Predicting Transfer Fee For Top 250 Expensive 21st Century Football Players With Random Forest

Hyeong Kyun (Daniel) Park

August 20, 2020

Introduction

Football, or soccer, has a sports industry known for its exceptional international popularity. In the 2018/19 season, European professional football alone has generated 28.9 billion euros in revenue and is increasing steadily (Gough, 2020). Every transfer season, football clubs exchange players with astronomical sums of money; however it has especially been in recent years that numerous transfer fee records were broken. In fact, 45 out of the top 50 player transfer fee records were broken in the last decade, where the most expensive deal was in 2017 when Neymar moved from F. C. Barcelona to Paris Saint Germain for 222 million euros ("Neymar", 2017).

This project aims to predict the transfer fee of top players in the 21st century given the dataset "Top 250 Football transfers from 2000 to 2018" from Kaggle with Random Forest. First, independent variables (predictors) and the dependent variable (transfer fee) will be identified. Next, the data will be cleaned to suit model training. Finally, the model performance will be analyzed and go through hyperparameter tuning to enhance predictability.

Data

The description of the dataset is as follows:

> The dataset of top 250 most expensive football transfers from season 2000-2001 until 2018-2019. The dataset is created on 1 August 2018 and for that reason may have an incomplete list of the latest transfer window, Summer 2018.
>
> There are 4700 total rows and 10 columns in this dataset. The columns contain the following information: the name of a football player, selling team and league, the league and team where a player is sold, an estimated market value of a player, an actual value of a transfer, the position of a player and season when a transfer took place (Ghazaryan, 2018).

```
# (1) Data Import & EDA

data = pd.read_csv("../input/top-250-football-transfers-from-2000-to-2018/
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4700 entries, 0 to 4699
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Name          4700 non-null   object
 1   Position      4700 non-null   object
 2   Age           4700 non-null   int64
 3   Team_from     4700 non-null   object
 4   League_from   4700 non-null   object
 5   Team_to       4700 non-null   object
 6   League_to     4700 non-null   object
 7   Season        4700 non-null   object
 8   Market_value  3440 non-null   float64
 9   Transfer_fee  4700 non-null   int64
dtypes: float64(1), int64(2), object(7)
memory usage: 367.3+ KB
None
```

Figure 1. DataFrame info. Only market_value, transfer_fee, and age are numerical data types, and season is classified as object. Market_value has missing values.

```
data.head()
```

	Name	Position	Age	Team_from	League_from	Team_to	League_to	Season	Market_value	Transfer_fee
0	Luís Figo	Right Winger	27	FC Barcelona	LaLiga	Real Madrid	LaLiga	2000-2001	NaN	60000000
1	Hernán Crespo	Centre-Forward	25	Parma	Serie A	Lazio	Serie A	2000-2001	NaN	56810000
2	Marc Overmars	Left Winger	27	Arsenal	Premier League	FC Barcelona	LaLiga	2000-2001	NaN	40000000
3	Gabriel Batistuta	Centre-Forward	31	Fiorentina	Serie A	AS Roma	Serie A	2000-2001	NaN	36150000
4	Nicolas Anelka	Centre-Forward	21	Real Madrid	LaLiga	Paris SG	Ligue 1	2000-2001	NaN	34500000

Figure 2. DataFrame first 5 rows. Market_value is not identifiable and Season is two years combined with a dash.

```
data.describe()
```

	Age	Market_value	Transfer_fee
count	4700.000000	3.440000e+03	4.700000e+03
mean	24.338723	8.622469e+06	9.447586e+06
std	3.230809	8.795181e+06	1.043772e+07
min	0.000000	5.000000e+04	8.250000e+05
25%	22.000000	3.500000e+06	4.000000e+06
50%	24.000000	6.000000e+06	6.500000e+06
75%	27.000000	1.000000e+07	1.082000e+07
max	35.000000	1.200000e+08	2.220000e+08

Figure 3. Statistical summary of numerical columns age, market_value, and transfer_fee. Minimum age is 0, which is problematic and will be handled during data cleaning.

