Predicting finalists in PGA Tour's FedEx Cup playoffs with different classification methods

March 2022

1. Introduction

In the world of golf, it is often debated which part of the game is the most important: putting, off the tee shots, or something else. Some say that off the tee shots are inconsequental in the light of other parts of the game, while others state that one could win tournaments with a perfect off the tee shot. Therefore, it is natural to wonder whether it could be possible to use off the tee statistics to predict the winners on the highest level of golf, the PGA Tour.

The PGA Tour is one of the most recognized golf tours in the world. The PGA Tour is divided into two phases, namely the regular season and the FedEx Cup playoffs. The regular season of the PGA Tour consists of a number of tournaments where the players compete to earn FedEx Cup points. After the regular season, the top 125 players who have earned the most points will participate in the FedEx Cup playoffs. Furthermore, the top 30 points leaders of the playoffs will get to participate in the last phase of the playoffs, the Tour Championship, where the champion of the whole PGA Tour is crowned. [1]

This report consists of four parts in addition to this introduction. In the first part, the *Problem Formulation*, I will present the machine learning problem of predicting the results of PGA Tour based on the off the tee statistics. In the second part, the *Methods*, I will explain the used models more comprehensively. In the third part, the *Results*, I will present the results of our models and compare them to choose the best model for our problem. In the last part, the *Conclusions*, there is some discussion about the solution for the problem and whether the prediction was successful or not.

2. Problem Formulation

In this paper I am going to assess whether it is possible to predict the golf players who make it to the Tour Championship based on their off the tee statistics, e.g. driving statistics, during the regular season of the PGA Tour. In other words, I am going to predict the players who end up in the top 30 points leaders on the PGA Tour out of the 125 players who compete in the FedEx Cup playoffs during season 2021.

As data points we have the top 125 players (one data point = one player) in the PGA Tour from six seasons, 2016-2021. Therefore, in total we have the statistics of 750 players in our dataset. These players and their regular season statistics were collected from the official website of the PGA Tour [2]. The data was collected by using web scraping methods and the code to conduct this can be found in the appendix of this paper. As initial features we have the average driving distance of the player, the driving accuracy

of the player, the club head speed, the ball speed, and the spin rate of the ball. All of these features are explained more comprehensively in Table 1.

In this machine learning problem we are interested in which of the players in the playoffs ended up in the top 30 points leaders, in other words were the finalists during each season. Therefore, as labels we have the set of values {1, -1}, where 1 means that the player finished the playoffs in the top 30 (rankings 1-30) and -1 that the player did not (rankings 31-125).

Table 1. The features of the machine learning problem.

Name of the Feature	Description	Unit	Type
Driving Distance	The average distance travelled by the ball from the player's tee shots.	Yards	Numerical
Driving Accuracy	The percentage of time the player's tee shot ends up in the fairway.	Percentage	Numerical
Club Head Speed	The speed at which the player's club impacts the ball on Par 4 and Par 5 tee shots.	Miles per Hour	Numerical
Ball Speed	The peak speed of the ball at launch from the player's Par 4 and Par 5 tee shots.	Miles per Hour	Numerical
Spin Rate	The spin rate of the golf ball immediately after leaving the club on Par 4 and Par 5 tee shots.	Revolutions Per Minute	Numerical

3. Methods

Missing Values and Description of the Data

In the gathered dataset there were some missing values as 11 different players did not have any feature values during different seasons. As these values would be quite hard to manually find out and fill, I decided to just drop all these 11 rows from the dataset. Therefore, the final size of the dataset was 739 rows (data points) and 9 columns of which 5 are potential features and the "Eligible" column contains the labels {1, -1}. For training and validation we use the data points from seasons 2016-2020 and for testing the data points from season 2021. The final size of the training and validation data set is therefore 615 data points, and the final size of the testing data set is 124 data points.

Data Augmentation and Normalization

In addition to missing values, the classes of the dataset were heavily skewed as there were 179 data points labeled as 1 and 560 data points labeled as -1. We have therefore over three times more data points in the other class (negative class). This could lead to a situation where the classifier algorithm classifies every data point to the dominating class, thus leading in our case to a relatively high accuracy but small (or zero) precision and recall.

