Supervised Machine Learning Using Decision Tree And Artificial Neural Network To Predict The Results Of College Football Matches

October 2023

Authors:

Antoine Burgaud

Axinte Octavian-Constantin

Contents

1. Overview

Introduction covers the topic of the paper.

Problem formulation contains the dataset description, features and labels selection and remarks about steps in the implementation.

Methods includes the ML methods used and why are they useful for the topic.

Preprocessing explains all the data manipulation needed for applying the methods.

Coding Parameters refers to the reasoning of code choices.

The results include the accuracy and the best method for this dataset.

Conclusion includes final remarks and appendix the code.

2. Introduction

College football is one of the most attended events in USA filling up to 100.000 supporters on a single match. While the beauty and unpredictability of these games have their old charm, the trend nowadays is to forecast match results using machine learning. For sport analytics two classification methods gained notoriety for their ability to handle classification tasks effectively: Decision Tree and Artificial Neural Network (ANN) (Bunker, 2022) The purpose of this paper is to answer the question every fan asks himself: "Can I accurately anticipate the results of a football game?" By examining historical game data, team statistics, time of the day and other relevant features, the aim is to use possibilities and probabilities as an advantage.

3. Problem formulation

The dataset used contains data collected for 18 years (2000-2018) of 63 teams from all the Division 1 football conferences. The dataset was obtained from Kaggle and can be downloaded from this link. It is mentioned that a part of the data is collected from Wikipedia so it may not be the most reliable. It contains 6673 datapoints (each game represents a datapoint) and 25 columns. Dropped columns include: "Date", "Site", "Tailgating", "Conference" as they offer no useful information for determining match results or have redundant information present in other columns. The label is denoted by the column "Result," indicating whether predictions are accurate; it is transformed in a binary data type. The dataset now comprises 20 features and 1 label, as detailed in the table below.

Feature	Data Type	Meaning
Team	String – categorical	The home team playing the match
Opponent	String – categorical	The opposing team
Rank	Integer – discrete	The ranking of the home team (if applicable)
Opponent Rank	Integer – discrete	The ranking of the opponent team (if applicable)
Attendance	Integer – discrete	The number of spectators at the match
Current Wins	Integer – discrete	The home team's current wins in the respective season.
Current Losses	Integer – discrete	The home team's current losses in the respective season.
Stadium Capacity	Integer – discrete	The capacity of the stadium where the match took place
Year	Integer – discrete	The year of the match
Month	Integer – discrete	The month of the match
Day	Integer – discrete	The day of the match
Time	Int – discrete	The time of the match
Fill Rate	Float – continuous	The percentage of stadium capacity filled
PRCP	Float – continuous	Precipitation (rain) in inches

SNOW	Float – continuous	Snowfall in inches
SNWD	Float – continuous	Snow depth in inches
TMAX	Integer – discrete	Maximum temperature on the match day
TMIN	Integer – discrete	Minimum temperature on the match day
TV	Bool – binary	Television broadcast Yes/No
New Coach	Bool – binary	New coach in the home team Yes/No

*Rank – Teams are ranked from 1 to 25. The rest are assigned “NR”.

*Opponent Rank – The same as for rank.

*TV - Transform it into a Boolean since it has no relevance on what channel it was broadcasted and there are hundreds of them.

4. Methods. Model motivation

We chose a Decision Tree algorithm for various reasons, first we have some categorical and continuous data, and Decision Tree deals well with these 2 types of data. Also, it is robust to outliers and the author mentioned in the description of the dataset that the data is sometimes not reliable. “The appeal of this algorithm is obvious in that it is fast to train and usually does not possess a cumbersome number of tuneable parameters. Importantly, they do not generate black-box models, but instead embody varying degrees of interpretability, depending on their implementation.” (Bunker, 2022)

ANN related algorithms are the most used method for predicting sport results. We implemented a model due to its ability to involve non-linear, complex relationships between various features. Also, ANN is highly adaptable, which is very useful in the sport world when only a single player transfer can strongly influence a team.

5. Preprocessing

Rank ‘Not Ranked’ datapoints: Rank is the ranking among the top 25 teams. We will be using Random Forest to solve this problem, even if it deals well with categorical data, we will set the ‘NR’ Rank datapoints to 26 since we will gain a lot of information for this feature if it is discrete instead of categorical.

TV: There are a hundred different TV channels so it will be easier for the model if it is transformed into a binary feature.

Time: Transformed the string Time into an integer so that it becomes discrete: information is more valuable for the decision trees.

Normalizing Data: Random Forest does not need to normalize the data.

Team, Opponent: Without any modifications, even if it is the same name, the algorithm does not recognise that the team feature and opponent feature mention about the same team. So, the method get_dummies() is used and the result are combined in such a way that the algorithm knows it is the same team playing at home or visitors.

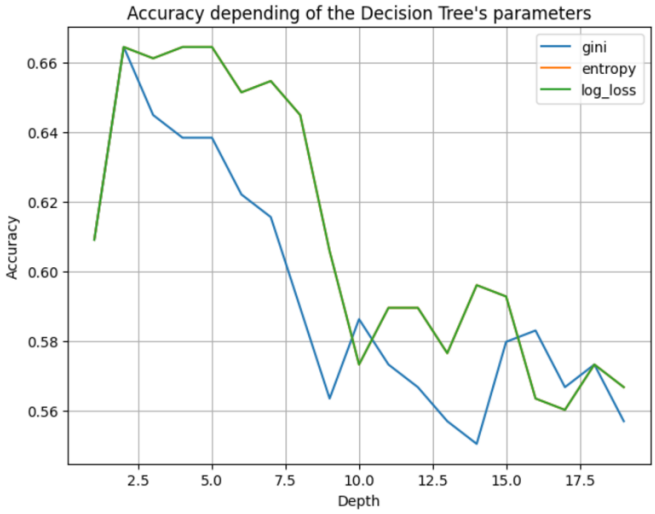
6. Coding Parameters

6.1. Training, Testing and Validation Sets

Since it is quite a large dataset, we chose to divide it in a 70-30 split. 70% of the dataset is used for training, whereas 30% of the dataset is used for validation. It would not be recommended to choose a smaller testing set because of overfitting problems. Also, optimal model training requires a sizable training set to effectively learn patterns. To have a real accuracy value there should be a large enough testing ground. (A Guide to Data Splitting in Machine Learning, n.d.)

Additionally, the validation set is employed in the cross-validation algorithm to determine optimal tuning parameters, maintaining a 1:3 ratio similar to the training and testing sets.

6.2. Loss function

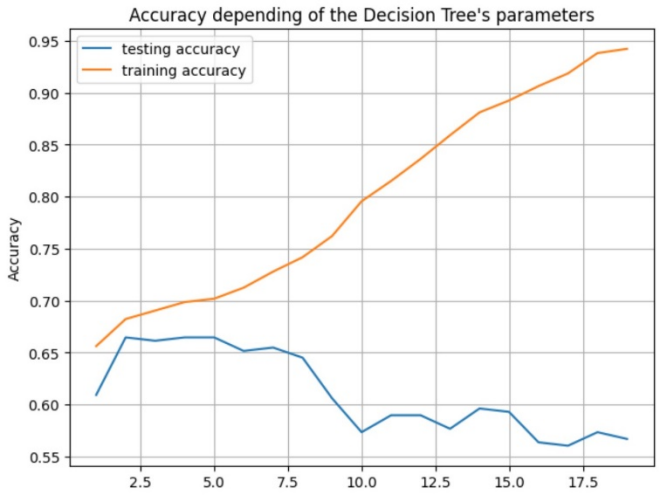


For decision trees, a comparison of three loss functions was conducted based on tree depth. Results indicate that log_loss and entropy yield superior accuracy, with both functions overlapping.

