

**EGE ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**YAPAY ZEKÂ YÖNTEMLERİ (3+0)**

**2021-2022 BAHAR YARIYILI**

**PROJE-2 DÖNEM RAPORU**

**TESLİM TARİHİ**

20/06/2022

**Konu:**

Futbol Ligleri Puan Tablosu Tahminleyen Yapay Zekâ Programı

**HAZIRLAYANLAR**

Çağatay Çetinkol, 05190000026

Anıl Muslu, 05190000045

# Kaynak Kodlar:

#!/usr/bin/env python  
# coding: utf-8  
  
# In[ ]:  
  
  
import pandas as pd  
import numpy as np  
  
from sklearn.metrics import accuracy\_score, f1\_score, recall\_score, precision\_score  
  
from sklearn.metrics import confusion\_matrix  
import matplotlib.pyplot as plt  
import seaborn as sn   
  
  
# ## Verisetlerinin okunarak düzenlemelerin yapılması:  
  
# In[ ]:  
  
  
data = pd.read\_csv("understat.com.csv", delimiter=",") # Ana kullanacağımız veri setimize başka bir veri setinden  
 # öznitelik eklemek istiyoruz.Bunun için 2 veri setini de okuyoruz.  
data2 = pd.read\_csv("Big\_5\_European\_football\_leagues\_teams\_stats.csv", delimiter=",")   
  
  
# In[ ]:  
  
  
data.rename(columns={"Unnamed: 0":"league"}, inplace=True) # Sutün isimlerinde gerekli düzenleme ve   
data.rename(columns={"Unnamed: 1":"year"}, inplace=True) # isimlendirme işlemlerini yapıyoruz  
data.rename(columns={"missed":"conceded"}, inplace=True)  
  
  
# In[ ]:  
  
  
pd.set\_option("display.max\_rows", None) # Veriyi daha detaylı inceleyebilmek adına tüm satır/sütunları yazdırıyoruz  
pd.set\_option("display.max\_columns", None)  
  
  
# In[ ]:  
  
  
data=data[data.league != "RFPL"] # Verimizdeki Rusya futbol ligini çıkarıp, Avrupanın 5 büyük ligini ele alıyoruz.  
  
  
# In[ ]:  
  
  
data = data.drop(columns="wins") # Gerekli olmayan öznitelikleri çıkarıyoruz  
data = data.drop(columns="draws")  
data = data.drop(columns="loses")  
  
  
# 2. verisetimiz ile ilgili işlemler:  
  
# In[ ]:  
  
  
data2=data2[data2.season.isin(["2014-2015", "2015-2016", "2016-2017", "2017-2018", "2018-2019", "2019-2020"])]  
  
  
# In[ ]:  
  
  
data2.rename(columns = {"season":"year", "squad":"team", "competition":"league"}, inplace=True)  
  
  
# In[ ]:  
  
  
data2["year"].replace({"2014-2015":2015, "2015-2016":2016, "2016-2017":2017, "2017-2018":2018,   
 "2018-2019":2019, "2019-2020":2020}, inplace=True)  
  
data2["league"].replace({"Premier League":"EPL", "Ligue 1":"Ligue\_1", "Fußball-Bundesliga":"Bundesliga",   
 "Serie A":"Serie\_A", "La Liga":"La\_liga"}, inplace=True)  
  
  
# In[ ]:  
  
  
data2=data2.loc[:,["league","year", "team", "goal\_diff", "assists", "pens\_made", "pens\_att", "shots\_on\_target"  
 , "shots\_on\_target\_against", "saves", "clean\_sheets", "players\_used", "games\_subs",   
 "points\_per\_match"]]  
  
  
# 2. verisetinde hataya sebep olan aynı takımın farklı isimlendirilmesi durumlarını düzeltiyoruz:  
  
# In[ ]:  
  
  
data2["team"].replace({"Málaga":"Malaga", "La Coruña":"Deportivo La Coruna", "Almería":"Almeria",   
 "Córdoba":"Cordoba", "Atlético Madrid":"Atletico Madrid", "Betis":"Real Betis",   
 "Sporting Gijón":"Sporting Gijon", "Alavés":"Alaves", "Leganés":"Leganes",   
 "Valladolid":"Real Valladolid", "Huesca":"SD Huesca"}, inplace=True) # La liga isim düzenlemeleri  
  