To handle this imbalance in our data, I decided to use data augmentation techniques. I balanced the data by adding three perturbations to each feature of class 1 data points of the training data set as the other class is approximately three times larger. The perturbations were obtained from the standard normal distribution (3.1). This way the training data set should be approximately balanced in every validation and testing split. The implementation in Python of this augmentation can be seen in the appendix.

$$\varepsilon \sim N(0,\ 1) \qquad (3.1)$$

In addition to data augmentation, I standardized the values of the features in the training set and applied this standardization (e.g., same parameters) also to the values in the validation and testing set. I used the standardization formula shown in 3.2, where μ is the mean and σ the standard deviation of the data.

$$z = \frac{x - \mu}{\sigma} \qquad (3.2)$$

Feature Selection

The features were chosen as I wanted to try to predict the PGA results using the off the tee shot statistics. As I had only five initial features, I decided not to reduce the number of these features as the model didn't seem to suffer from complex feature space. However, there can be seen a slight multicollinearity between some of the features and therefore Principle Component Analysis (PCA) might prove to be useful if one wants to improve the model performance even further.

Model Validation

To validate the machine learning models I used k-fold cross validation as with this validation method we can obtain more reliable estimates of the performance of the model. Within each fold, the data preprocessing was conducted as this way we could avoid using training data in the validation phase (e.g. data augmentation samples) and therefore overfitting. In the cross validation, I used 5 folds (a train size of 0.8), as we do not have a lot of data for training and validation.

Model 1: Logistic Regression

As the first machine learning method to predict the finalists I used logistic regression. Logistic regression is a binary classification algorithm that uses the linear hypothesis space to predict the binary labels (3.3). The labels are obtained by comparing the hypothesis to a threshold. I chose this method as the first method because our problem is a binary classification problem where logistic regression is a simple yet efficient solution. As the loss function the logistic regression uses logistic loss, which can be minimized by efficient gradient-based methods [3]. This makes it a computationally feasible loss function for our purposes.

$$h(x) = w^T x,\ w \in R^n \qquad (3.3)$$

To implement the model in Python I used the LogisticRegression class from the scikit-learn-package [4]. The code to implement this can be found in the appendix. As the metrics for the validation error and to assess the model performance I used 0-1 loss (accuracy score) and f1-score.

Model 2: Decision Tree

As the second machine learning method I used a decision tree classifier. A decision tree is a non-parametric method used in classification (and regression) that consists of different nodes that represent different decision rules on the features. The hypothesis space of the decision tree is the set of all hypotheses that can be formed with the tree [3]. As the loss function I decided to use Gini impurity (3.4) as it is the default option in the Python class I used to implement the model [5]. In each node the parameters are selected that minimizes this loss function.

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk}) \qquad (3.4)$$

I chose this method because the decision tree is different from logistic regression and other methods that use a linear hypothesis space. Therefore, a decision tree could detect non-linear relationships in the data better than other linear classification methods. The disadvantage of this method is that it is prone to overfit as an uncontrolled decision tree can eventually grow so deep that it will make decisions based on single data points. To reduce this risk of overfitting I added some constraints to the maximum depth the decision tree is allowed to grow. From Figure 1 we can see that the best average validation accuracy of the model is obtained using the maximum depth of 4.