Method

Environment

The programming language will be Python. The following libraries will be used: numpy, pandas, matplotlib, and sklearn.

```
# (0) Import libraries

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection._split import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import validation_curve
```

Figure 4. Imported Libraries

Independent and Dependent Variables

There are many factors that can influence transfer fee, but some are more equal than others:

Player Performance

This perhaps is the most obvious one, and there are numerous subfactors to this category: physical traits, mentality and intelligence, stamina, individual statistics such as goals and assists, and so on. As influential as player performance is to their transfer fee, it is extremely difficult to generalize. Additionally, considering this study analyzes only the top 250 most expensive

football players collected each year between 2000 to 2018- the elite of the elites- and due to the limitation of the dataset, this factor will be omitted. It is without a doubt however that certain players prove to be exceptional outliers such as Cristiano Ronaldo or Lionel Messi.

Age

The specific age at which a player peaks and declines differs across positions and other individual factors such as injuries or work rate. However, a common trend in football transfers is that values decline drastically as players reach their 30s. This is included in the dataset.

Position

Certain positions are in more demand than others, and this depends highly on the time period as popular tactics and styles of football change over time. Some roles may be more highly valued in certain clubs than others depending on the individual needs. For simplicity, the study will assume that demands for certain positions will be relatively constant throughout the seasons and clubs. This is included in the dataset.

Season

Transfer fees have been climbing steadily over seasons due to market growth and general inflation. This is included in the dataset.

Market Value

Market value is representative of several factors such as those mentioned above, and most of the times there seems to be a direct relationship between market value and transfer fee. This is included in the dataset.

In sum, the study will be utilizing the following columns as predictors: Age, Season, Market_Value, and Position.

Data Cleaning

First, the row with player age 0 discovered in Figure 3 will be addressed. Upon searching for exceptionally young players, less than 15 years old, only row 236 was found.

```
print(data.loc[data['Age'] < 15])
```

```
                Name        Position  Age Team_from    League_from  Team_to  \
236  Marzouq Al-Otaibi  Centre-Forward    0    Shabab   Saudi Arabia  Ittihad

         League_to     Season  Market_value  Transfer_fee
236   Saudi Arabia  2000-2001           NaN       2000000
```

Figure 5. Row 236 with age 0 player

Marzouq Al-Otaibi, the player in question, was transferred at the age of 25 (“Marzouq Al-Otaibi”, 2014). This age value will be corrected. Next, columns Name, Team_from, League_from, Team_to, and League_to will be omitted. This is because the study assumes individual name values and clubs have little effect on the transfer fee. Additionally, the Season column will be modified into a continuous variable by defining it as the year in which each season started. Also, the missing market values will be filled in with the transfer fee in order to preserve 1000 rows of data and still be able to utilize the Market_value column. Finally, the unique Position column values will be transformed into individual binary columns to suit model training.

```
# (2) Data Cleaning

# Fixing age 0 player Marzouq Al-Otaibi to 25 according to TransferMarkt
data.loc[data['Age'] == 0, 'Age'] = 25

# Dropping relatively irrelevant columns
for drop_columns in ['Name', 'Team_from', 'League_from', 'Team_to', 'League_to']:
    data = data.drop(drop_columns, axis='columns')

# Redefining Seasons column to beginning year of season
data.Season = data.Season.str.slice(start=0, stop=4).astype(int)

# Handling NaN values of Market_value
data.Market_value.fillna(data.Transfer_fee, inplace=True)

# Changing positions into separate binary columns for machine learning
positionsArray = data.Position.unique().astype(str)
data = pd.concat((data,pd.get_dummies(data['Position'])),axis=1)
data = data.drop('Position', axis='columns')

# Check data
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4700 entries, 0 to 4699
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 4700 non-null   int64
 1   Season              4700 non-null   int64
 2   Market_value        4700 non-null   float64
 3   Transfer_fee        4700 non-null   int64
 4   Attacking Midfield  4700 non-null   uint8
 5   Central Midfield    4700 non-null   uint8
 6   Centre-Back         4700 non-null   uint8
 7   Centre-Forward      4700 non-null   uint8
 8   Defender            4700 non-null   uint8
 9   Defensive Midfield  4700 non-null   uint8
 10  Forward             4700 non-null   uint8
 11  Goalkeeper          4700 non-null   uint8
 12  Left Midfield       4700 non-null   uint8
 13  Left Winger         4700 non-null   uint8
 14  Left-Back           4700 non-null   uint8
```

```
 15  Midfielder          4700 non-null   uint8
 16  Right Midfield      4700 non-null   uint8
 17  Right Winger        4700 non-null   uint8
 18  Right-Back          4700 non-null   uint8
 19  Second Striker      4700 non-null   uint8
 20  Sweeper             4700 non-null   uint8
dtypes: float64(1), int64(3), uint8(17)
memory usage: 225.0 KB
```