data2["team"].replace({"Manchester Utd":"Manchester United", "Swansea City":"Swansea", "Stoke City":"Stoke",   
 "West Brom":"West Bromwich Albion", "Leicester City":"Leicester",   
 "Newcastle Utd":"Newcastle United", "Hull City":"Hull",   
 "QPR":"Queens Park Rangers", "Norwich City":"Norwich",   
 "Wolves":"Wolverhampton Wanderers", "Cardiff City":"Cardiff",   
 "Sheffield Utd":"Sheffield United"}, inplace=True) # EPL isim düzenlemeleri  
  
data2["team"].replace({"Paris S-G":"Paris Saint Germain", "Saint-Étienne":"Saint-Etienne",   
 "Bastia":"SC Bastia", "Evian":"Evian Thonon Gailland",   
 "Gazélec Ajaccio":"GFC Ajaccio", "Nîmes":"Nimes"}, inplace=True) # Ligue-1 isim düzenlemeleri  
  
data2["team"].replace({"M'Gladbach":"Borussia M.Gladbach", "Leverkusen":"Bayer Leverkusen",   
 "Dortmund":"Borussia Dortmund", "Eint Frankfurt":"Eintracht Frankfurt",   
 "Köln":"FC Cologne", "Stuttgart":"VfB Stuttgart", "Hertha BSC":"Hertha Berlin",   
 "Paderborn 07":"Paderborn", "RB Leipzig":"RasenBallsport Leipzig",  
 "Düsseldorf":"Fortuna Duesseldorf", "Nürnberg":"Nuernberg"}, inplace=True) # Bundesliga isim düzenlemeleri  
  
data2["team"].replace({"Milan":"AC Milan", "Hellas Verona":"Verona", "SPAL":"SPAL 2013",   
 "Parma":"Parma Calcio 1913"}, inplace=True) # Serie-A isim düzenlemeleri  
  
  
# ## Ana verisetimiz ile 2. verisetimizi birleştiriyoruz:  
  
# In[ ]:  
  
  
# 2 verisetindeki aynı yıla denk gelen ama farklı ifade edilmiş olan sezonları denkleştiriyoruz:  
data["year"].replace({2014:2015, 2015:2016, 2016:2017, 2017:2018, 2018:2019, 2019:2020}, inplace=True)   
  
  
# In[ ]:  
  
  
data=pd.merge(data, data2, on=["year", "team"]) # Daha kapsamlı olan yeni verisetimizi elde ediyoruz.  
  
  
# In[ ]:  
  
  
data.rename(columns={"league\_x":"league"}, inplace=True)  
  
  
# In[ ]:  
  
  
#Verisetimizi amacımıza uygun şekilde gerekli alt verisetlerine bölüyoruz:  
  
data\_int = data.iloc[:,[1,2,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,22,23,24,25,26,27,28,29,30,31,32]]   
  
data\_x = data\_int.drop(columns="pts")  
  
train\_data = data\_x.loc[(data['year'] >= 2015) & (data['year'] <= 2019)]  
test\_data = data\_x.loc[(data['year'] == 2020)]  
  
train\_target = data\_int.loc[(data['year'] >= 2015) & (data['year'] <= 2019)].pts  
test\_target = data\_int.loc[(data['year'] == 2020) ].pts  
  
  
# In[ ]:  
  
  
data  
  
  
# ## "Scale" işlemleri için gerekli tanımlamalar:  
  
# In[ ]:  
  
  
from sklearn.preprocessing import StandardScaler  
  
  
# In[ ]:  
  
  
sc = StandardScaler()  
  
scaled\_data = sc.fit\_transform(data\_int)  
  
  
# In[ ]:  
  
  
scaled\_train\_data = sc.fit\_transform(train\_data)  
scaled\_test\_data = sc.transform(test\_data)  
  
  
# # "MLP" Classifier  
  
# In[ ]:  
  
  
from sklearn.neural\_network import MLPClassifier  
  
  
# In[ ]:  
  
  
model\_mlp = MLPClassifier(max\_iter=10000, solver='adam', random\_state=5)  
  
model\_mlp.fit(train\_data,train\_target)  
  
  
# In[ ]:  
  
  
mlp\_result = model\_mlp.predict(test\_data) #Modelimizin tahminlemesine göz atıyoruz.  
  
mlp\_result  
  
  
# In[ ]:  
  
  
  
  
  
# In[ ]:  
  
  
laliga\_mlp\_result = mlp\_result[:20] #Elde ettiğimiz sonuçları liglere göre ayırıyoruz.  
epl\_mlp\_result = mlp\_result[20:40]  
bundesliga\_mlp\_result = mlp\_result[40:58]  
serieA\_mlp\_result = mlp\_result[58:78]  
ligue1\_mlp\_result = mlp\_result[78:98]  
  
