the XGBoost model has proven to be highly effective in predicting IPL match winners, with exceptional performance metrics, including an accuracy of 95.07%, precision of 93.25%, recall of 95.07%, and an F1-score of 94.08%. To improve the model further, additional hyperparameter tuning and ensemble techniques could enhance its performance, while incorporating new features such as player-specific data, match context, and weather conditions may provide deeper insights. Challenges faced during the process, such as handling missing data, class imbalances, and model interpretability, were mitigated through strategies like data imputation, stratified sampling, and regularization. With further refinement, the model has the potential to be even more accurate and robust in predicting IPL match outcomes.