Machine Learning Project

IPL Win Predictor

Submitted by:

(102103546) Aditya Vashishta
(102103551) Sarthak Gautam

Submitted to:

Mr Harpreet

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Report: IPL Win Predictor

- **Introduction :**

Cricket is a sport that involves various strategies, and predicting the outcome of a match is an intriguing challenge. This report explores the development of an IPL (Indian Premier League) cricket match win predictor using machine learning techniques.

- **Dataset Description main files :**

Deliveries.csv **:** **(**https://www.kaggle.com/datasets/ramjidoolla/ipl-data-set?select=deliveries.csv**)**

This section contains detailed information about each delivery in IPL cricket matches. Here are the fields in this section:

- inning: Inning number (1st inning or 2nd inning).
- match_id: Unique identifier for each IPL match.
- batting_team: Team that is currently batting.
- bowling_team: Team that is currently bowling.
- over: The current over in the match.
- ball: The current ball being bowled in the over.
- batsman: The batsman facing the current ball.
- non_striker: The non-striker at the other end.
- bowler: The bowler delivering the current ball.
- is_super_over: Indicates whether the current over is a Super Over (True/False).
- wide_runs: Extra runs awarded for a wide ball.
- bye_runs: Extra runs awarded for byes.
- legbye_runs: Extra runs awarded for leg byes.
- noball_runs: Extra runs awarded for a no-ball.
- penalty_runs: Extra runs awarded as a penalty.
- batsman_runs: Runs scored by the batsman off the current ball.
- extra_runs: Total extra runs (wide, bye, leg bye, no ball, penalty) in the current ball.
- total_runs: Total runs (batsman runs + extra runs) in the current ball.
- player_dismissed: Batsman who got dismissed (if any) on the current ball.
- dismissal_kind: Type of dismissal (e.g., caught, bowled, lbw, etc.) if a dismissal occurred.
- fielder: Player involved in the dismissal (e.g., the fielder who caught the ball).
-

Matches.csv **:** (https://www.kaggle.com/datasets/ramjidoolla/ipl-data-set?select=matches.csv)

This section contains general information about IPL matches:

- Season: The season in which the IPL match took place.
- Id : Unique Identifier for each match
- city: The city where the match was held.
- date: The date on which the match was played.
- team1: The first team playing in the match.
- team2: The second team playing in the match.
- toss_winner: The team that won the toss.
- toss_decision: The decision made by the toss-winning team (batting or bowling).
- result: The result of the match (normal, tie, no result).
- dl_applied: Whether the Duckworth-Lewis method was applied in the match (True/False).
- winner: The team that won the match.
- win_by_runs: The margin of victory (in runs) if the team batting first won.
- win_by_wickets: The margin of victory (in wickets) if the team batting second won.
- player_of_match: The player awarded as the Man of the Match.
- venue: The stadium or venue where the match was played.
- umpire1: The first on-field umpire.
- umpire2: The second on-field umpire.
- umpire3: The third umpire (if any).

These datasets seems to provide a comprehensive overview of IPL cricket matches, including detailed delivery-level data along with match-specific information. It can be used for various analyses such as player performance, team strategies, match outcomes, and more.

The key steps in data preprocessing include:

- Calculating the total score for each innings in the match and appending Target score.
- Removing Teams that are no longer actively playing cricket and changing names.
- Handling missing values and filtering out matches with DL (Duckworth-Lewis) method applied.
- Manipulating the dataset to train on 2nd innings data only.
- Merging columns to create new features that reduce redundancy in data.
- Removing irrelevant columns and calculating current score , runs left , balls left and wickets left , crr , rrr , result.

- Merging match information with total scores to create a consolidated dataset.

- **Used Methodology :**

In our machine learning project, we conducted a comprehensive analysis to identify the most suitable algorithm for predicting IPL cricket match outcomes. The methodology can be summarized as follows:

1. **Dataset Selection :**

   We utilized two main datasets, namely `Deliveries.csv` and `Matches.csv`, containing detailed information about IPL matches.

2. **Feature Selection :**

   We considered a set of relevant features for prediction, including details about the batting team, bowling team, host city, runs left, balls left, wickets left, total runs, current run rate (CRR), and required run rate (RRR).

3. **Data Preprocessing :**

   - The data underwent a crucial step of splitting into training and testing sets to assess the model's performance accurately.

   - Categorical features were one-hot encoded to facilitate their inclusion in the model.