  
# In[ ]:  
  
  
laliga\_teams = data.loc[(data['year'] == 2020 ) & (data['league'] == "La\_liga")].team # Her ligden amacımıza yönelik gerekli  
laliga\_real\_points = data.loc[(data["year"]==2020) & (data["league"]=="La\_liga")].pts # verileri ayırıyoruz.  
laliga\_real\_positions=data.loc[(data["year"]==2020) & (data["league"]=="La\_liga")]["position"]  
  
epl\_teams = data.loc[(data['year'] == 2020 ) & (data['league'] == "EPL")].team  
epl\_real\_points = data.loc[(data["year"]==2020) & (data["league"]=="EPL")].pts  
epl\_real\_positions=data.loc[(data["year"]==2020) & (data["league"]=="EPL")]["position"]  
  
bundesliga\_teams = data.loc[(data['year'] == 2020 ) & (data['league'] == "Bundesliga")].team  
bundesliga\_real\_points = data.loc[(data["year"]==2020) & (data["league"]=="Bundesliga")].pts  
bundesliga\_real\_positions=data.loc[(data["year"]==2020) & (data["league"]=="Bundesliga")]["position"]  
  
serieA\_teams = data.loc[(data['year'] == 2020 ) & (data['league'] == "Serie\_A")].team  
serieA\_real\_points = data.loc[(data["year"]==2020) & (data["league"]=="Serie\_A")].pts  
serieA\_real\_positions=data.loc[(data["year"]==2020) & (data["league"]=="Serie\_A")]["position"]  
  
ligue1\_teams = data.loc[(data['year'] == 2020 ) & (data['league'] == "Ligue\_1")].team  
ligue1\_real\_points = data.loc[(data["year"]==2020) & (data["league"]=="Ligue\_1")].pts  
ligue1\_real\_positions=data.loc[(data["year"]==2020) & (data["league"]=="Ligue\_1")]["position"]   
  
  
# In[ ]:  
  
  
# Tahminlenen ve gerçekte gözlemlenen sonuçlar arasındaki benzerlikleri görebilmek adına yeni bir dataframe oluşturuyoruz.  
# Bu yöntemde Serie-A ligini ele alıyoruz  
  
serieA\_mlp\_df={"team": serieA\_teams, "PredictedPoints":serieA\_mlp\_result,   
 "RealPoints":serieA\_real\_points, "RealPositions":serieA\_real\_positions}   
  
serieA\_mlp\_df=pd.DataFrame(serieA\_mlp\_df)  
  
serieA\_mlp\_df.sort\_values(by="PredictedPoints", ascending=False, inplace=True)  
  
serieA\_mlp\_df.reset\_index(inplace=True)  
serieA\_mlp\_df.drop(columns="index", inplace=True)  
  
  
# In[ ]:  
  
  
list1=[0]\*20  
for i in range(20):  
 list1[i]=i+1  
list1  
  
  
# In[ ]:  
  
  
serieA\_mlp\_df.insert(0, "PredictedPosition", list1)  
  
  
# In[ ]:  
  
  
serieA\_mlp\_df # Yeni oluşturduğumuz tahminlenen ve gerçekte olan sonuçları gösteren dataframe'imize göz atıyoruz.  
  
  
# In[ ]:  
  
  
# Serie-A için MLP modelin puanlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(serieA\_real\_points, serieA\_mlp\_result), recall\_score(serieA\_real\_points, serieA\_mlp\_result, average="micro"), precision\_score(serieA\_real\_points, serieA\_mlp\_result, average="micro")  
  
  
# In[ ]:  
  
  
# Seria-A için MLP modelin pozisyonlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(serieA\_mlp\_df["RealPositions"], serieA\_mlp\_df["PredictedPosition"] ), recall\_score(serieA\_mlp\_df["RealPositions"], serieA\_mlp\_df["PredictedPosition"], average="micro"), precision\_score(serieA\_mlp\_df["RealPositions"], serieA\_mlp\_df["PredictedPosition"], average="micro")  
  
  
# In[ ]:  
  
  
# Tüm ligler için MLP modelin puanlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(test\_target, mlp\_result), recall\_score(test\_target, mlp\_result, average="micro"), precision\_score(test\_target, mlp\_result, average="micro")  
  