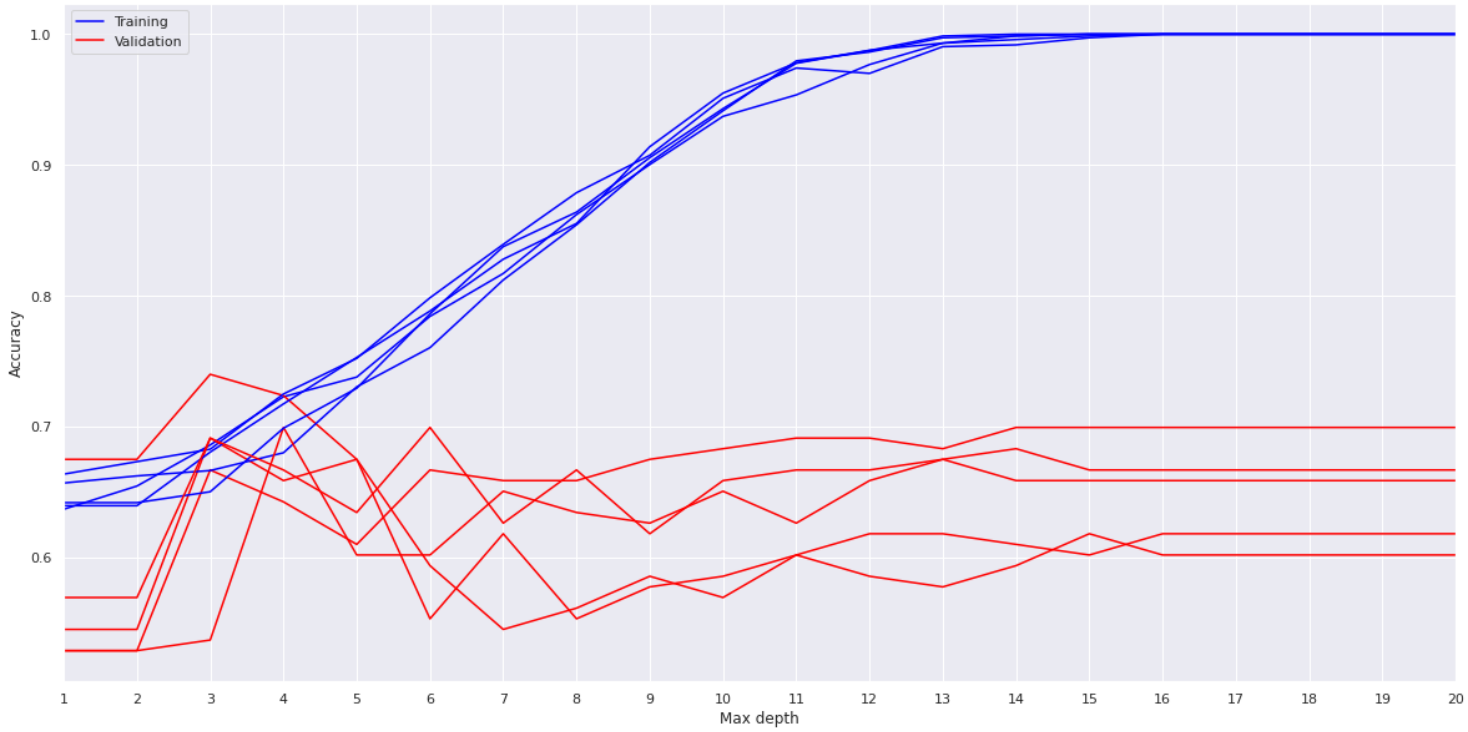

Figure 1. The accuracy of the Decision Tree using different maximum depth values.

To implement the model in Python I used the DecisionTreeClassifier class from the scikit-learn-package [5]. The code to implement this can be found in the appendix. As the metrics for the validation error and to assess the model performance I used the same metrics as in the logistic regression.

4. Results

The average training accuracies, validation accuracies, training F1-scores, and validation F1-scores of the models obtained from the 5-fold cross validation can be seen in Table 2.

Table 2. The results of the models

Metric	Logistic Regression	Decision Tree
Mean training accuracy	0.65990	0.70542
Mean validation accuracy	0.66504	0.60813
Mean training F1-score	0.64770	0.71139
Mean validation F1-score	0.47577	0.42845

From the results we can see that the decision tree performs better in every metric for the training set, but the logistic regression performs better on the validation set. This indicates that the decision tree fits better to the training data but fails to generalize on data that it has not seen before. This is a typical problem with decision trees. All in all, our results from both of the models are moderate or even poor regarding the F1-scores. However, as the logistic regression clearly performs better than the decision tree on the validation set (higher accuracy and F1-score), I decided to select it as the final chosen method.

Testing the model on statistics from 2021 PGA Tour

Now that we have selected the final method and tuned the hyperparameters, we can test this model on some unseen test data. As the goal of this report was to predict the finalists during the season 2021 of PGA Tour, it is natural to use this data as our testing data. As the test accuracy we get **0.6209** and as the test F1-score we obtain **0.5252**.

5. Conclusion

In this report I presented the machine learning problem of predicting the finalists in PGA Tour based on the off the tee statistics obtained from six different seasons, including the season to be predicted. To solve this problem I used two different classification methods, logistic regression and decision tree. From these two methods logistic regression proved to be the better choice based on the validation errors of the models. As the final test metrics we got 0.6209 accuracy and 0.5252 F1-score.

Looking at the obtained test errors it is quite obvious that there is a lot of room for improvement as now our model seems to perform just slightly better than tossing a coin. However, in logistic regression the gap between the training error and the validation error was not that bad, which means that the chosen model doesn't seem to overfit the data. Therefore, using more historical data or finding more off the tee shot statisctics could improve the model performance. It is also possible that one just cannot predict the finalists using only off the tee statistics and the model needs additional data from other areas of the game, such as putting. Based on the results obtained from this report this seems to be a highly possible hypothesis.

