VASHISHT HACKATHON

TEAM SEEMO PROJECT REPORT

TOPIC : PREDICTION OF SPORTS RESULTS USING MACHINE LEARNING

TEAM MEMBERS:

SEETHA MAHALAKSHMI C H:
9921005109@klu.ac.in
NALLAGANTLA MOUNIKA
nallagantlamounika@gmail.com
YASWANTH SRINIVAS
9921005107@klu.ac.in
GNANA CHAITANYA TURE
9822005004@klu.ac.in

ABSTRACT

This paper describes the use of machine learning in sports. Given the recent trend in Data science and sport analytics, the use of Machine Learning and Data Mining as techniques in sport reveals the essential contribution of technology in results and performance prediction. The purpose of this paper is to benchmark existing analysis methods used in literature, to understand the prediction processes used to model Data collection and its analysis; and determine the characteristics of the variables controlling the performance. Finally, this paper will suggest the reliable tool for Data mining analysis technique using Machine Learning

In this project we have done kabaddi sport for predicting the result by taking recent data of every team and their match ,wins and loses with other teams .we used logistic regression , decision tree classifier , random forest classifier

For training the data set ,for accuracy and result .

CODE:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from datetime import datetime, timedelta
# import all libraries and dependencies for data
visualization
pd.options.display.float_format='{:.4f}'.format
plt.rcParams['figure.figsize'] = [8,8]
pd.set_option('display.max_columns', 500)
pd.set_option('display.max_colwidth', -1)
sns.set(style='darkgrid')
import matplotlib.ticker as ticker
import matplotlib.ticker as plticker
# import all libraries and dependencies for machine
learning
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.base import TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing
from sklearn.base import TransformerMixin
from sklearn.metrics import roc_curve
# Local file path. Please change the file path
accordingly
path = '/content/2019_Pro_Kabaddi.xlsx'
```

```
# Reading the Team_leaderboard file on which Analysis
needs to be done
file = "2019_Pro_Kabaddi.xlsx"
df_mr = pd.read_excel(file)
df_mr.head(100)
df_mr = df_mr.drop(['Week','Team 1_Score','Team
2_Score','MarginScore']
,axis=1)
df_mr.head()#Building the model
df_mr = df_mr.reset_index(drop=True)
df_mr.loc[df_mr.Winner ==
df_mr.Team_1,'winning_team']=1
df_mr.loc[df_mr.Winner == df_mr.Team_2,
'winning_team']=2
df_mr = df_mr.drop(['winning_team'], axis=1)
df_mr.head()
#convert team-1 and team-
2 from categorical variables to continous inputs
# Get dummy variables
final = pd.get_dummies(df_mr, prefix=['Team_1',
'Team_2'], columns=['Te
am_1', 'Team_2'])
# Separate X and y sets
X = final.drop(['Winner'], axis=1)
y = final["Winner"]
# Separate train and test sets
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3
0, random_state=100)
#logistic Regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
Y_pred = logreg.predict(X_test)
score = logreg.score(X_train, y_train)
score2 = logreg.score(X_test, y_test)
print("Training set accuracy: ", '%.3f'%(score))
print("Test set accuracy: ", '%.3f'%(score2))
#Decision Tree Classifierdecision_tree =
DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
Y_pred = decision_tree.predict(X_test)
score = decision_tree.score(X_train, y_train)
score2 = decision_tree.score(X_test, y_test)
print("Training set accuracy: ", '%.3f'%(score))
```

```
print("Test set accuracy: ", '%.3f'%(score2))
#Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100,
max_depth=50,
random_state=200)
rf.fit(X_train, y_train)
score = rf.score(X_train, y_train)
score2 = rf.score(X_test, y_test)
print("Training set accuracy: ", '%.3f'%(score))
print("Test set accuracy: ", '%.3f'%(score2))
Model=[
#ensmble method
RandomForestClassifier(),
#GLM
LogisticRegression(),
#Trees
DecisionTreeClassifier(),
]
Model_columns = []
Model_compare = pd.DataFrame(columns = Model_columns)
row_index = 0
for alg in Model:
predicted = alg.fit(X_train,y_train).predict(X_test)
Model_name = alg.__class__.__name__
Model_compare.loc[row_index,'Model Name'] =
Model_name
Model_compare.loc[row_index, 'Model Train Accuracy']
= round(alg.sc
ore(X_train,y_train), 4)
Model_compare.loc[row_index, 'Model Test Accuracy'] =
round(alg.sco
re(X_test,y_test), 4)
row_index+=1
Model_compare.sort_values(by = ['Model Test
Accuracy'], ascending = Fal
se, inplace = True)
Model_compare
plt.subplots(figsize=(6,6))
sns.barplot(x="Model Name", y="Model Train
Accuracy",data=Model_compare
,palette='hot',edgecolor=sns.color_palette('bright',3
))
plt.xticks(rotation=90)
plt.title('Model Train Accuracy Comparison')
```

