nfldatabowl2024-jack-degesero

February 14, 2024

1 NFLDataBowl: Using Decision Tree Analysis to Forecast Tackling

1.0.1 Jack DeGesero

1.0.2 In my project, I utilize decision tree analysis to create a machine learning (ML) model to predict if a ball carrier will get tackled during a play. To prepare the data to train the model, I join the provided NFL Big Data Bowl datasets to indicate if a tackle occured on a play for each player, each play, every game, for weeks 1-9 of the 2022 NFL season. By using the equation for entropy, H(x) = -Σp(x)logbp(x), we can create metrics to assess which attributes (e.g. height) tell us the most information when it comes to classifying a tackle. The higher the information gain, the more relevant that attribute is in the outcome of a tackle. After assessing these metrics, we can then predict whether or not a tackle will occur with some degree of accuracy.

See code below.

1.1 Data Loading

1.1.1 The most preliminary step is to import all packages used for analysis and to load the data sets from their raw formats.

[1]:
```
# Import the data

import numpy as py #series
import pandas as pd #dataframes
import matplotlib.pyplot as plt #graphs
import seaborn as sns #analysis graphs

from sklearn import tree #for tree object
from sklearn import model_selection #for partition into test and training data
from sklearn import preprocessing #to change attributes
from sklearn import metrics #for checking model accuracy
```

[2]:
```
games = pd.read_csv('games.csv')
plays = pd.read_csv('plays.csv')
tackles = pd.read_csv('tackles.csv')
players = pd.read_csv('players.csv')
```

```
week1 = pd.read_csv('tracking_week_1.csv')
week2 = pd.read_csv('tracking_week_2.csv')
week3 = pd.read_csv('tracking_week_3.csv')
week4 = pd.read_csv('tracking_week_4.csv')
week5 = pd.read_csv('tracking_week_5.csv')
week6 = pd.read_csv('tracking_week_6.csv')
week7 = pd.read_csv('tracking_week_7.csv')
week8 = pd.read_csv('tracking_week_8.csv')
week9 = pd.read_csv('tracking_week_9.csv')
```

1.2 Data Preprocessing

1.2.1 Before performing ML, all of the data must be reorganized into one table which will be the model's input. This is done with merge operations. A merge will join two tables into one based on a key value unique to all entries in one table but not unique in the other. In our merged table, each row will show a players average tracking data (& other characteristics) for every player in each play, every game, for weeks 1-9 of the 2022 season.

1.2.2 Once the new table is created, many values still need to be removed (e.g. keys, which after merging, are no longer needed) and cleaned (e.g. height in feet & inches to just inches).

[3]:
```
#merge frames
df = pd.merge(games, plays, on=['gameId']).merge(
    tackles, on=['gameId', 'playId']).merge(
    players, on=['nflId'])

#get averages
avgs = pd.concat([week.groupby(['gameId', 'playId', 'nflId'])[['jerseyNumber',␣
↪'x', 'y', 's', 'a', 'dis', 'o']].agg('mean') for week in [week1, week2,␣
↪week3, week4, week5, week6, week7, week8, week9]])

df = pd.merge(df, avgs, on=['gameId', 'playId', 'nflId'], how='left').
↪sort_values(by=['gameId', 'playId']) #merge avgs w initial frame

#dimensionality reduction
df.
↪drop(columns=['season','homeFinalScore','visitorFinalScore','ballCarrierDisplayName',
             ␣
↪'penaltyYards','prePenaltyPlayResult','playResult','playNullifiedByPenalty',
             ␣
↪'expectedPoints','expectedPointsAdded','foulName1','foulName2','foulNFLId1',␣
↪'pff_missedTackle',
              'foulNFLId2','displayName', 'playDescription', 'jerseyNumber',␣
↪'assist', 'forcedFumble',
```

```python
                 'gameId', 'playId', 'nflId'], inplace=True)

#other cleaning operations

df['birthDate'] = pd.to_datetime(df['birthDate'], errors='coerce').apply(lambda x: int(x.timestamp()) if not pd.isnull(x) else None)#convert birthdate to UNIX
df['birthDate'].fillna(df['birthDate'].mean(), inplace=True) #replace missing bdates w average bdate of players

df['gameClock'] = df['gameClock'].apply(lambda x: int(x.split(':')[0]) * 60 + int(x.split(':')[1])) #convert game clock to seconds
df['gameDate'] = pd.to_datetime(df['gameDate'], errors='coerce').apply(lambda x: int(x.timestamp()) if not pd.isnull(x) else None) #convert date to UNIX
df['gameTimeEastern'] = pd.to_datetime(df['gameTimeEastern'], format='%H:%M:%S').dt.strftime('%H%M') #make gametime into HHMM
df['height'] = df['height'].str.split('-').apply(lambda x: int(x[0]) * 12 + int(x[1])) #convert feet&inches to inches

#converting categories to integers
df['position'] = preprocessing.LabelEncoder().fit_transform(df['position'])
df['collegeName'] = preprocessing.LabelEncoder().fit_transform(df['collegeName'])
df['passResult'] = preprocessing.LabelEncoder().fit_transform(df['passResult'])
df['offenseFormation'] = preprocessing.LabelEncoder().fit_transform(df['offenseFormation'])

df['homeTeamAbbr'] = preprocessing.LabelEncoder().fit_transform(df['homeTeamAbbr'])
df['visitorTeamAbbr'] = preprocessing.LabelEncoder().fit_transform(df['visitorTeamAbbr'])
df['possessionTeam'] = preprocessing.LabelEncoder().fit_transform(df['possessionTeam'])
df['defensiveTeam'] = preprocessing.LabelEncoder().fit_transform(df['defensiveTeam'])
df['yardlineSide'] = preprocessing.LabelEncoder().fit_transform(df['yardlineSide'])

#remove all rows with any missing values
df.dropna(inplace=True)
```