	Age	Season	Market_value	Transfer_fee	Attacking Midfield	Central Midfield	Centre-Back	Centre-Forward	Defender	Defensive Midfield	...	Goalkeeper	Left Midfield	Left Winger	Left-Back	Midfielder	Right Midfield	Right Winger	Right-Back	Second Striker	Sweeper
0	27	2000	60000000.0	60000000	0	0	0	0	0	0	...	0	0	0	0	0	0	1	0	0	0
1	25	2000	56810000.0	56810000	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
2	27	2000	40000000.0	40000000	0	0	0	0	0	0	...	0	0	1	0	0	0	0	0	0	0
3	31	2000	36150000.0	36150000	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
4	21	2000	34500000.0	34500000	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 21 columns

Figure 6. Data cleaning

Machine Learning Model

In order to account for the complexity of the predictors, the random forest regressor model was chosen as this particular study is a regression question and the model is robust and reduces overfitting.

```
# (3) Machine Learning: Random Forest Regressor

X = data
y = data['Transfer_fee']

# Pick test size of 0.5 to avoid overfitting + gives better r^2 and lower rmse than 0.4
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5, random_state=42)
forest = RandomForestRegressor(random_state = 1)

# Performance on Test Set
forest.fit(X_train, y_train)
y_pred = forest.predict(X_test)

rmsd = np.sqrt(mean_squared_error(y_test, y_pred))
r2_value = r2_score(y_test, y_pred)

print(rmsd)
print(r2_value)

# Performance on Training Set
forest.fit(X_test, y_test)
y_pred = forest.predict(X_train)

rmsd = np.sqrt(mean_squared_error(y_train, y_pred))
r2_value = r2_score(y_train, y_pred)

print(rmsd)
print(r2_value)
```

```
2107411.1114350706
0.95765187610463
676182.8580960765
0.9959502583531464
```

Figure 7. Random Forest Regressor training. Root mean squared deviation and r squared for test set and root mean squared deviation and r squared for training set in that order.

The model generated an R^2 value of 0.9577 and RMSD of 2107411.1114 on the test set. In order to improve the model and reduce any overfitting, hyperparameters were tuned as the following.

Hyperparameter Tuning

```
# (4) Hyperparameter Tuning

### (4.1) Number of Estimators
num_est = np.arange(100, 1000, 50)
train_scoreNum, test_scoreNum = validation_curve(
                                RandomForestRegressor(),
                                X = X_train, y = y_train,
                                param_name = 'n_estimators',
                                param_range = num_est, cv = 3)

# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scoreNum, axis=1)
train_std = np.std(train_scoreNum, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scoreNum, axis=1)
test_std = np.std(test_scoreNum, axis=1)

# Plot mean accuracy scores for training and test sets
plt.plot(num_est, train_mean, label="Training score", color="black")
plt.plot(num_est, test_mean, label="Cross-validation score", color="dimgrey")

plt.title("Validation Curve With Random Forest")
plt.xlabel("Number Of Trees")
plt.ylabel("Accuracy Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

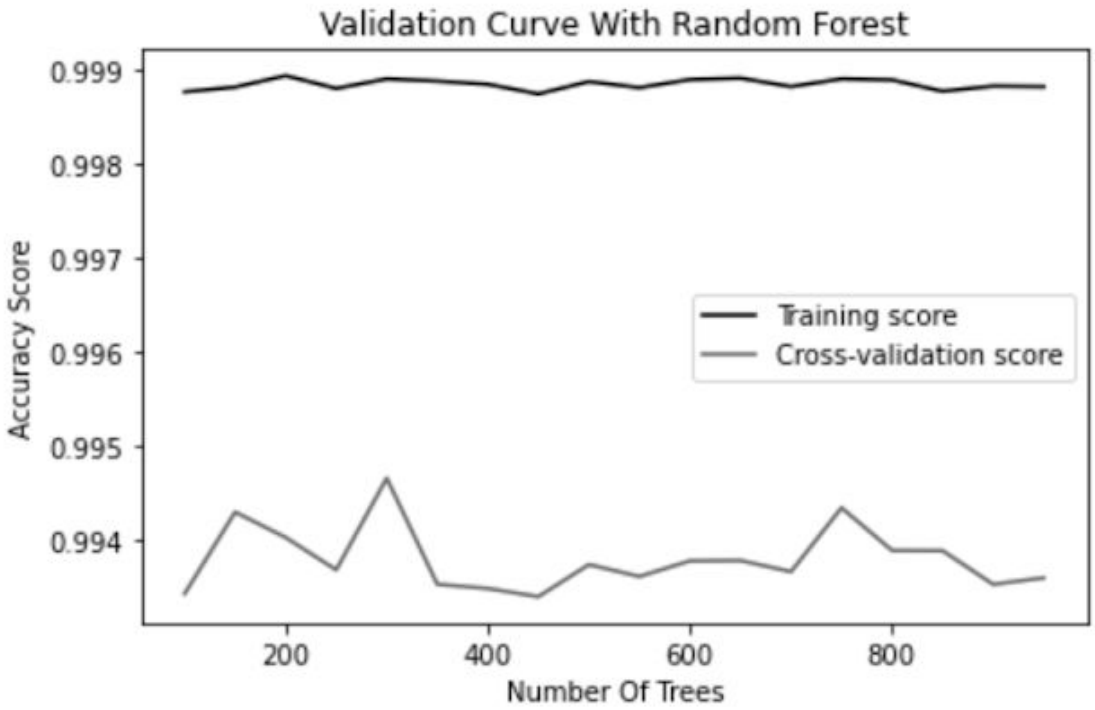

Figure 8. N_estimators parameter validation curve. Choose 300.

```
### (4.2) Max Depth
m_depth = [5, 10, 15, 20, 25, 30]
train_scoreNum, test_scoreNum = validation_curve(
                                RandomForestRegressor(),
                                X = X_train, y = y_train,
                                param_name = 'max_depth',
                                param_range = m_depth, cv = 3)

# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scoreNum, axis=1)
train_std = np.std(train_scoreNum, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scoreNum, axis=1)
test_std = np.std(test_scoreNum, axis=1)

# Plot mean accuracy scores for training and test sets
plt.plot(m_depth, train_mean, label="Training score", color="black")
plt.plot(m_depth, test_mean, label="Cross-validation score", color="dimgrey")

plt.title("Max_Depth")
plt.xlabel("Number Of Trees")
plt.ylabel("Accuracy Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()
```

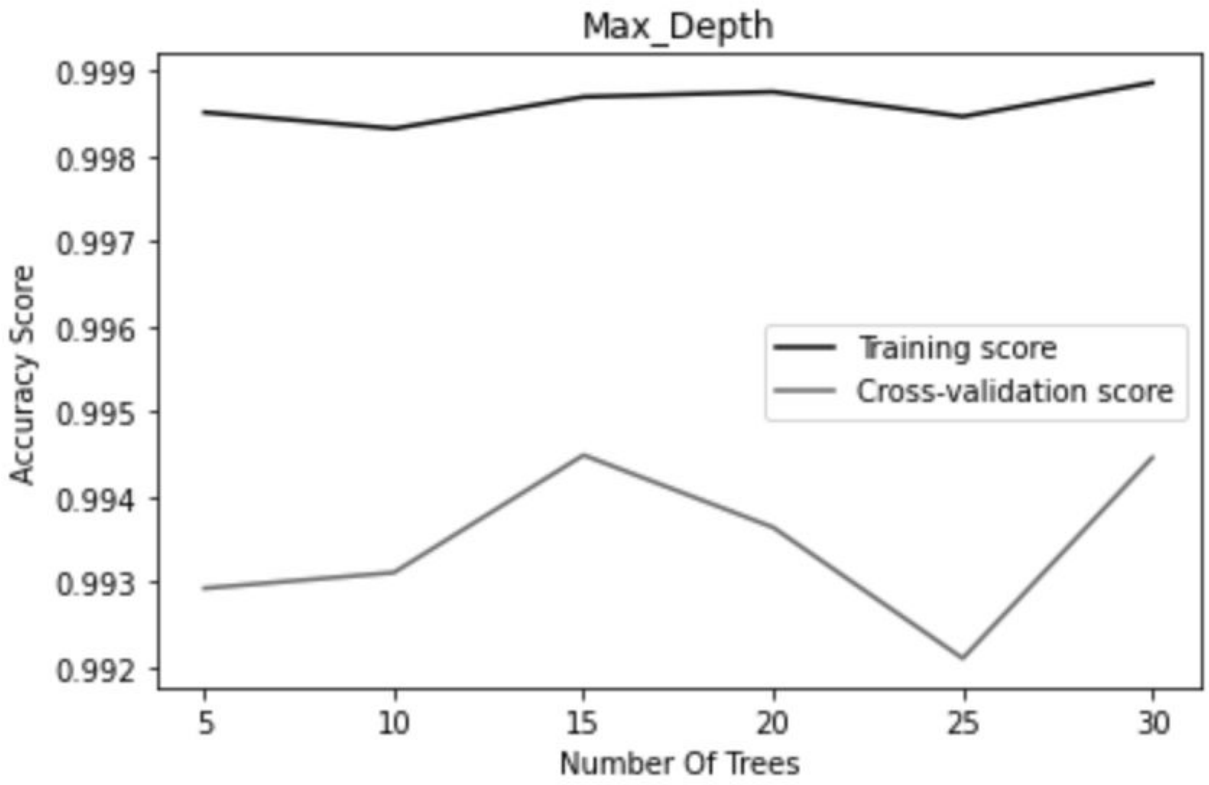

Figure 9. Max_depth parameter validation curve. Choose 15.

```
### (4.3) Minimum Samples Split
m_split = [2, 5, 10, 15, 20, 25, 30]
train_scoreNum, test_scoreNum = validation_curve(
                                RandomForestRegressor(),
                                X = X_train, y = y_train,
                                param_name = 'min_samples_split',
                                param_range = m_split, cv = 3)

# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scoreNum, axis=1)
train_std = np.std(train_scoreNum, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scoreNum, axis=1)
test_std = np.std(test_scoreNum, axis=1)

# Plot mean accuracy scores for training and test sets
plt.plot(m_split, train_mean, label="Training score", color="black")
plt.plot(m_split, test_mean, label="Cross-validation score", color="dimgrey")

plt.title("Validation Curve With Random Forest")
plt.xlabel("Min_Samples_Split")
plt.ylabel("Accuracy Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()

# Pick 2
```

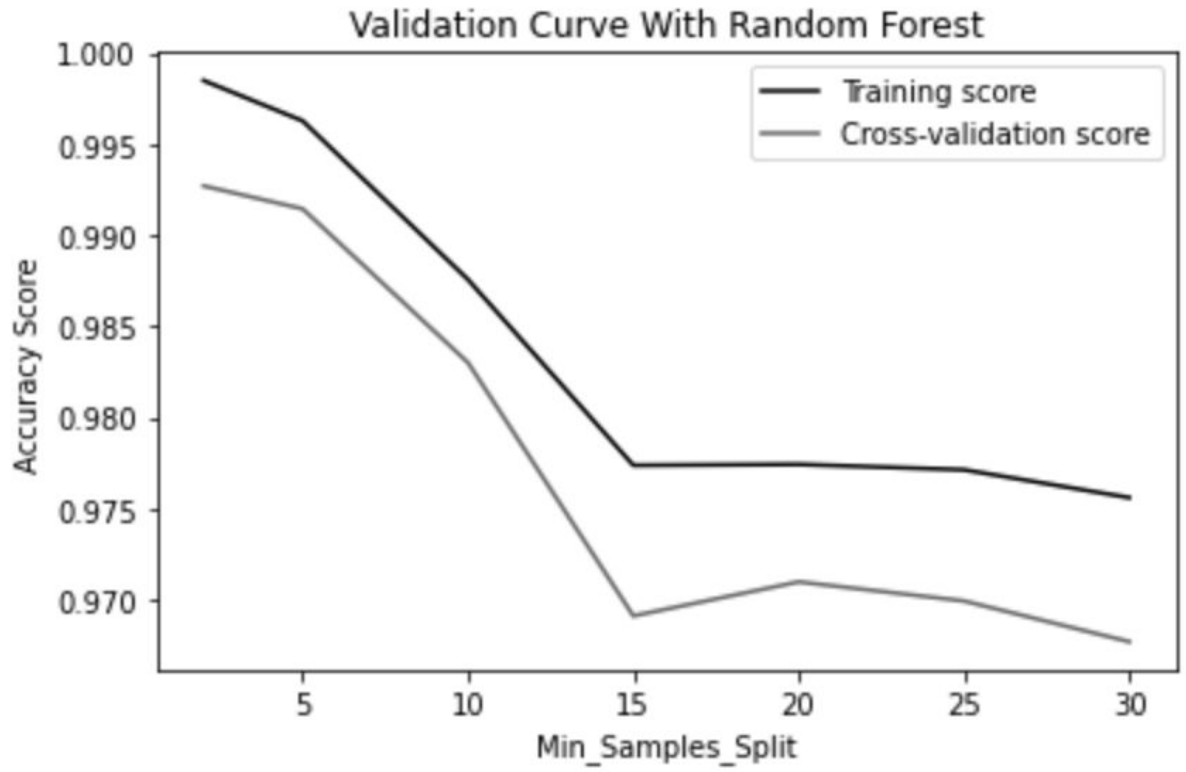

Figure 10. Min_samples_split parameter validation curve. Choose 2.