4. **Algorithm Evaluation :**

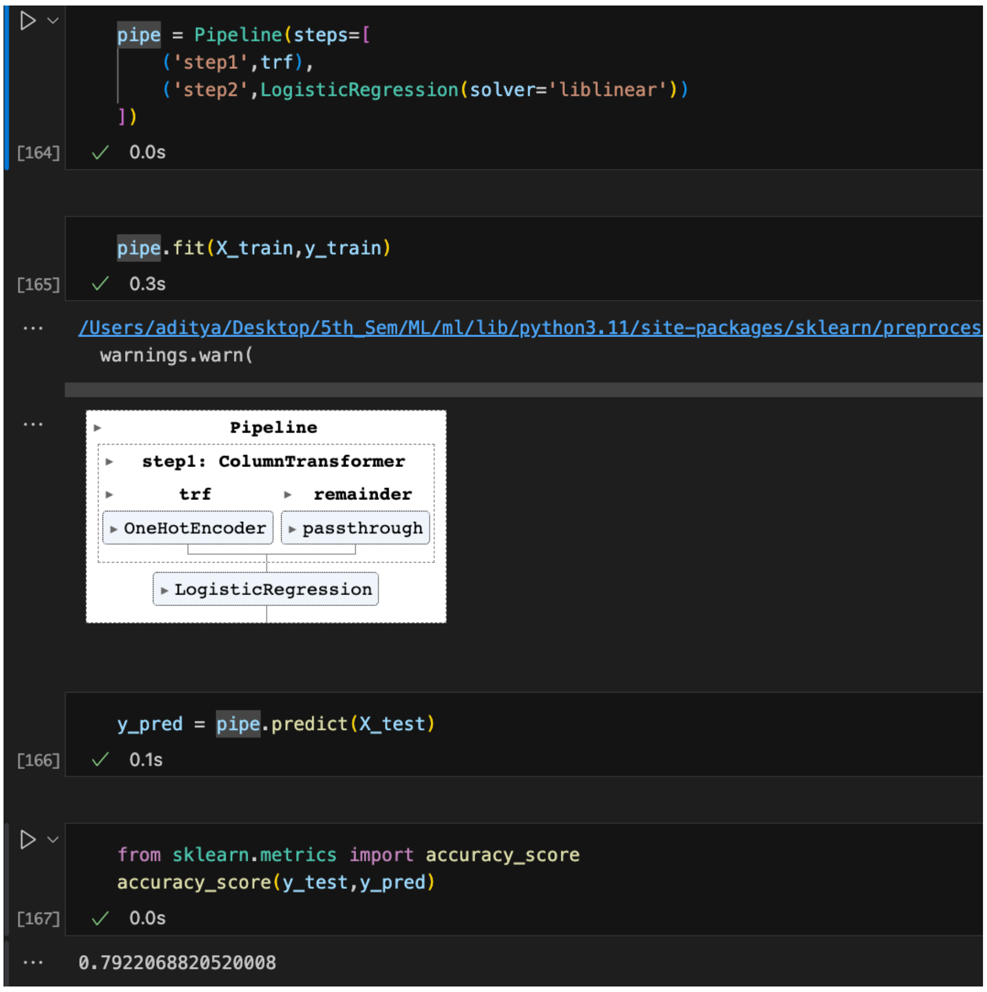
   - We explored the efficacy of different algorithms, specifically testing all solvers available for logistic regression and the Random Forest algorithm.

- The solvers for logistic regression were assessed to determine their impact on predictive accuracy.

Logistic regression is a classification algorithm used for predicting binary outcomes. In scikit-learn, the logistic regression model provides different solvers to optimize the model's parameters. Each solver uses a different optimization algorithm to find the weights that maximize the likelihood of the observed data. Here are insights into the common solvers available in scikit-learn's logistic regression and considerations for their use in the context of predicting IPL cricket match outcomes:

4.1 **Liblinear :**

- Insight: Suitable for small to medium-sized datasets, particularly when the number of features is significant compared to the number of samples.
- Considerations: May not scale well for large datasets, and it is not the best choice when dealing with multicollinearity.