```
plt.show()
plt.subplots(figsize=(7,6))
sns.barplot(x="Model Name", y="Model Test
Accuracy",data=Model_compare,
palette='hot',edgecolor=sns.color_palette('dark',7))
plt.xticks(rotation=90)
plt.title('Model Test Accuracy Comparison')
plt.show()
path = '../input/pro-kabaddi-2019/'
file = "../content/2019_Pro_Kabaddi.xlsx"
ranking = pd.read_excel(file,'Points Table_2019')
fixtures =
pd.read_csv("/content/kabbadi_fixtures.csv.xls")
# List for storing the group stage games
pred_set = []
fixtures.head(135)
ranking.head(12)# Create new columns with wins of
each team
fixtures.insert(1, 'First_Position',
fixtures['Team_1'].map(ranking.set
_index('Team')['WINS']))
fixtures.insert(2, 'Second_Position',
fixtures['Team_2'].map(ranking.se
t_index('Team')['WINS']))
# We only need the group stage games, so we have to
slice the dataset
fixtures = fixtures.iloc[:131, :]
fixtures.head(10)
for index, row in fixtures.iterrows():
if row['First_Position'] > row['Second_Position']:
pred_set.append({'Team_1': row['Team_1'], 'Team_2':
row['Team_2
'], 'winning_team': None})
else:
pred_set.append({'Team_1': row['Team_2'], 'Team_2':
row['Team_1
'], 'winning_team': None})
pred_set = pd.DataFrame(pred_set)
backup_pred_set = pred_set
# Get dummy variables and drop winning_team column
pred_set = pd.get_dummies(pred_set, prefix=['Team_1',
'Team_2'], column
s=['Team_1', 'Team_2'])
```

```
# Add missing columns compared to the model's
training dataset
missing_cols = set(final.columns) -
set(pred_set.columns)
for c in missing_cols:
pred_set[c] = 0
pred_set = pred_set[final.columns]
pred_set = pred_set.drop(['Winner'], axis=1)
pred_set.head()
#group matches
predictions = rf.predict(pred_set)for i in
range(fixtures.shape[0]):
print(backup_pred_set.iloc[i, 1] + " and " +
backup_pred_set.iloc[i
, 0])
if predictions[i] == 1:
print("Winner: " + backup_pred_set.iloc[i, 1])
else:
print("Winner: " + backup_pred_set.iloc[i, 0])
print("")
def clean_and_predict(matches, ranking, final, rf):
positions = []
# Loop to retrieve each team's position according to
Wins
for match in matches:
positions.append(ranking.loc[ranking['Team'] ==
match[0],'WINS'
].iloc[0])
positions.append(ranking.loc[ranking['Team'] ==
match[1],'WINS'
].iloc[0])
# Creating the DataFrame for prediction
pred_set = []
i = 0
j = 0
while i < len(positions):
dict1 = {}
# If wins of first team is better then this team will
be the 'T
eam_1' team, and vice-versa
if positions[i] > positions[i + 1]:
dict1.update({'Team_1': matches[j][0], 'Team_2':
matches[j]
[1]})
```

```
else:
dict1.update({'Team_1': matches[j][1], 'Team_2':
matches[j]
[0]})
# Append updated dictionary to the list, that will
later be con
verted into a DataFrame
pred_set.append(dict1)
i += 2 j += 1
# Convert list into DataFrame
pred_set = pd.DataFrame(pred_set)
backup_pred_set = pred_set
# Get dummy variables and drop winning_team column
pred_set = pd.get_dummies(pred_set, prefix=['Team_1',
'Team_2'], co
lumns=['Team_1', 'Team_2'])
# Add missing columns compared to the model's
training dataset
missing_cols2 = set(final.columns) -
set(pred_set.columns)
for c in missing_cols2:
pred_set[c] = 0
pred_set = pred_set[final.columns]
pred_set = pred_set.drop(['Winner'], axis=1)
# Predict
predictions = rf.predict(pred_set)
for i in range(len(pred_set)):
print(backup_pred_set.iloc[i, 1] + " and " +
backup_pred_set.il
oc[i, 0])
if predictions[i] == 1:
print("Winner: " + backup_pred_set.iloc[i, 1])
else:
print("Winner: " + backup_pred_set.iloc[i, 0])
print("")
elim1 = [('Haryana Steelers', 'Bengaluru Bulls')]
elim2 = [('U Mumba', 'UP Yoddha')]
clean_and_predict(elim1, ranking, final, rf)
clean_and_predict(elim2, ranking, final, rf)
semi1 = [('Haryana Steelers', 'Dabang Delhi')]
semi2 = [('UP Yoddha', 'Bengal
Warriors')]clean_and_predict(semi1, ranking, final,
rf)
clean_and_predict(semi2, ranking, final, rf)
```

```
finale = [('Dabang Delhi','Bengal Warriors')]
clean_and_predict(finale, ranking, final, rf)
file="/content/Player_Stat_Season 7.xlsx"
# Reading the Team_leaderboard file on which Analysis
needs to be done

df_Player_Total_points = pd.read_excel(file,'Total
Points')
df_Player_Total_points.head()
df_Player_Total_points['Avg Player Points'] =
df_Player_Total_points['P
oints']/df_Player_Total_points['Matches']
df_Player_Total_points =
df_Player_Total_points.drop(['Points','Matches
'],axis=1)
df_Player_Total_points =
df_Player_Total_points.sort_values(by='Avg Pla
yer Points',ascending=False)
df_Player_Raid_points = pd.read_excel(file,'Raid
Points')
#df_Player_Raid_points.head(2)
df_Player_Raid_points['Avg Player Raid Points'] =
df_Player_Raid_points
['Points']/df_Player_Raid_points['Matches']df_Player_
Raid_points =
df_Player_Raid_points.drop(['Points','Matches']
,axis=1)
df_Player_Raid_points =
df_Player_Raid_points.sort_values(by='Avg Playe
r Raid Points',ascending=False)
plt.subplots(figsize = (12,8))
ax = sns.barplot(x = 'Player', y = 'Avg Player
Points',data=df_Player_T
otal_points)
plt.title("Avg Player Points of Player in S7",
fontsize = 18)
plt.ylabel("Avg Player Points", fontsize = 15)
plt.xlabel("Player",fontsize = 15)
ax=ax.set_xticklabels(ax.get_xticklabels(),
rotation=75)
plt.subplots(figsize = (7,8))
ax = sns.barplot(x = 'Player', y = 'Avg Player Raid
Points', data=df_Pl
```

```
ayer_Raid_points)
plt.title("Avg Raid Points of Player in S7", fontsize
= 18)
plt.ylabel("Avg Player Raid Points", fontsize = 15)
plt.xlabel("Player",fontsize = 15)
ax=ax.set_xticklabels(ax.get_xticklabels(),
rotation=75)
df_Player_Tackle_points = pd.read_excel(file,'Tackle
Points')
df_Player_Tackle_points['Avg Player Tackle Points'] =
df_Player_Tackle_
points['Points']/df_Player_Tackle_points['Matches']
df_Player_Tackle_points =
df_Player_Tackle_points.drop(['Points','Match
es'],axis=1)
df_Player_Tackle_points =
df_Player_Tackle_points.sort_values(by='Avg P
layer Tackle Points',ascending=False)
plt.subplots(figsize = (12,8))
ax = sns.barplot(x = 'Player', y = 'Avg Player Tackle
Points', data=df_
Player_Tackle_points, edgecolor=(0,0,0),
linewidth=0.2)
plt.title("Avg Tackle Points of Player in S7",
fontsize = 18)
plt.ylabel("Avg Player Tackle Points", fontsize = 15)
plt.xlabel("Player",fontsize = 15)
ax=ax.set_xticklabels(ax.get_xticklabels(),
rotation=75)df_Player_Raid_points =
df_Player_Raid_points.sort_values(by='Successfu
l Raid %',ascending=False)
df_Player_Raid_points =
df_Player_Raid_points.reset_index(drop=True)
df_Player_Raid_points =
df_Player_Raid_points.loc[:5,:]
df_Player_Raid_points
df_Player_Tackle_points =
df_Player_Tackle_points.sort_values(by='Succe
ssful Tackle %',ascending=False)
df_Player_Tackle_points =
df_Player_Tackle_points.reset_index(drop=True
)
df_Player_Tackle_points =
df_Player_Tackle_points.loc[:10,:]
```