  
# In[ ]:  
  
  
cm\_mlp = confusion\_matrix(serieA\_mlp\_df["RealPositions"], serieA\_mlp\_df["PredictedPosition"], labels=np.arange(0,10))  
plt.figure(figsize=(10,7))  
sn.heatmap(cm\_mlp, annot=True, fmt='d')  
plt.xlabel('Predicted')  
plt.ylabel('Actual')  
  
  
# ## Scale edilmiş verilerle MLP Classifier  
  
# In[ ]:  
  
  
scaled\_mlp = MLPClassifier(max\_iter=10000,solver="adam", random\_state=5)  
  
scaled\_mlp.fit(scaled\_train\_data, train\_target)  
  
  
# In[ ]:  
  
  
scaled\_mlp\_result = scaled\_mlp.predict(scaled\_test\_data)  
  
scaled\_mlp\_result #Modelimizin tahminlemesine göz atıyoruz.  
  
  
# In[ ]:  
  
  
laliga\_sc\_mlp\_result = scaled\_mlp\_result[:20] #Elde ettiğimiz sonuçları liglere göre ayırıyoruz.  
epl\_sc\_mlp\_result = scaled\_mlp\_result[20:40]  
bundesliga\_sc\_mlp\_result = scaled\_mlp\_result[40:58]  
serieA\_sc\_mlp\_result = scaled\_mlp\_result[58:78]  
ligue1\_sc\_mlp\_result = scaled\_mlp\_result[78:98]  
  
  
# In[ ]:  
  
  
# Tahminlenen ve gerçekte gözlemlenen sonuçlar arasındaki benzerlikleri görebilmek adına yeni bir dataframe oluşturuyoruz.  
# Bu yöntemde Bundesliga ligini ele alıyoruz  
  
bundesliga\_sc\_mlp\_df={"team": bundesliga\_teams, "PredictedPoints":bundesliga\_sc\_mlp\_result,   
 "RealPoints":bundesliga\_real\_points, "RealPositions":bundesliga\_real\_positions}   
  
bundesliga\_sc\_mlp\_df=pd.DataFrame(bundesliga\_sc\_mlp\_df)  
  
bundesliga\_sc\_mlp\_df.sort\_values(by="PredictedPoints", ascending=False, inplace=True)  
  
bundesliga\_sc\_mlp\_df.reset\_index(inplace=True)  
bundesliga\_sc\_mlp\_df.drop(columns="index", inplace=True)  
  
  
# In[ ]:  
  
  
list2=[0]\*18  
for i in range(18):  
 list2[i]=i+1  
list2  
  
  
# In[ ]:  
  
  
bundesliga\_sc\_mlp\_df.insert(0, "PredictedPosition", list2)  
  
  
# In[ ]:  
  
  
bundesliga\_sc\_mlp\_df # Yeni oluşturduğumuz tahminlenen ve gerçekte olan sonuçları gösteren dataframe'imize göz atıyoruz.  
  
  
# In[ ]:  
  
  
# Bundesliga için Scaled MLP modelin puanlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(bundesliga\_real\_points, bundesliga\_sc\_mlp\_result), recall\_score(bundesliga\_real\_points, bundesliga\_sc\_mlp\_result, average="micro"), precision\_score(bundesliga\_real\_points, bundesliga\_sc\_mlp\_result, average="micro")  
  
  
# In[ ]:  
  
  
# Bundesliga için Scaled MLP modelin pozisyonlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(bundesliga\_sc\_mlp\_df["RealPositions"], bundesliga\_sc\_mlp\_df["PredictedPosition"] ), recall\_score(bundesliga\_sc\_mlp\_df["RealPositions"], bundesliga\_sc\_mlp\_df["PredictedPosition"], average="micro"), precision\_score(bundesliga\_sc\_mlp\_df["RealPositions"], bundesliga\_sc\_mlp\_df["PredictedPosition"], average="micro")  
  
  
# In[ ]:  
  
  
# Tüm ligler için Scaled MLP modelin doğruluk değerlerinin ölçülmesi  
accuracy\_score(test\_target, scaled\_mlp\_result), recall\_score(test\_target, scaled\_mlp\_result, average="micro"), precision\_score(test\_target, scaled\_mlp\_result, average="micro")  
  
  
# In[ ]:  
  
  
cm\_mlp = confusion\_matrix(bundesliga\_sc\_mlp\_df["RealPositions"], bundesliga\_sc\_mlp\_df["PredictedPosition"], labels=np.arange(0,10))  
plt.figure(figsize=(10,7))  
sn.heatmap(cm\_mlp, annot=True, fmt='d')  
plt.xlabel('Predicted')  
plt.ylabel('Actual')  
  