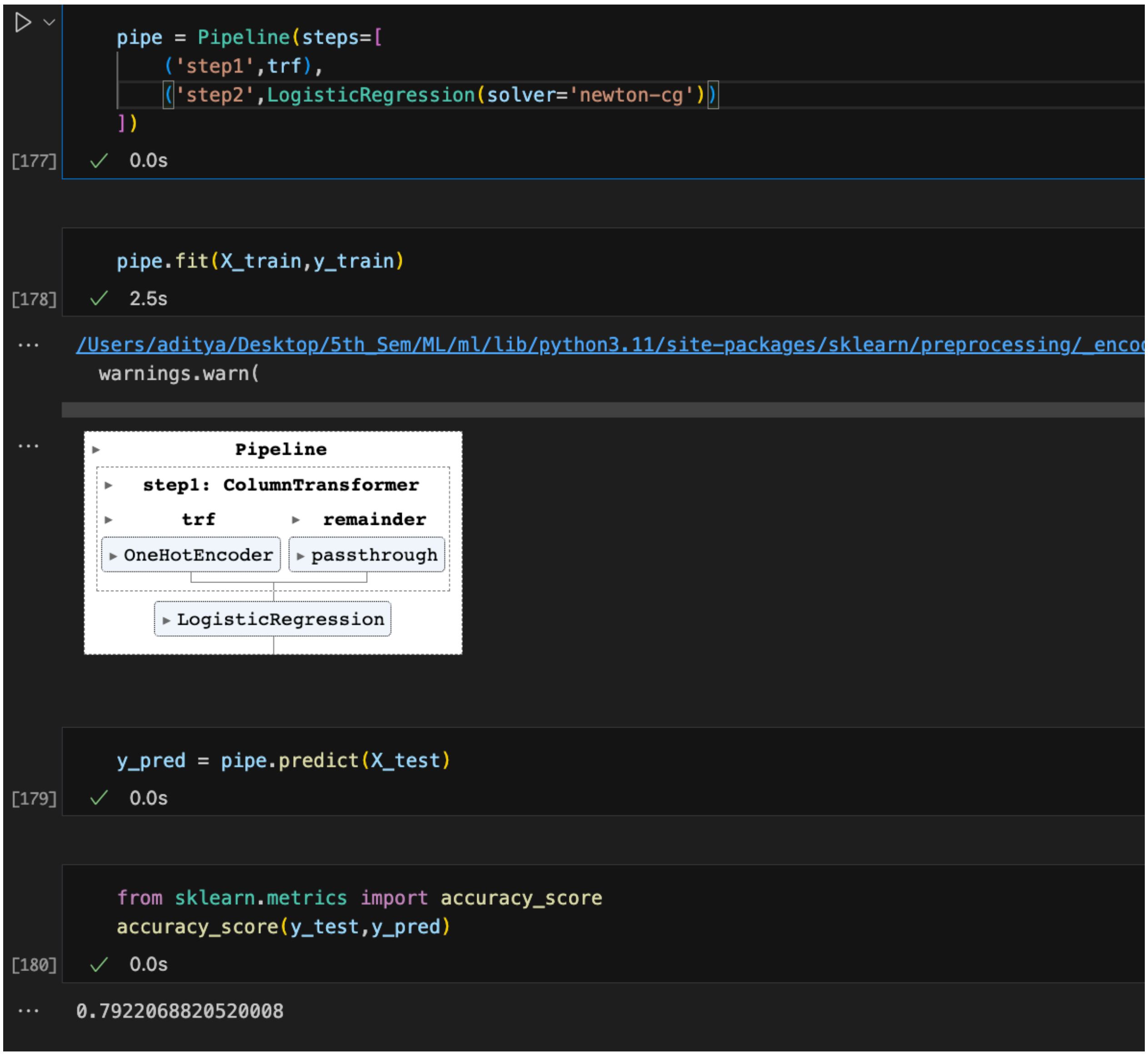
```
pipe = Pipeline(steps=[
    ('step1',trf),
    ('step2',LogisticRegression(solver='liblinear'))
])
[164] ✓ 0.0s

pipe.fit(X_train,y_train)
[165] ✓ 0.3s

/Users/aditya/Desktop/5th_Sem/ML/ml/lib/python3.11/site-packages/sklearn/preproces
  warnings.warn(
```

```
y_pred = pipe.predict(X_test)
[166] ✓ 0.1s

from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
[167] ✓ 0.0s

0.7922068820520008
```

4.2 **Newton-CG :**

- Insight: Well-suited for datasets with a large number of features. It uses a Newton method to optimize the weights.

- Considerations: Can be computationally expensive for very large datasets. It also requires the computation of the Hessian matrix, which can be memory-intensive.



```
pipe = Pipeline(steps=[
    ('step1',trf),
    ('step2',LogisticRegression(solver='newton-cg'))
])
[177]  ✓ 0.0s
```

```
pipe.fit(X_train,y_train)
[178]  ✓ 2.5s
... /Users/aditya/Desktop/5th_Sem/ML/ml/lib/python3.11/site-packages/sklearn/preprocessing/_encod
      warnings.warn(
```

```
... Pipeline
      step1: ColumnTransformer
      trf                 remainder
      OneHotEncoder       passthrough
      LogisticRegression
```

```
y_pred = pipe.predict(X_test)
[179]  ✓ 0.0s
```

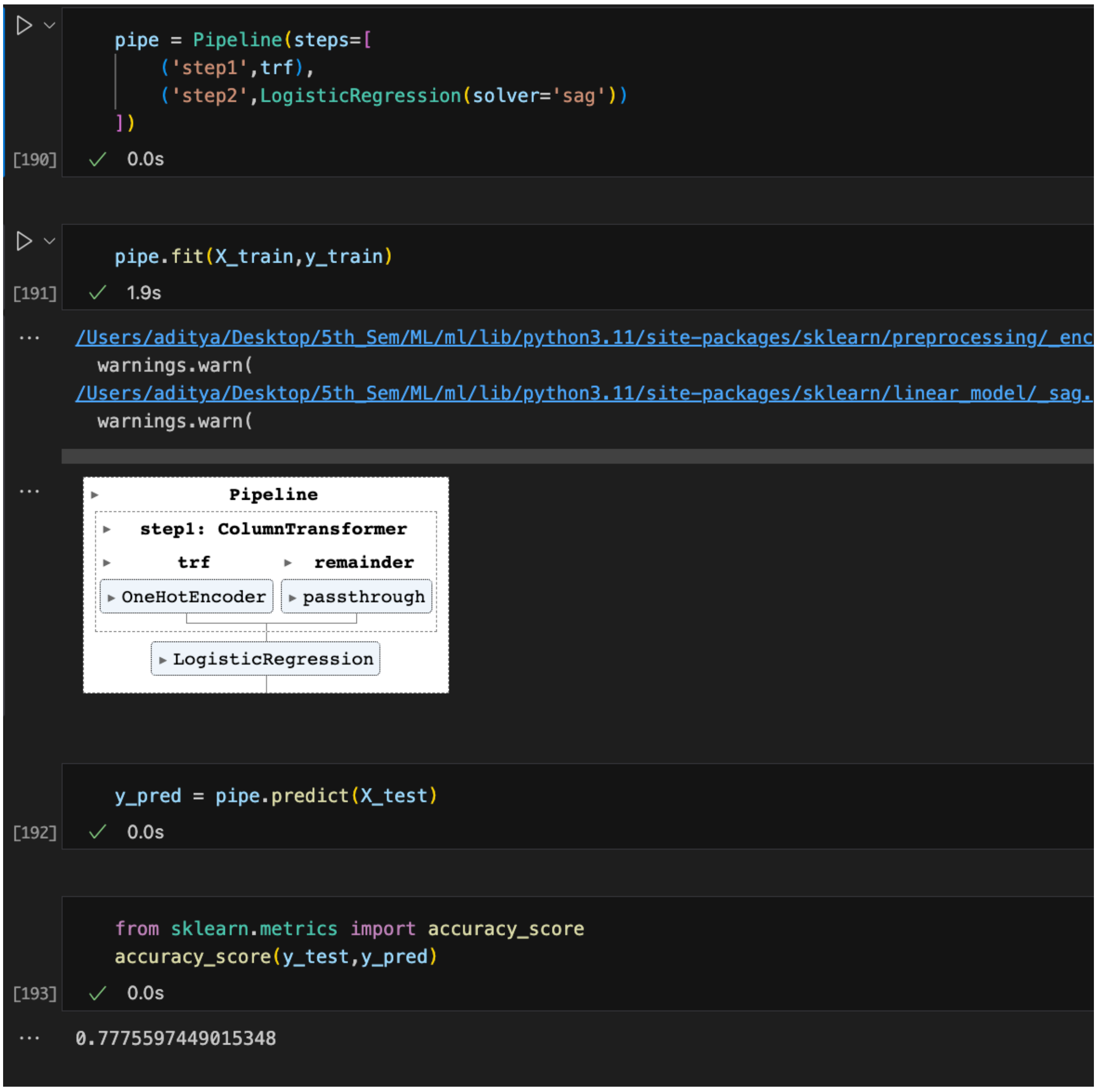
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
[180]  ✓ 0.0s
... 0.7922068820520008
```

4.3 **Sag (Stochastic Average Gradient):**

- Insight: Designed for large datasets and can be more efficient than other solvers, especially when the dataset is sparse.

- Considerations: Appropriate for large datasets with a high number of samples and features.

However, it may converge more slowly on smaller datasets.



```
pipe = Pipeline(steps=[
    ('step1',trf),
    ('step2',LogisticRegression(solver='sag'))
])
```
[190] ✓ 0.0s

```
pipe.fit(X_train,y_train)
```
[191] ✓ 1.9s

```
/Users/aditya/Desktop/5th_Sem/ML/ml/lib/python3.11/site-packages/sklearn/preprocessing/_enc
  warnings.warn(
/Users/aditya/Desktop/5th_Sem/ML/ml/lib/python3.11/site-packages/sklearn/linear_model/_sag.
  warnings.warn(
```

```
y_pred = pipe.predict(X_test)
```
[192] ✓ 0.0s

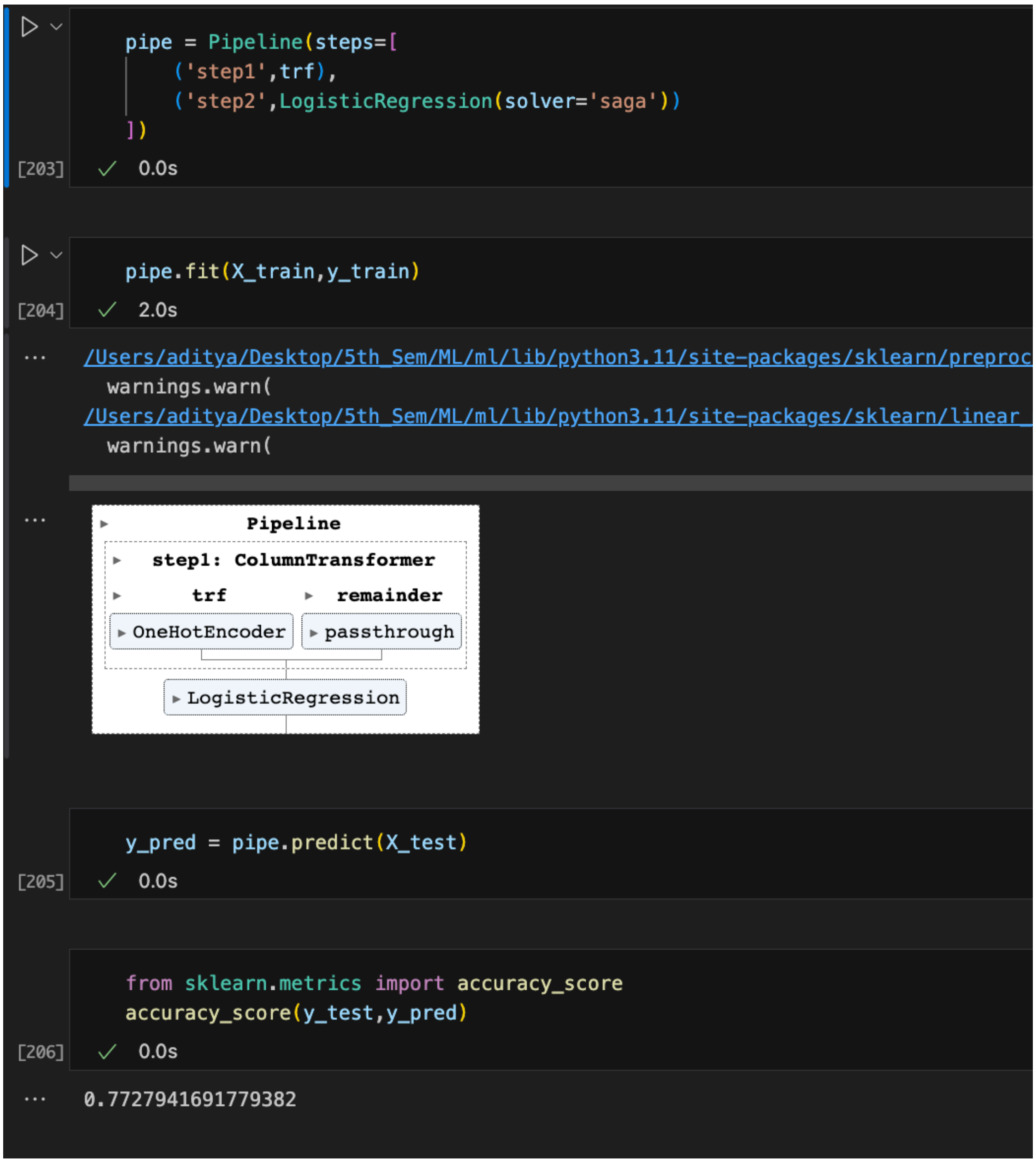
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```
[193] ✓ 0.0s

```
0.7775597449015348
```

4.4 **Saga (Stochastic Average Gradient Descent) :**

- Insight: An extension of Sag that also supports L1 regularization.

- Considerations: Useful when feature selection or sparsity is important. Can be slower to converge compared to other solvers.



```
pipe = Pipeline(steps=[
    ('step1',trf),
    ('step2',LogisticRegression(solver='saga'))
])
```
[203] ✓ 0.0s

```
pipe.fit(X_train,y_train)
```
[204] ✓ 2.0s

```
/Users/aditya/Desktop/5th_Sem/ML/ml/lib/python3.11/site-packages/sklearn/preproc
  warnings.warn(
/Users/aditya/Desktop/5th_Sem/ML/ml/lib/python3.11/site-packages/sklearn/linear_
  warnings.warn(
```

```
Pipeline
step1: ColumnTransformer
trf                remainder
OneHotEncoder      passthrough
LogisticRegression
```

```
y_pred = pipe.predict(X_test)
```
[205] ✓ 0.0s

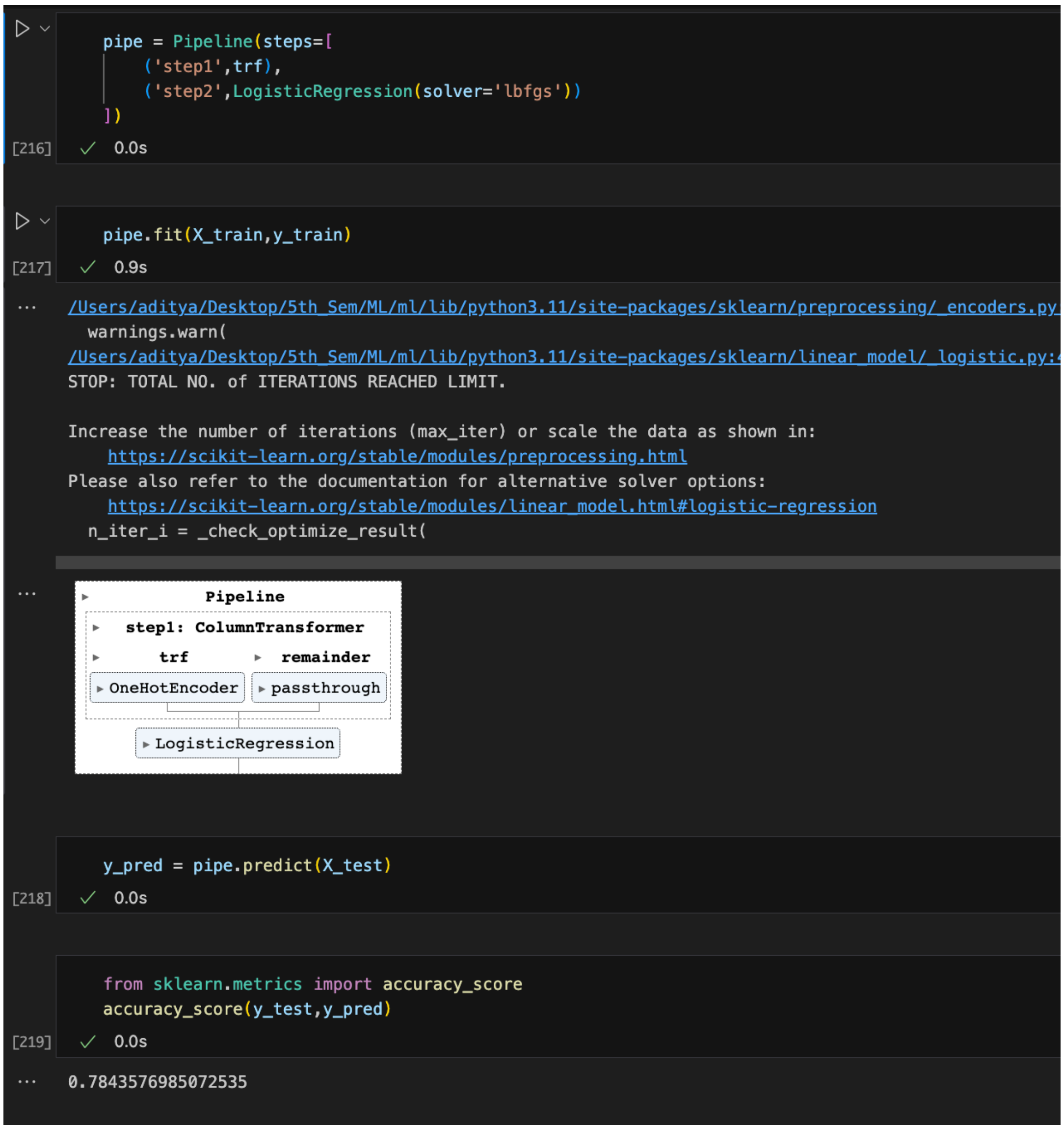
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```
[206] ✓ 0.0s

```
0.7727941691779382
```

4.5 **Lbfgs (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) :**

- Insight: A quasi-Newton method that works well for moderate-sized datasets.

- Considerations: Efficient for problems with a moderate number of samples. May not be the best choice for high-dimensional data.



```
pipe = Pipeline(steps=[
    ('step1',trf),
    ('step2',LogisticRegression(solver='lbfgs'))
])
```
[216] ✓ 0.0s

```
pipe.fit(X_train,y_train)
```
[217] ✓ 0.9s

```
/Users/aditya/Desktop/5th_Sem/ML/ml/lib/python3.11/site-packages/sklearn/preprocessing/_encoders.py
  warnings.warn(
/Users/aditya/Desktop/5th_Sem/ML/ml/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
y_pred = pipe.predict(X_test)
```
[218] ✓ 0.0s

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```
[219] ✓ 0.0s

```
0.7843576985072535
```

It's essential to experiment with different solvers, monitor convergence, and choose the one that balances computational efficiency and predictive accuracy for the specific characteristics of your dataset.

While Random Forest is a powerful algorithm for classification and regression tasks, there are specific considerations that might make Logistic Regression a more suitable choice, especially when prediction is done based on probabilities. Here are detailed reasons why Random Forest might not be the optimal choice and the advantages that Logistic Regression offers in this context:

Random Forest Considerations :

1. **Lack of Probability Interpretability:**

- Issue: Random Forest produces predictions by averaging the results of multiple decision trees. While it's excellent for making accurate predictions, it doesn't provide easily interpretable probabilities.

- Importance: In scenarios where understanding the probability of a certain outcome is crucial (such as predicting match outcomes in cricket), having a clear interpretation of predicted probabilities is valuable.

2. **Complexity and Overfitting :**

- Issue: Random Forest tends to be a more complex model, consisting of multiple decision trees.