6. References

[1] https://www.pgatour.com/fedexcup/fedexcup-overview.html

[2] https://www.pgatour.com/stats.html

[3] Jung, Alexander. (2022). *Machine Learning: The Basics*. mlbook.cs.aalto.fi

[4] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model._LogisticRegression.html

[5] https://scikit-learn.org/stable/modules/tree.html#tree-classification

7. Appendix

Appendix 1 - Web Scraping

[8]:
```
import pandas as pd
import numpy as np
import requests
from bs4 import BeautifulSoup
```

[2]:
```
def get_name_ranking(url, season):
    df2 = pd.DataFrame(columns=["Name", "Season", "Ranking"])
    response = requests.get(url)
    content = response.content
    soup = BeautifulSoup(content, "html.parser")
    table = soup.find(id="statsTable")
    rows = table.find_all("tr")
    names = []
    for row in rows[1:126]:
        name = row.find("a").text.strip()
        ranking = int(row.find("td").text.strip())
        names.append((ranking, name))
    names = np.array(names)
    df2["Name"] = names[:, 1]
    df2["Season"] = season
    df2["Ranking"] = names[:, 0]
    return df2[:125]
```

[3]:
```
def scrape_data(url, header, df, season):
    response = requests.get(url)
    content = response.content
    soup = BeautifulSoup(content, "html.parser")
    table = soup.find(id="statsTable")
    rows = table.find_all("tr")
    values = []
    for row in rows[1:]:
        name = row.find("a").text.strip()
        value = float(row.find_all("td")[4].text.strip().replace(",", ""))
        values.append((name, season, value))
    df2 = pd.DataFrame(values, columns=["Name", "Season", header])
    result = pd.merge(left=df, right=df2, on=["Name", "Season"], how="left")
    return result
```

[4]:
```
df = pd.DataFrame(columns=["Name", "Season", "Ranking"])
seasons = ["2016", "2017", "2018", "2019", "2020", "2021"]
for season in seasons:
    url = f"https://www.pgatour.com/stats/stat.02671.y{season}.html"
    df2 = get_name_ranking(url, season)

    # Driving Distance
    url_driving = f"https://www.pgatour.com/stats/stat.101.y{season}.eoff.t013.
↪html"
    header = "Driving Distance"
    df2 = scrape_data(url_driving, header, df2, season)

    # Driving Accuracy
    url_accuracy = f"https://www.pgatour.com/stats/stat.102.y{season}.eoff.t013.
↪html"
    header = "Driving Accuracy"
    df2 = scrape_data(url_accuracy, header, df2, season)

    # Club Head Speed
    url_club = f"https://www.pgatour.com/stats/stat.02401.y{season}.eoff.t013.
↪html"
    header = "Club Head Speed"
    df2 = scrape_data(url_club, header, df2, season)

    # Ball Speed
    url_ball = f"https://www.pgatour.com/stats/stat.02402.y{season}.eoff.t013.
↪html"
    header = "Ball Speed"
    df2 = scrape_data(url_ball, header, df2, season)

    # Spin Rate
    url_spin = f"https://www.pgatour.com/stats/stat.02405.y{season}.eoff.t013.
↪html"
    header = "Spin Rate"
    df2 = scrape_data(url_spin, header, df2, season)

    df = pd.concat((df, df2), ignore_index=True)
```

[7]:
```
df = df.dropna()
df.loc[df["Ranking"].astype(int) <= 30, "Eligible"] = 1
df.loc[df["Ranking"].astype(int) > 30, "Eligible"] = -1
df
```

Appendix 2 - Data Augmentation

[1]:
```
import numpy as np
```

[2]:
```
def augment_data(X, y):
    aug_X = []
    aug_y = []
    for i in range(X.shape[0]):
        if y[i] == 1:
            for _ in range(3):
                perturbations = np.random.standard_normal(5)
                aug_X.append(X[i] + perturbations)
                aug_y.append(1)
        else:
            aug_X.append(X[i])
            aug_y.append(y[i])
    return np.array(aug_X), np.array(aug_y)
```

Appendix 3 - Logistic Regression

[1]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold, train_test_split
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
```

[ ]:
```
# Testing data set
X_test = df[df["Season"] == 2021].iloc[:, 3:8].to_numpy()
y_test = df[df["Season"] == 2021].iloc[:, 8].to_numpy()

# Training and validation data set
X_rem = df[df["Season"] != 2021].iloc[:, 3:8].to_numpy()
y_rem = df[df["Season"] != 2021].iloc[:, 8].to_numpy()
```

[ ]:
```
cv = KFold(n_splits=5, random_state=42, shuffle=True)

lr_validation_accuracies = []
lr_validation_f1s = []

lr_training_accuracies = []
lr_training_f1s = []

for train_index, val_index in cv.split(y_rem):
    X_train, X_val = X_rem[train_index], X_rem[val_index]
    y_train, y_val = y_rem[train_index], y_rem[val_index]

    # AUGMENT THE TRAINING DATA
    X_train_aug, y_train_aug = augment_data(X_train, y_train)

    # STANDARDIZE
    scaler = StandardScaler()
    X_train_final = scaler.fit_transform(X_train_aug)

    # MACHINE LEARNING MODEL
    lr = LogisticRegression()
    lr.fit(X_train_final, y_train_aug)

    # TRAINING ERROR
```

```
    y_pred_train = lr.predict(X_train_final)

    accuracy_train = metrics.accuracy_score(y_train_aug, y_pred_train)
    f1_train = metrics.f1_score(y_train_aug, y_pred_train)
    lr_training_accuracies.append(accuracy_train)
    lr_training_f1s.append(f1_train)

    # VALIDATION ERROR
    X_val_final = scaler.transform(X_val)
    y_pred_val = lr.predict(X_val_final)

    accuracy = metrics.accuracy_score(y_val, y_pred_val)
    f1 = metrics.f1_score(y_val, y_pred_val)
    lr_validation_accuracies.append(accuracy)
    lr_validation_f1s.append(f1)
```

Appendix 4 - Decision Tree

[1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
```

[ ]:
```
# Testing data set
X_test = df[df["Season"] == 2021].iloc[:, 3:8].to_numpy()
y_test = df[df["Season"] == 2021].iloc[:, 8].to_numpy()

# Training and validation data set
X_rem = df[df["Season"] != 2021].iloc[:, 3:8].to_numpy()
y_rem = df[df["Season"] != 2021].iloc[:, 8].to_numpy()
```

[ ]:
```
cv = KFold(n_splits=5, random_state=42, shuffle=True)

dt_validation_accuracies = []
dt_validation_f1s = []

dt_training_accuracies = []
dt_training_f1s = []

for train_index, val_index in cv.split(y_rem):
    X_train, X_val = X_rem[train_index], X_rem[val_index]
    y_train, y_val = y_rem[train_index], y_rem[val_index]

    # AUGMENT THE TRAINING DATA
    X_train_aug, y_train_aug = augment_data(X_train, y_train)

    # STANDARDIZE
    scaler = StandardScaler()
    X_train_final = scaler.fit_transform(X_train_aug)


    # MACHINE LEARNING MODEL
    dt = DecisionTreeClassifier(random_state=42, max_depth=4, criterion="gini")
```

```
dt.fit(X_train_final, y_train_aug)

# TRAINING ERROR
y_pred_train = dt.predict(X_train_final)
accuracy_train = metrics.accuracy_score(y_train_aug, y_pred_train)
f1_train = metrics.f1_score(y_train_aug, y_pred_train)

dt_training_accuracies.append(accuracy_train)
dt_training_f1s.append(f1_train)

# VALIDATION ERROR
X_val_final = scaler.transform(X_val)
y_pred_val = dt.predict(X_val_final)

accuracy = metrics.accuracy_score(y_val, y_pred_val)
f1 = metrics.f1_score(y_val, y_pred_val)

dt_validation_accuracies.append(accuracy)
dt_validation_f1s.append(f1)
```

Appendix 5 - Test set

```
[ ]: X_train_aug, y_train_aug = augment_data(X_rem, y_rem)
scaler = StandardScaler()
X_train_final = scaler.fit_transform(X_train_aug)

lr = LogisticRegression()
lr.fit(X_train_final, y_train_aug)

X_test_final = scaler.transform(X_test)
y_pred_test = lr.predict(X_test_final)

accuracy = metrics.accuracy_score(y_test, y_pred_test)
f1 = metrics.f1_score(y_test, y_pred_test)
print(f"Accuracy: {accuracy}")
print(f"F1-score: {f1}")
```