  
# # Random Forest  
  
# In[ ]:  
  
  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.model\_selection import cross\_validate  
  
  
# In[ ]:  
  
  
model\_forest = RandomForestClassifier()  
model\_forest.fit(train\_data, train\_target)  
  
  
# In[ ]:  
  
  
forest\_result = model\_forest.predict(test\_data) # Modelimizin tahminlemesine göz atıyoruz  
  
forest\_result  
  
  
# In[ ]:  
  
  
laliga\_rf\_result = forest\_result[:20] #Elde ettiğimiz sonuçları liglere göre ayırıyoruz.  
epl\_rf\_result = forest\_result[20:40]  
bundesliga\_rf\_result = forest\_result[40:58]  
serieA\_rf\_result = forest\_result[58:78]  
ligue1\_rf\_result = forest\_result[78:98]  
  
  
# In[ ]:  
  
  
# Tahminlenen ve gerçekte gözlemlenen sonuçlar arasındaki benzerlikleri görebilmek adına yeni bir dataframe oluşturuyoruz.  
# Bu yöntemde EPL ligini ele alıyoruz  
  
epl\_rf\_df={"team": epl\_teams, "PredictedPoints":epl\_rf\_result,   
 "RealPoints":epl\_real\_points, "RealPositions":epl\_real\_positions}   
  
epl\_rf\_df=pd.DataFrame(epl\_rf\_df)  
  
epl\_rf\_df.sort\_values(by="PredictedPoints", ascending=False, inplace=True)  
  
epl\_rf\_df.reset\_index(inplace=True)  
epl\_rf\_df.drop(columns="index", inplace=True)  
  
  
# In[ ]:  
  
  
list1=[0]\*20  
for i in range(20):  
 list1[i]=i+1  
list1  
  
  
# In[ ]:  
  
  
epl\_rf\_df.insert(0, "PredictedPosition", list1)  
  
  
# In[ ]:  
  
  
epl\_rf\_df # Yeni oluşturduğumuz tahminlenen ve gerçekte olan sonuçları gösteren dataframe'imize göz atıyoruz.  
  
  
# In[ ]:  
  
  
# EPL için Random Forest modelin puanlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(epl\_real\_points, epl\_rf\_result), recall\_score(epl\_real\_points, epl\_rf\_result, average="micro"), precision\_score(epl\_real\_points, epl\_rf\_result, average="micro")  
  
  
# In[ ]:  
  
  
# EPL için Random Forest modelin pozisyonlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(epl\_rf\_df["RealPositions"], epl\_rf\_df["PredictedPosition"] ), recall\_score(epl\_rf\_df["RealPositions"], epl\_rf\_df["PredictedPosition"], average="micro"), precision\_score(epl\_rf\_df["RealPositions"], epl\_rf\_df["PredictedPosition"], average="micro")  
  
  
# In[ ]:  
  
  
# Tüm ligler için Random Forest modelin puanlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(test\_target, forest\_result), recall\_score(test\_target, forest\_result, average="micro"), precision\_score(test\_target, forest\_result, average="micro")  
  
  
# In[ ]:  
  
  
cm\_mlp = confusion\_matrix(epl\_rf\_df["RealPositions"], epl\_rf\_df["PredictedPosition"], labels=np.arange(0,10))  
plt.figure(figsize=(10,7))  
sn.heatmap(cm\_mlp, annot=True, fmt='d')  
plt.xlabel('Predicted')  
plt.ylabel('Actual')  
  
  
# ## Random Forest with Cross Validation  
  
# In[ ]:  
  
  
model\_forest\_cross = RandomForestClassifier()  
cross\_validate(model\_forest\_cross, train\_data, train\_target, cv=10, scoring=['accuracy', 'recall\_micro', 'precision\_weighted'], n\_jobs=10)  
  
  
# ## Scale edilmiş verilerle "Random Forest"   
  
# In[ ]:  
  
  
scaled\_model\_forest = RandomForestClassifier(criterion='entropy')  
scaled\_model\_forest.fit(scaled\_train\_data, train\_target)  
  
  
# In[ ]:  
  
  
scaled\_rf\_result = scaled\_model\_forest.predict(scaled\_test\_data)  
  
scaled\_rf\_result # Modelimizin tahminlemesine göz atıyoruz.  
  