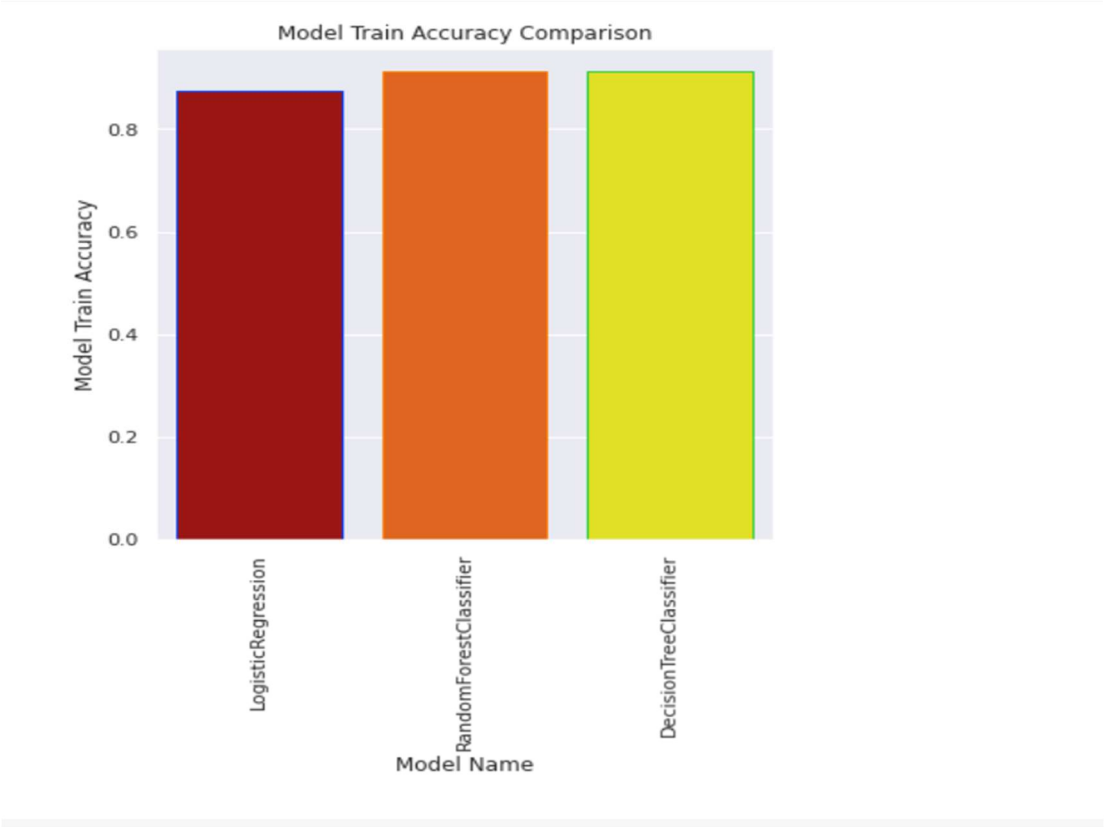
```
df_Player_Tackle_points
plt.subplots(figsize = (12,8))
ax = sns.barplot(x = 'Player', y = 'Successful Tackle
%', data=df_Playe
r_Tackle_points, edgecolor=(0,0,0), linewidth=0.2)
plt.title("Successful Tackle % of Player in S7",
fontsize = 18)
plt.ylabel("Successful Tackle %", fontsize = 15)
plt.xlabel("Player",fontsize = 15)
ax=ax.set_xticklabels(ax.get_xticklabels(),
rotation=75)
```

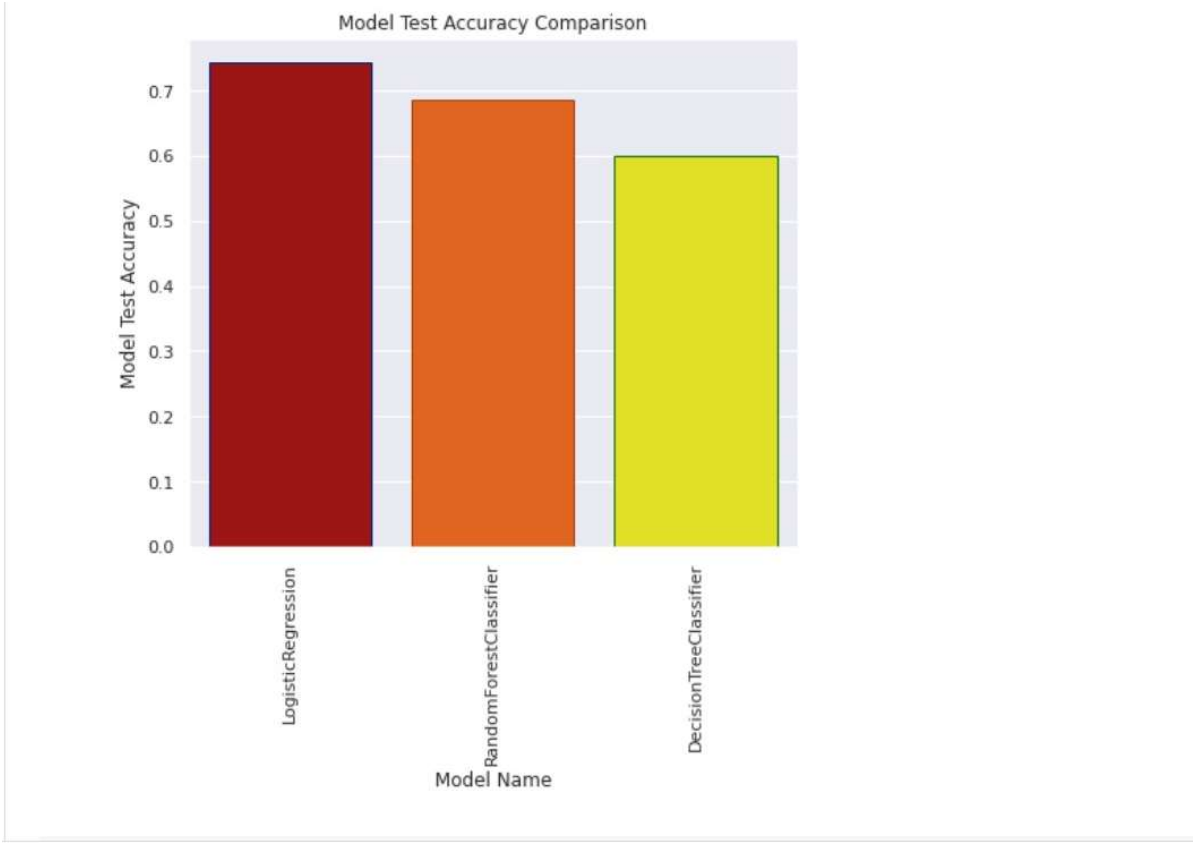
OUTPUT:

```
[ ]  #logistic Regression
     logreg = LogisticRegression()
     logreg.fit(X_train, y_train)
     Y_pred = logreg.predict(X_test)

     score = logreg.score(X_train, y_train)
     score2 = logreg.score(X_test, y_test)

     print("Training set accuracy: ", '%.3f'%(score))
     print("Test set accuracy: ", '%.3f'%(score2))

     Training set accuracy:  0.875
     Test set accuracy:  0.743
```

```
[69] #Random Forest Classifier
     rf = RandomForestClassifier(n_estimators=100, max_depth=2
                                 random_state=100)
     rf.fit(X_train, y_train)


     score = rf.score(X_train, y_train)
     score2 = rf.score(X_test, y_test)


     print("Training set accuracy: ", '%.3f'%(score))
     print("Test set accuracy: ", '%.3f'%(score2))
```

```
Training set accuracy:  0.912
Test set accuracy:  0.686
```




Model Test Accuracy Comparison
Model Test Accuracy
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
LogisticRegression
RandomForestClassifier
DecisionTreeClassifier
Model Name