Consequently, log_loss is the chosen option.

Regarding the Neural Network, the utilization of log_loss is mandatory in scikit-learn due to the model's construction.

6.3. Training and Validation Errors. Depth for Decision Tree



The figure to the left shows the dependency between the accuracy and depth for the testing and training sets. Having a greater depth apparently will generate a better accuracy if only the training set is considered. This is the result of overfitting. Finding the best combination for certain data but worse accuracy for any other dataset. The depth of 5 should be chosen due to the maximum value in depth for the testing set.

To find the other best parameters for the decision tree, a randomized search by cross validation has been performed according to (Will Koehrsen, 2018). By looking over all the possible combinations, the best tuning is:

'splitter': 'best',

'min_samples_split': 2,

'min_samples_leaf': 2,

'max_features': 'sqrt',

'max_depth': 5,

criterion': 'log_loss'

These parameters will be used to compute the test error.

The ANN can overfit if there are too many training steps and if the architecture is too complex. This overfit will create a difference between training and validation set accuracy. The better set of parameters found for this model is :

number of iterations : 2000

hidden layers : (20,)

learning_rate : 1e-5

These parameters avoid overfitting, which can verified by looking at the training and validation errors :

Training accuracy : 0.58

Testing accuracy : 0.57

7. Results

The accuracy for ANN method is 57% and for Decision Tree is 67% so the method preferred for this dataset is the Decision Tree.

8. Conclusion

In summary, this paper uses machine learning to predict college football game result. It goes through the challenges of handling various types of data. The ML methods used are Decision Tree and Artificial Neural Networks (ANN), and together with dataset modifications and model evaluation strategies contribute to enhancing the accuracy of match result predictions in the field of college football.

9. References

A Guide to Data Splitting in Machine Learning. (n.d.). Retrieved from https://medium.com/@datasciencewizards/a-guide-to-data-splitting-in-machine-learning-49a959c95fa1

Bunker, R. &. (2022). The Application of Machine Learning Techniques for Predicting Match Results in Team Sport: A Review. *Journal of Artificial Intelligence Research*.

Will Koehrsen. (2018). *Hyperparameter Tuning the Random Forest in Python*. Retrieved from towardsdatascience: https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Kaggle dataset link: https://www.kaggle.com/datasets/jeffgallini/college-football-attendance-2000-to-2018

10. Appendix

main

October 12, 2023

[175]:
```
import numpy as np
import matplotlib.pyplot as plt
import math as m
import pandas as pd
import seaborn as sns
import pydotplus
import os

import sklearn.metrics as metrics
from sklearn.tree import export_text, plot_tree, DecisionTreeClassifier,␣
 ↪export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.neural_network import MLPClassifier
```

[127]:
```
#ENV SETTINGS VARIABLES
_PRINT = False
_SHOW = False
_SAVE_IMAGE = False
_SAVE_CSV = True
```

[82]:
```
#FUNCTION
def save(data : pd.DataFrame,name : str):
    if _SAVE_CSV : data.to_csv(os.getcwd() + '/' + name + '.csv')
```

[129]:
```
#IMPORT & CLEAN DATA
data = pd.DataFrame(pd.read_csv('CFBeattendance.csv', encoding='cp1252'))

if _PRINT : print('data.head(10)',data.head(10))

#dropping data
columnsDrop = ['Date','Site','Tailgating','Conference']
data.drop(columnsDrop, axis=1, inplace=True)
if _PRINT : print('data.dtypes',data.dtypes)
```

```
#Cleaning Team and Opponent Column
data.Team = data.Team.apply(lambda x : x.replace('*',' '))
data.Team = data.Team.apply(lambda x : x[6:].lstrip() if 'No.' in x else x)
data.Opponent = data.Opponent.apply(lambda x : x.replace('*',' '))
data.Opponent = data.Opponent.apply(lambda x : x[6:].lstrip() if 'No.' in x␣
 ↪else x)
homeTeams = data.groupby('Team').count().
 ↪sort_values(by='Opponent',ascending=False)['Opponent'].index
opponentTeams = data.groupby('Opponent').count().
 ↪sort_values(by='Team',ascending=False)['Team']


# How many match are they when an opponentTeam can also be a homeTeam in the␣
 ↪dataset
sum = 0
for homeTeam in homeTeams :
    if homeTeam not in opponentTeams.index:
        if _PRINT : print(homeTeam,'is not an Opponent team')
    else :
        sum+=opponentTeams[homeTeam]
if _PRINT : print(sum,' games where the opponent is also a homeTeam in the␣
 ↪dataset \n')

# Only keep these games
data = data[data.Opponent.apply(lambda x : x in data.Team.values)]
data = data[data.Team.apply(lambda x : x in data.Opponent.values)]

#get dummies for the team name
team_dummies  = pd.get_dummies(data.Team,dtype=int)*2 + pd.get_dummies(data.
 ↪Opponent,dtype=int)
data[team_dummies.columns] = team_dummies
save(data,'data')

#1.0.2 Rank 'NR'
#Non Ranked team become 26th
data.loc[:,'Rank'] = data.Rank.apply(lambda x : int(26) if x=='NR' else int(x))
data.loc[:,'Opponent_Rank'] = data.Opponent_Rank.apply(lambda x : int(26) if␣
 ↪x=='NR' else int(x))

# Tranform TV channels into boolean
data.loc[:,'TV'] = data['TV'].apply(lambda x : int(x=='Not on TV'))

#Int new_coach
data.loc[:,'New Coach'] = data['New Coach'].apply(int)
```

```
#Transform Time as int
if _PRINT : print('data.Time.str.contains(\'PM\')',data.Time.str.contains('PM'))
time = data.Time.str.replace(':','')
afternoonshift = data.Time.apply(lambda x : 1200 if 'PM' in x else 0)
data.Time = time.apply(lambda x : int(x.split()[0]))
data.Time = data.Time + afternoonshift

if _PRINT : print('data.Result.apply(lambda x : x.split(\' \')[0]).
 ↪unique()',data.Result.apply(lambda x : x.split(' ')[0]).unique())
if _PRINT : print('Find the \'NC\' case')
if _PRINT : print('--data[data.Result.str.contains(\'NC\')]',data[data.Result.
 ↪str.contains('NC')])

# Resolve the 'NC' case and Transform into boolean
data.Result = data.Result.str.replace('NC','L').apply(lambda x : x.split('␣
 ↪')[0]=='W')
data.loc[:,'Result'] = data['Result'].apply(int)

data = data.drop(columns=['Team','Opponent'])
save(data,'data')

if _PRINT : print(data)

if _SHOW :
    sns.countplot(x = "Result", data = data)
    plt.title("Result")
    plt.show()

    sns.countplot(x = "Team", data = data, hue = "Result")
    plt.xticks(rotation = 45)
    plt.title("Teams")
    plt.show()

y = data['Result'] # encode edibility as 1 or 0

X = data.drop(columns = ["Result"])
save(X,'test')

if _PRINT : print(X.columns)
if _PRINT : print(X.head())
#X = pd.get_dummies(X,dtype = int)
if _PRINT : print(X.columns)
if _PRINT : print(X.head())
if _PRINT : print('X.shape ',X.shape)
if _PRINT : print(X.head())
```

```
save(X,'final')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)#,␣
 ↪random_state=2)

print('X_train.shape ',X_train.shape)
print('X_test.shape ',X_test.shape)
save(X,'final')
```

```
X_train.shape  (1227, 79)
X_test.shape  (307, 79)
```

[293]:
```
#TRAIN AND TEST

clfs = [DecisionTreeClassifier(random_state=0, max_depth=2),
        DecisionTreeClassifier(random_state=0, max_depth=3),
        DecisionTreeClassifier(random_state=0, max_depth=4),
        DecisionTreeClassifier(random_state=0, max_depth=5),
        DecisionTreeClassifier(random_state=0, max_depth=30),
        RandomForestClassifier(criterion='gini'),
        RandomForestClassifier(criterion='entropy'),
        RandomForestClassifier(criterion='log_loss')
]

clfs_names = ['DT depth2',
              'DT depth3',
              'DT depth4',
              'DT depth5',
              'DT depth30',
              'RF gini',
              'RF entropy',
              'RF log_loss']

clfs = [DecisionTreeClassifier(random_state=0, max_depth=3)]
clfs_names = ['DT depth3']

def train_test_acc(clf,training = False):
    model = clf.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_training = model.predict(X_train)
    if training : return accuracy_score(y_train,y_pred_training)
    return accuracy_score(y_test, y_pred)

for clf,clfs_name in zip(clfs,clfs_names):
    model = clf.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

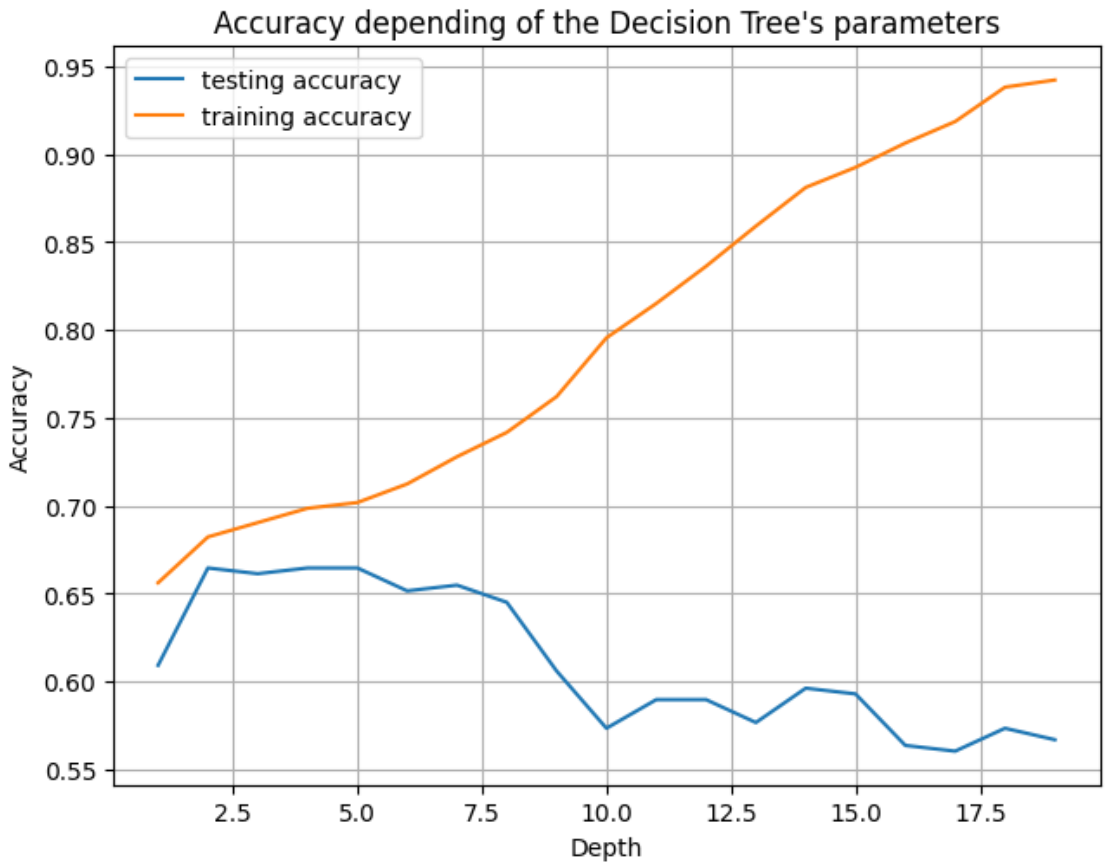
```
    acc = accuracy_score(y_test, y_pred)
    acc = train_test_acc(clf)
    confmat = confusion_matrix(y_test, y_pred)
    print(clfs_name)
    print("Accuracy:", acc)

    # plot the confusion matrix
    if _PRINT:
        ax = plt.subplot()
        sns.heatmap(confmat,annot=True, fmt='g', ax=ax)
        ax.set_xlabel('Predicted labels',fontsize=15)
        ax.set_ylabel('True labels',fontsize=15)
        plt.show()

        plot_tree(model, feature_names = list(X.columns),  filled = True,␣
↪class_names=["lose", "win"])
        plt.show()
    # save as pdf for a high res image:
    if _SAVE_IMAGE :
        d_tree = export_graphviz(clf, feature_names = list(X.columns), filled =␣
↪True, class_names=["lose", "win"])
        pydot_graph = pydotplus.graph_from_dot_data(d_tree)
        pydot_graph.write_pdf(os.getcwd() + '/' + 'football_tree.pdf')
```

```
DT depth3
Accuracy: 0.6449511400651465
```

[317]:
```
depth = range(1,20)
criterias = ['entropy']
random_state = 42
accs = np.
↪array([[train_test_acc(DecisionTreeClassifier(max_depth=i,criterion=criterion,random_state=
↪for i in depth] for criterion in criterias])
accs_training = np.
↪array([[train_test_acc(DecisionTreeClassifier(max_depth=i,criterion=criterion,random_state=
↪for i in depth] for criterion in criterias])
for acc,acc_training in zip(accs,accs_training) :
    plt.plot(depth,acc)
    plt.plot(depth,acc_training)
plt.tight_layout()
plt.title('Accuracy depending of the Decision Tree\'s parameters')
plt.xlabel('Depth')
plt.ylabel('Accuracy')
plt.legend(('testing accuracy','training accuracy'))
plt.grid()
plt.show()
```



[ ]:
```
criterion = ['log_loss']
splitter = ['best', 'random']
max_depth = [2,3,4,5,6,7,8]
min_samples_split = [2, 5, 10, 15, 20]
min_samples_leaf = [1, 2, 3, 4]
max_features = ['sqrt', 'log2']
# Create the random grid
random_grid = {'criterion': criterion,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'splitter':splitter}

rf = DecisionTreeClassifier()
rf_random = RandomizedSearchCV(estimator = rf,
                               param_distributions = random_grid,
                               n_iter = 200,
                               cv = 3,
```

```
                                verbose=2,
                                random_state=42,
                                n_jobs = -1)
rf_random.fit(X_train, y_train)
```

[319]:
```
print(train_test_acc(DecisionTreeClassifier(**rf_random.best_params_)))
rf_random.best_params_
```

```
0.6254071661237784
```

[319]:
```
{'splitter': 'best',
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 5,
 'criterion': 'log_loss'}
```

[ ]:
```
mlp = MLPClassifier(solver='adam',
                    alpha=1e-4,
                    max_iter = 2000,
                    hidden_layer_sizes=(20,),
                    verbose=True,
                    learning_rate_init=1e-5,
                    learning_rate='adaptive',
                    n_iter_no_change = 10000)
mlp.fit(X_train,y_train)
y_pred = mlp.predict(X_test)
print(accuracy_score(y_test, y_pred))

plt.plot(mlp.loss_curve_)

plt.title("Loss Curve", fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```

[303]:
```
print(accuracy_score(mlp.predict(X_train),y_train))
accuracy_score(mlp.predict(X_test),y_test)
```

```
0.5778321108394459
```

[303]:
```
0.5700325732899023
```

[ ]:
```
#FINAL TEST ERROR
```