  
# In[ ]:  
  
  
laliga\_sc\_rf\_result = scaled\_rf\_result[:20] #Elde ettiğimiz sonuçları liglere göre ayırıyoruz.  
epl\_sc\_rf\_result = scaled\_rf\_result[20:40]  
bundesliga\_sc\_rf\_result = scaled\_rf\_result[40:58]  
serieA\_sc\_rf\_result = scaled\_rf\_result[58:78]  
ligue1\_sc\_rf\_result = scaled\_rf\_result[78:98]  
  
  
# In[ ]:  
  
  
# Tahminlenen ve gerçekte gözlemlenen sonuçlar arasındaki benzerlikleri görebilmek adına yeni bir dataframe oluşturuyoruz.  
# Bu yöntemde La liga ligini ele alıyoruz  
  
laliga\_sc\_rf\_df={"team": laliga\_teams, "PredictedPoints":laliga\_sc\_rf\_result,   
 "RealPoints":laliga\_real\_points, "RealPositions":laliga\_real\_positions}   
  
laliga\_sc\_rf\_df=pd.DataFrame(laliga\_sc\_rf\_df)  
  
laliga\_sc\_rf\_df.sort\_values(by="PredictedPoints", ascending=False, inplace=True)  
  
laliga\_sc\_rf\_df.reset\_index(inplace=True)  
laliga\_sc\_rf\_df.drop(columns="index", inplace=True)  
  
  
# In[ ]:  
  
  
list1=[0]\*20  
for i in range(20):  
 list1[i]=i+1  
list1  
  
  
# In[ ]:  
  
  
laliga\_sc\_rf\_df.insert(0, "PredictedPosition", list1)  
  
  
# In[ ]:  
  
  
laliga\_sc\_rf\_df # Yeni oluşturduğumuz tahminlenen ve gerçekte olan sonuçları gösteren dataframe'imize göz atıyoruz.  
  
  
# In[ ]:  
  
  
# La liga için Scaled Random Forest modelin puanlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(laliga\_real\_points, laliga\_sc\_rf\_result), recall\_score(laliga\_real\_points, laliga\_sc\_rf\_result, average="micro"), precision\_score(laliga\_real\_points, laliga\_sc\_rf\_result, average="micro")  
  
  
# In[ ]:  
  
  
# La liga için Scaled Random Forest modelin pozisyonlara göre doğruluk değerlerinin ölçülmesi  
accuracy\_score(laliga\_sc\_rf\_df["RealPositions"], laliga\_sc\_rf\_df["PredictedPosition"] ), recall\_score(laliga\_sc\_rf\_df["RealPositions"], laliga\_sc\_rf\_df["PredictedPosition"], average="micro"), precision\_score(laliga\_sc\_rf\_df["RealPositions"], laliga\_sc\_rf\_df["PredictedPosition"], average="micro")  
  
  
# In[ ]:  
  
  
# Tüm ligler için Scaled Random Forest modelin doğruluk değerlerinin ölçülmesi  
accuracy\_score(test\_target, scaled\_rf\_result), recall\_score(test\_target, scaled\_rf\_result, average="micro"), precision\_score(test\_target, scaled\_rf\_result, average="micro")  
  
  
# In[ ]:  
  
  
cm\_mlp = confusion\_matrix(laliga\_sc\_rf\_df["RealPositions"], laliga\_sc\_rf\_df["PredictedPosition"], labels=np.arange(0,10))  
plt.figure(figsize=(10,7))  
sn.heatmap(cm\_mlp, annot=True, fmt='d')  
plt.xlabel('Predicted')  
plt.ylabel('Actual')  
  
  
# ## Random Forest yönteminin normalizasyon yöntemiyle denenmesi:  
  
# ### "Min Max Scaler" ile Random Forest yöntemi:  
  
# In[ ]:  
  
  
from sklearn.preprocessing import MinMaxScaler  
  
min\_sc = MinMaxScaler()  
m\_sc\_train\_data = min\_sc.fit\_transform(train\_data)  
m\_sc\_test\_data = min\_sc.transform(test\_data)  
  
  
# In[ ]:  
  
  
diff\_forest = RandomForestClassifier(n\_jobs=10)  
diff\_forest.fit(m\_sc\_train\_data, train\_target)  
diff\_forest\_result = diff\_forest.predict(m\_sc\_test\_data)  
  
  
# In[ ]:  
  
  
accuracy\_score(test\_target, diff\_forest\_result), recall\_score(test\_target, diff\_forest\_result, average="micro"), precision\_score(test\_target, diff\_forest\_result, average="weighted")  
  
  
# In[ ]: