Benchmarking of Machine Learning algorithms for Indian Premier League matches.

Presented by:

SHREYA MONDAL	21052362
HARDIK CHAUHAN	21052904
SAUMYADEEP MAHANTA	21052918
UZAIF ALI	21052930
JITEN AGARWAL	21052976

INTRODUCTION

IPL and its significance in cricket.

he Indian Premier League (IPL) is a premier T20 cricket league held annually in India, known for its high-octane matches, global player participation, and significant viewership. It has revolutionized the sport by introducing a club-based format and has become a major event in the cricket calendar, contributing to the global popularity of the game.

Importance of predictive modeling in sports analytics.

Predictive modeling in sports analytics plays a crucial role in forecasting outcomes, optimizing player performance, and enhancing strategic decision-making. In the context of the IPL, it allows teams to analyze vast amounts of data to gain competitive advantages and provides insights that contribute to the evolving dynamics of cricket strategy.

TOOLS USED

Python for programming.

Numpy for numerical computations.

Pandas for data manipulation.

Scikit-learn for implementing machine learning models.

METHODOLOGY

Data Collection

Data collection is the first and the foremost step for a successful construction of a model. The accuracy of a model totally depends on the authenticity of the data on which it has been trained. We have gathered data of the matches from 2008 to 2021 from Kaggle and the 2022 and 2023 IPL data has been entered manually.

Our dataset consists of six attributes namely venue, team1, team2, toss_winner, toss_decision and winner. The description of each attribute is given below:

Attribute Name	**Attribute description**
venue	The stadium where the match is being played
team1	Playing team 1
team2	Playing team 2
toss_winner	Team which won the toss
toss_decision	Decision taken by the toss winning team, i.e. bat or field
Winner	Match winning team

METHODOLOGY

Data Preprocessing

Data Filtration

Data filtration is one of the most crucial steps in ensuring reliability and the quality of our predictive model. We have only kept the records of the teams which are currently (i.e. In the year 2024) playing. The teams of which we have gathered the data are 'MI', 'CSK', 'RCB', 'DC', 'GT', 'LSG', 'PBKS', 'RR', 'KKR', 'SRH'. Our filtering process begins by creating a Boolean mask which examines every match entry in our dataset to verify both the competing teams are present in our predefined list of teams. If at least one of the competing teams is not present in our list, then that match record is excluded from further analysis.

Data Transformation

After selecting the appropriate features for our model, the next step is data transformation. Here we have used LabelEncoder class from sklearn.preprocessing module to transform our data. LabelEncoder converts and maps values of each of the attributes into a numerical format that can be interpreted by the predictive algorithms thus allowing our model to learn from these values.

METHODOLOGY

MODELS USED

```
knn = KNeighborsClassifier(n_neighbors=5)
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,
test_size=0.2, random_state=1) #stratify=Y
knn.fit(X_train, Y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_pred, Y_test)
print(f"Accuracy score of test data: {accuracy:.2f}")
train_pred = knn.predict(X_train)
accuracy = accuracy_score(train_pred, Y_train)
print("Accuracy score of train data =
{}".format(accuracy))
```

K-Nearest Neighbors (KNN)

We selected the K-Nearest Neighbours (KNN) algorithm for our IPL match winner prediction. The approach is simple, straightforward and effective, forecasting outcomes based on the closest or the nearest similar samples, known as ‘neighbours. Our specific model is set to consider the five nearest neighbours for its predictions(k=5). We divided our data into two sets: 20% were used to test our model, while the remaining 80% were used to train it. This section helps to guarantee that our model works. After training, we run our model through some tests. With an accuracy score of 0.41 based on the test data, it correctly picked the winner 41% of the time. It was accurate 59% of the time on the training set, with a score of 0.59. These outcomes show how well our approach predicts the winners. The lower score in the test data suggests that we need to make improveme

METHODOLOGY

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
tree_classifier = DecisionTreeClassifier(max_depth=11,
criterion='gini')
tree_classifier.fit(X_train, y_train)
train_pred = tree_classifier.predict(X_train)
accuracy = accuracy_score(train_pred, y_train)
print("Accuracy score of train data =
{}".format(accuracy))
y_pred = tree_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of test data: {accuracy:.2f}")
```

Decision Tree

We have anticipated the results of IPL matches employing a Decision Tree Classifier, a model that renders choices based on an arrangement of questions and criteria. This strategy is comparable to a flowchart, where a node represents a feature, a branch represents a choice rule, and a leaf speaks to an outcome. By restricting the max_depth parameter in our model to 11, we have diminished the number of parts within the tree in arrange to reduce overfitting. We utilized Gini impurity degree to assess a split's quality. The Decision Tree was prepared utilizing the training set, which contains 80% of the information, and the remaining 20% was utilized for testing in order to ensure an adjusted approach to model evaluation. The accuracy score for the training set of information was roughly 81.17%, illustrating a high degree of exactness within the model's predictions. The model's success on concealed information is reflected in its 58% accuracy when applied to the test data.

METHODOLOGY

```
X_train, X_test, y_train, y_test = train_test_split(X,
Y, test_size=0.2, random_state=42)
rf_classifier =
RandomForestClassifier(n_estimators=100,
max_depth=11, random_state=42)
rf_classifier.fit(X_train, y_train)
train_pred = rf_classifier.predict(X_train)
accuracy = accuracy_score(train_pred, y_train)
print("Accuracy score of train data =
{}".format(accuracy))
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of test data: {accuracy:.2f}")
```

Random Forest Classifier (RFC)

Our model employs the random forest classifier, an effective outfit learning procedure which builds various decision trees amid training and decides the mode of the classes (classification) of the individual trees for forecast. This method is profoundly famous for its exceptional exactness and high throughput on huge datasets. We designed our Random Forest Classifier algorithm with estimators=100 to create 100 trees and max_depth=11 to cap the depth of each tree in arrange to anticipate overfitting. The random_state=42 alternative guarantees that our model produces solid and steady results. The model was prepared utilizing 80/20 parts of the information, with 80% going toward training and 20% going toward testing. After training, the model's precision on the training set was roughly 85.37%, showing a high degree of accuracy within the model's forecasts. The model's success on inconspicuous information is reflected in its 53% accuracy when applied to the test data.

METHODOLOGY

```
classifier = svm.SVC(kernel='poly' , degree =7 ,
gamma='scale')
classifier.fit(X_train, y_train)
train_pred = classifier.predict(X_train)
accuracy = accuracy_score(train_pred, y_train)
print("Accuracy score of train data =
{}".format(accuracy))
test_pred = classifier.predict(X_test)
accuracy2 = accuracy_score(test_pred, y_test)
print('Accuracy score on test data =
{}'.format(accuracy2))
```

Support Vector Machine Polynomial

We have included a Support Vector Machine (SVM) with a polynomial kernel in our IPL match winner prediction. Strong classifier SVM finds the ideal hyperplane that maximizes the margin between classes. The model can fit non-linear connections because of the polynomial kernel, which is especially helpful for complicated datasets. The SVM model was constructed with a degree 7 polynomial kernel (degree=7) and a gamma value of "scale," which automatically adjusts the parameter dependant on the number of features to prevent over-fitting. 20% of the dataset was used for model testing, and the remaining 80% was used for training. After training, the model demonstrated a good match to the training set with an accuracy score of roughly 60.33% on the training set. However, the accuracy dropped to 40.15% on the test data. This shows that the model may not generalize to new data as well, even when it has learned patterns from the training set.

METHODOLOGY

```
classifier = svm.SVC(kernel='rbf', gamma='scale')
classifier.fit(X_train, y_train)
train_pred = classifier.predict(X_train)
accuracy = accuracy_score(train_pred, y_train)
print("Accuracy score of train data =
{}".format(accuracy))
test_pred = classifier.predict(X_test)
accuracy2 = accuracy_score(test_pred, y_test)
print('Accuracy score on test data =
{}'.format(accuracy2))
```

Support Vector Machine Radial Basis Function

Here, we included a Support Vector Machine (SVM) with a Radial Basis Function (RBF). The SVM model was constructed with an RBF kernel and a gamma value of "scale," which automatically adjusts the parameter dependant on the number of features. 20% of the dataset was used for model testing, and the remaining 80% was used for training. The accuracy score of training data was roughly 45.70% and of testing was data 38.69%.

METHODOLOGY

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
lgbm_classifier = LGBMClassifier(boosting_type='gbdt
num_leaves=4, learning_rate=0.1, n_estimators=200)
lgbm_classifier.fit(X_train, y_train)
train_pred = lgbm_classifier.predict(X_train)
accuracy = accuracy_score(train_pred, y_train)
print("Accuracy score of train data =
{}".format(accuracy))
y_pred = lgbm_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy score of test data: {accuracy:.2f}")
```

LightGBM

For our prediction of the IPL match winner, we have consolidated LightGBM, an effective gradient boosting system that utilizes tree-based learning strategies. LightGBM is exceedingly known for its proficiency and speed, particularly when working with expansive datasets, and for its ability in overseeing categorical highlights. We designed our LightGBM classifier with boosting_type='gbdt' for gradient boosted decision trees, num_leaves=4 to control the tree model's complexity, learning_rate=0.1 to decide each tree's effect on the result, and n_estimators=200 to decide the number of boosted trees to be made. These settings were chosen to adjust model complexity and training time. The dataset was separated into two parts: 20% was saved for testing and the remaining 80% for training. The model's accuracy of 84.10% on the training set shows a great match to the training data. The model can generalize to unused data rather well, however it might still require a few fine-tuning, as seen by its 54% accuracy on the test set.