This complexity can lead to overfitting, especially when dealing with smaller datasets.

- **Importance :** Overfitting might result in less reliable probability estimates, particularly when the model is trying to capture noise in the data rather than true underlying patterns.

While Random Forest is a versatile and powerful algorithm, Logistic Regression's interpretability, probabilistic output, and regularization properties make it particularly well-suited for scenarios where predictions are based on probabilities, and the interpretability of the model's predictions is crucial.

5. **Model Training :**

- Logistic Regression Pipeline: Constructed a logistic regression pipeline to streamline the training process.
- Random Forest: Trained the Random Forest algorithm to evaluate its performance in comparison to logistic regression.

6. **Performance Assessment :**

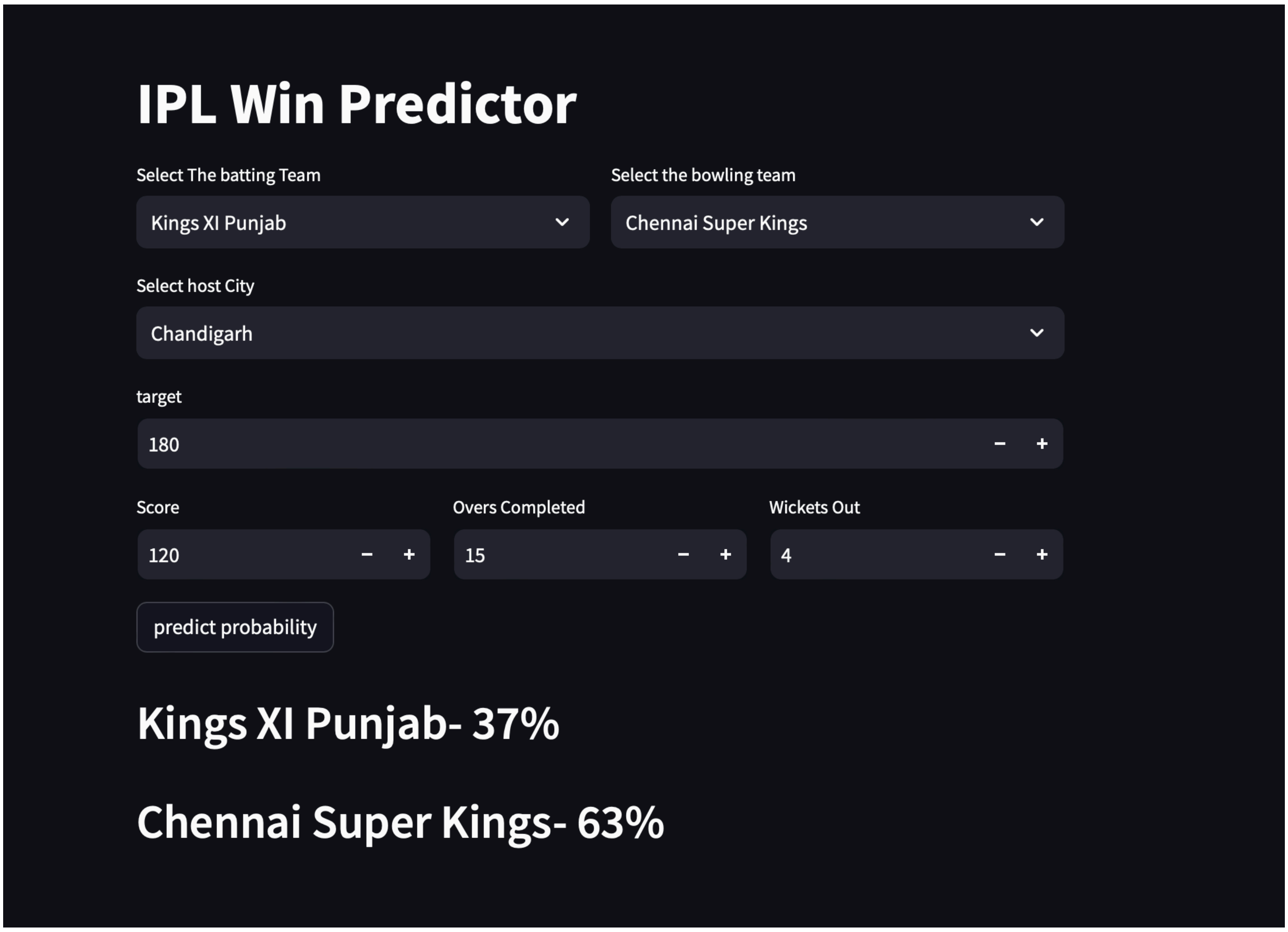
- Evaluated the performance of each algorithm using appropriate metrics, such as accuracy, to determine their effectiveness in predicting IPL match outcomes.

By systematically comparing various solvers for logistic regression and employing the Random Forest algorithm, our methodology aimed to identify the most suitable algorithm for achieving accurate predictions in the context of IPL cricket matches. The results of this analysis provide valuable insights into the strengths and limitations of different algorithms for this specific prediction task.

- **Result :**

The trained model demonstrates promising predictive capabilities. The Streamlit web application allows users to input match details and obtain predictions for the probability of winning or losing for the selected teams.

- **Sample Output :**



IPL Win Predictor

Select The batting Team: Kings XI Punjab

Select the bowling team: Chennai Super Kings

Select host City: Chandigarh

target: 180

Score: 120

Overs Completed: 15

Wickets Out: 4

Kings XI Punjab- 37%

Chennai Super Kings- 63%