```
### (4.4) Minimum Samples Leaf
m_leaf = [1, 2, 4, 6, 8, 10]
train_scoreNum, test_scoreNum = validation_curve(
                                RandomForestRegressor(),
                                X = X_train, y = y_train,
                                param_name = 'min_samples_leaf',
                                param_range = m_leaf, cv = 3)

# Calculate mean and standard deviation for training set scores
train_mean = np.mean(train_scoreNum, axis=1)
train_std = np.std(train_scoreNum, axis=1)

# Calculate mean and standard deviation for test set scores
test_mean = np.mean(test_scoreNum, axis=1)
test_std = np.std(test_scoreNum, axis=1)

# Plot mean accuracy scores for training and test sets
plt.plot(m_leaf, train_mean, label="Training score", color="black")
plt.plot(m_leaf, test_mean, label="Cross-validation score", color="dimgrey")

plt.title("Validation Curve With Random Forest")
plt.xlabel("Min_Samples_Leaf")
plt.ylabel("Accuracy Score")
plt.tight_layout()
plt.legend(loc="best")
plt.show()

# Pick 2
```

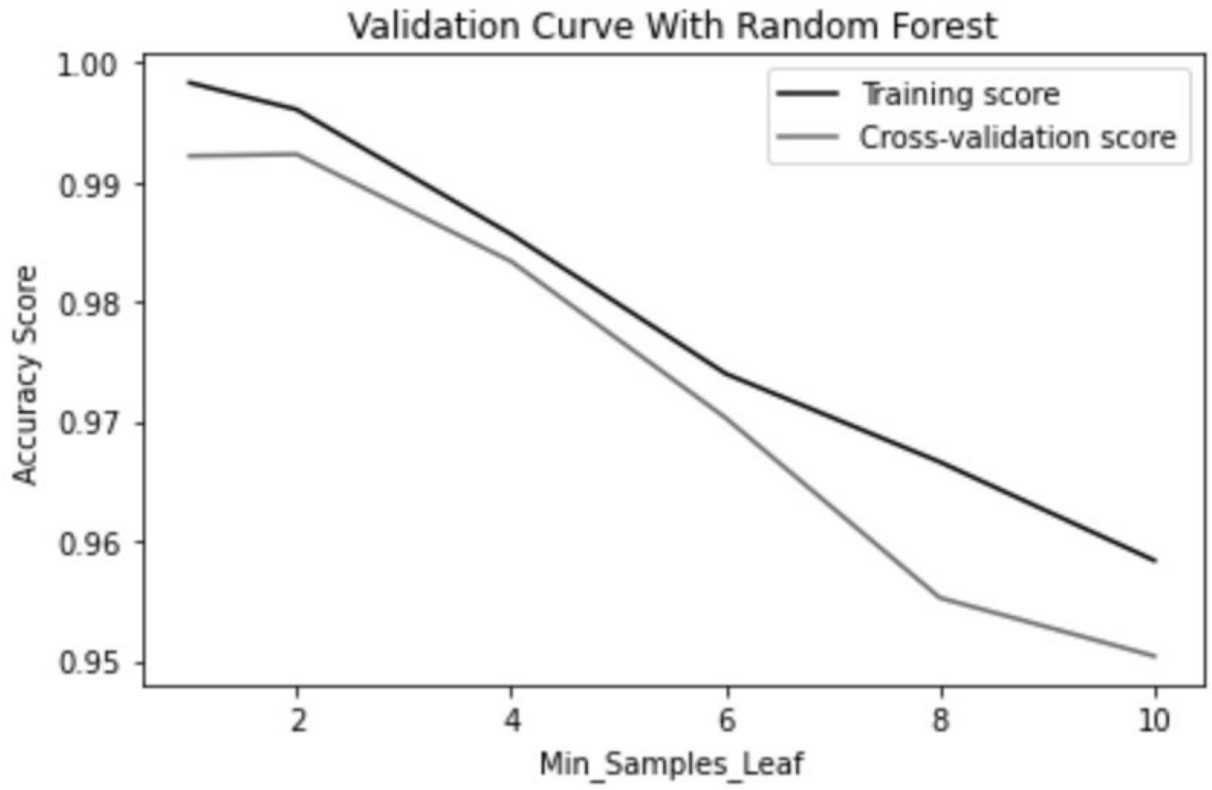

Figure 11. Min_samples_leaf parameter validation curve. Choose 2.

```
# (5) Machine Learning With Hyperparameter Tuning: Random Forest Regressor

X = data
y = data['Transfer_fee']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5, random_state=42)
forest = RandomForestRegressor(random_state = 1, n_estimators = 300, max_depth = 15, min_samples_split= 2, min_samples_leaf = 2)

# Performance on Test Set
forest.fit(X_train, y_train)
y_pred = forest.predict(X_test)

rmsd = np.sqrt(mean_squared_error(y_test, y_pred))
r2_value = r2_score(y_test, y_pred)

print(rmsd)
print(r2_value)

# Performance on Training Set
forest.fit(X_test, y_test)
y_pred = forest.predict(X_train)

rmsd = np.sqrt(mean_squared_error(y_train, y_pred))
r2_value = r2_score(y_train, y_pred)

print(rmsd)
print(r2_value)

# Pick pre-tuned model because performance on training set increases after tuning but on test set it decreases, a hint of overfit
```

```
2149572.4350867574
0.9559404748243417
424983.33740366995
0.9984002842886316
```

Figure 12. Hyperparameter Tuned Model

This model with tuned hyperparameters generated an R^2 value of 0.9559 and RMSD of 2149572.4351 on the test set. Performance decreased on the test set compared to the previous model, but it increased in the training set. As this is a hint of overfitting, in the latter model, it is best to stick with the initial random forest regressor.

Results & Discussion

This random forest regressor model predicts the transfer fees of expensive 21st century soccer players given the following predictor variables: Age, season, market value, and position.

The model demonstrates relatively high performance: an R^2 value of 0.9577 and RMSD of 2107411.1114 on the test set. Upon attempting to tune the hyperparameters in order to avoid overfitting and improve the model, it was found the initial model with default parameters performed better than the adjusted.

Due to the simplicity of the model training process and assumptions, there is much room for improvement. A limitation with a random forest regressor and other tree-based models is it cannot extrapolate, so the performance is questionable for transfers that will happen in the far future. A solution in the future could be to train other models such as linear regression or neural networks instead. Furthermore, analysis could be drawn from more transfer data outside just the top 250 and the 21st century to enhance model accuracy. More predictors such as those accounting for individual player statistics- goals, assists, physical profiles, and so on- could potentially improve the model accuracy..

References

Christina Gough. (2020, July 1). *Market size of professional football in Europe 2006-2019* Statista - The Statistics Portal. Retrieved May 10, 2016, from https://www.statista.com/statistics/261223/european-soccer-market-total-revenue/

Ghazaryan, Vardan. (2018, August 1). *Top 250 Football transfers from 2000 to 2018.* Kaggle, Version 1. Retrieved December 20, 2017, from https://www.kaggle.com/vardan95ghazaryan/top-250-football-transfers-from-2000-to-2018

Marzouq Al-Otaibi. (2014, July 1). Transfermarkt. Retrieved August 20, 2020, from https://www.transfermarkt.us/marzouq-al-otaibi/profil/spieler/28152

Neymar: Paris St-Germain sign Barcelona forward for world record 222m euros. (2017, August 3). BBCSport. Retrieved August 20, 2020, from https://www.bbc.co.uk/sport/football/40762417