METHODOLOGY

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
train_pred = nb_classifier.predict(X_train)
accuracy = accuracy_score(train_pred, y_train)
print("Accuracy score of train data =
{}".format(accuracy))
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy score of test data: {accuracy:.2f}")
```

Naïve Bayes

The Naive Bayes classifier in this study is the Gaussian Naive Bayes variant, which works well with continuous data and is based on the idea that the features have a normal distribution. Under strict independence requirements between features, the probabilistic classifier Naive Bayes applies the Bayes theorem. With test_size=0.2, the dataset was split 80-20, using 80% for training and 20% for testing, so that the model could be trained. By using random_state=42, the split's repeatability is assured. With training data, the Gaussian Naive Bayes classifier achieved an accuracy score of 38.76%, whereas with test data, it was approximately 31%. The model's ability to predict accurate outcomes based on training data and its generalization to new, untrained data are both demonstrated by these evaluations. The low accuracy values show that the Naive Bayes classifier, despite being a quick and easy model, might not be the ideal choice for this specific prediction problem.

METHODOLOGY

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=41)
model = CatBoostClassifier(iterations=100, depth=4,
learning_rate=0.1, loss_function='MultiClass',
verbose=False)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of test data: {accuracy:.2f}")
train_pred = model.predict(X_train)
accuracy = accuracy_score(train_pred, y_train)
print("Accuracy score of train data =
{}".format(accuracy))
```

CatBoost

Our model utilizes the CatBoost Classifier, a complex gradient boosting method that's especially viable when utilized to categorize data. "Categorical Boosting," or "CatBoost," may be a framework planned to supply amazing performance with speed and precision. The model was designed with the following parameters: depth=4 to control the complexity of the model, iterations=100 to produce 100 trees, loss_function='MultiClass' reasonable for multi-class classification issues, and learning_rate=0.1 to influence the redress of each tree. The verbose=False parameter keeps up organization within the training output. We part our dataset into training and testing sets utilizing an 80-20 proportion, and we used random_state = 41 to guarantee consistency in our results. With an accuracy of 66.54% on the training set, the CatBoost demonstrated to be a great fit for the training set of information. The model can generalize to unused/ unseen data, as evidenced by its 55% accuracy on the test set, in spite of the fact that there may be room for enhancement.

RESULT ANALYSIS

Model	Train Accuracy	Test Accuracy	Over/Underfitting	Generalization Ability
KNN	59.05%	41%	NA	Moderate
Decision Tree	81.17%	58%	NA	Moderate
Random Forest	85.37%	53%	NA	Moderate
SVM Polynomial	60.33%	40.15%	NA	Moderate
SVM RBF	45.70%	38.69%	NA	Low
LightGBM	84.09%	54%	NA	Moderate
Naive Bayes	38.76%	31%	NA	Low
CatBoost	66.54%	55%	NA	Moderate

The decision tree model stands out of all the model tested with the highest test accuracy suggesting it the most effective model. However, the LightGBM and Random Forest models have potential but are required to be tuned to overcome the overfitting condition. Naive Bayes and SVM with RBF Kernel models require a reconsideration of feature selection or model parameters to enhance their predictive power.

CONCLUSION

The challenge of predicting T20 cricket match outcomes lies in the game's inherent unpredictability and the multitude of influencing factors. This model, which incorporates venue, toss decisions, and winner data, has been tested with various machine learning algorithms. The Decision Tree algorithm emerged as the most accurate, with a test accuracy of 58%. The model's training process is crucial, utilizing historical data from 2008 to 2023, with the exception of 2020 and 2021, to learn and predict outcomes. Despite its successes, the model has limitations, such as not considering pitch conditions, player strike rates, bowler economy, and form, which are significant in cricket. Additionally, external factors like weather conditions, which can sway the game's direction, are also not accounted for. Future enhancements could include these variables for a more comprehensive prediction model.

FUTURE SCOPE

Such models can also be used to predict various other things such as the total fantasy points of the player by the end of this season, best playing 11, best team of the season. The model can also be expanded to different leagues in other countries which could help to bring good talents under the lime light. Player specific models can also be created by including more features relevant to the performances of the players. This research can have more features added to improve the overall accuracy of the model and to expand its applicability to predict a wide range of events.

REFERENCES

- A. Ghosh, A. Sinha, P. Mondal, A. Roy and P. Saha: Indian Premier League Player Selection Model Based on Indian Domestic League Performance

- A. Santra, A. Sinha, P. Saha A. K. Das: A Novel Regression based Technique for Batsman Evaluation in the Indian Premier League

- (2002) The IEEE website. [Online]. Available: http://www.ieee.org/

- Wikipedia [Online] Available: https://www.wikipedia.org/