[4]:
```python
df
```

[4]:
```
     week    gameDate gameTimeEastern  homeTeamAbbr  visitorTeamAbbr  \
164     1  1662595200            2020            16                3
303     1  1662595200            2020            16                3
```

```
667        1  1662595200           2020          16                3
302        1  1662595200           2020          16                3
699        1  1662595200           2020          16                3
…     …        …            …           …             …
7825       9  1667779200           2015          22                2
14743      9  1667779200           2015          22                2
14801      9  1667779200           2015          22                2
14744      9  1667779200           2015          22                2
14741      9  1667779200           2015          22                2

       ballCarrierId  quarter  down  yardsToGo  possessionTeam  …  weight  \
164            42489        1     1         10               3  …     208
303            47857        1     2          3               3  …     242
667            42489        1     2          8               3  …     240
302            52494        1     2          9               3  …     242
699            44881        1     3          8              16  …     191
…                …      …    …         …              …   …      …
7825           46160        4     4          3              22  …     220
14743          46160        4     2         10              22  …     197
14801          46160        4     2         10              22  …     180
14744          44879        4     3          1              22  …     197
14741          52942        4     2         10              22  …     197

          birthDate  collegeName  position          x          y         s  \
164    7.829568e+08           40         0  78.350000  36.037727  3.486818
303    6.464448e+08          154         5  61.442727  42.175152  5.228788
667    7.159104e+08           44         2  49.292273  27.878182  4.765455
302    6.464448e+08          154         5  36.600625  47.548750  7.376250
699    6.725376e+08          109         9  53.887500   3.320000  7.004167
…               …            …       …          …         …        …
7825   7.971847e+08          103         4  40.103333  47.460952  5.730952
14743  7.265376e+08          161         0  48.577105  14.097105  2.002895
14801  7.971847e+08           51         0  48.955526  14.515526  1.919474
14744  7.265376e+08          161         0  66.170889   4.755111  1.579556
14741  7.265376e+08          161         0  82.595085   1.008136  1.483729

              a       dis           o
164    2.860455  0.356364  130.904091
303    3.008182  0.526364  115.292727
667    3.930455  0.477273  198.322273
302    2.981250  0.745000  311.054375
699    2.733333  0.715000  168.189167
…           …         …          …
7825   3.045714  0.578095  205.558571
14743  2.089737  0.201842  137.226842
14801  2.111053  0.189737  138.949737
14744  1.920222  0.157111  220.999333
```

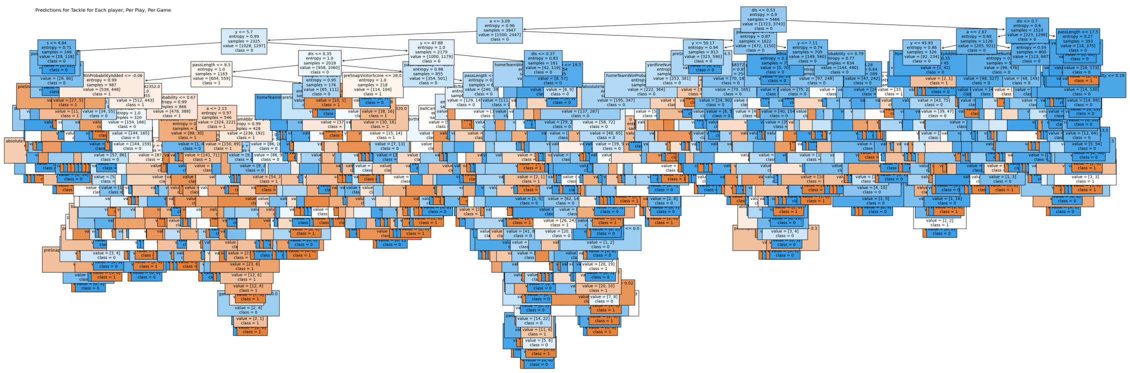
```
14741  1.628305  0.150169  108.929322

[6833 rows x 38 columns]
```

1.3 Training

1.3.1 The decision tree model can only be built and trained after it is cleaned. After cleaning, 80% of the new rows will be used for the model's construction and training, the remaining 20% of rows will be used later to test the model's accuracy. In a decision tree, entropy evaluates the alignment of data rows with the target class (tackling), aiding in determining the best classification path. After visualizing all possible classification paths based on the observed characteristics, a tree-like data structure is created.

[5]:

```
allAtr = df[list(df.drop(columns=['tackle']).columns.values)] #all attributes␣
↪except class attribute

classAtr = df['tackle'] #class attribute

#Partition 80%/20%
xtr, xt, ytr, yt = model_selection.train_test_split(allAtr, classAtr,␣
↪test_size=0.2, random_state=42, shuffle=False)

#instantiates tree object
AllTree = tree.DecisionTreeClassifier(criterion="entropy", random_state=42)

#make the tree
AllTree.fit(xtr, ytr)

#plot the tree
plt.figure(figsize=(48, 16))
tree.plot_tree(AllTree, feature_names=allAtr.columns, class_names=list(map(str,␣
↪df['tackle'].unique())), filled=True, impurity=True, precision=2,␣
↪fontsize=10)
plt.title('Predictions for Tackle for Each player, Per Play, Per Game',␣
↪loc='left')
plt.show()
```



1.4 Analysis

1.4.1 Based on the paths represented in the graph above, the model will predict whether or not a tackle occurred in the test data by examining all other features of the players, the play, and the game without being told if a tackle happened. The predictions are then compared to the observed historical data. Metrics of these results will tell us the accuracy of the predictions. The accuracy rating is ultimately derived from the importance values we previously obtained by using the equation for entropy.

[6]:
```python
#predict test data based on training data
prediction = AllTree.predict(xt)

#get accuracy
accuracy = metrics.accuracy_score(yt, prediction)
print("Accuracy of Model:", accuracy)

#get gain of each attribute examined, put in list
importances = AllTree.feature_importances_

print("Gain:")

#list of tuples with attribute name and importance, sort the list
feature_importance_list = [(i, importances[xtr.columns.get_loc(i)]) for i in␣
↪xtr.columns]
feature_importance_list.sort(key=lambda x: x[1], reverse=True)

#print results
for feature, importance in feature_importance_list:
    print(f'Attribute: {feature}, Importance: {importance:.5f}')
```

```
Accuracy of Model: 0.6159473299195318
Gain:
Attribute: dis, Importance: 0.09051
Attribute: y, Importance: 0.06234
```

```
Attribute: a, Importance: 0.05453
Attribute: birthDate, Importance: 0.04539
Attribute: x, Importance: 0.04177
Attribute: passLength, Importance: 0.04052
Attribute: passProbability, Importance: 0.03996
Attribute: o, Importance: 0.03885
Attribute: ballCarrierId, Importance: 0.03706
Attribute: s, Importance: 0.03560
Attribute: weight, Importance: 0.03257
Attribute: collegeName, Importance: 0.03181
Attribute: gameClock, Importance: 0.03068
Attribute: absoluteYardlineNumber, Importance: 0.02997
Attribute: visitorTeamAbbr, Importance: 0.02982
Attribute: preSnapVisitorScore, Importance: 0.02911
Attribute: yardlineNumber, Importance: 0.02785
Attribute: homeTeamWinProbabilityAdded, Importance: 0.02782
Attribute: gameDate, Importance: 0.02257
Attribute: visitorTeamWinProbilityAdded, Importance: 0.02191
Attribute: yardlineSide, Importance: 0.02037
Attribute: homeTeamAbbr, Importance: 0.02001
Attribute: defensiveTeam, Importance: 0.01937
Attribute: possessionTeam, Importance: 0.01832
Attribute: preSnapVisitorTeamWinProbability, Importance: 0.01806
Attribute: yardsToGo, Importance: 0.01802
Attribute: preSnapHomeScore, Importance: 0.01747
Attribute: preSnapHomeTeamWinProbability, Importance: 0.01700
Attribute: defendersInTheBox, Importance: 0.01671
Attribute: height, Importance: 0.01617
Attribute: gameTimeEastern, Importance: 0.01305
Attribute: position, Importance: 0.00835
Attribute: quarter, Importance: 0.00801
Attribute: offenseFormation, Importance: 0.00673
Attribute: week, Importance: 0.00598
Attribute: down, Importance: 0.00574
Attribute: passResult, Importance: 0.00000
```

1.5 Conclusion

1.5.1 After utilizing decision tree analysis, we are able to predict if a ball carrier will be tackled with 61.5% accuracy. The model also gives us insight into the most relevant features in predicting the tackle. The three most relevant being average distance traveled, average acceleration on the field, and average y-. It is important to consider metrics like these since they give unique insights into not readily apparent patterns observed from historical data.